SYBASE®
An SAP® Company

DB-Library™/C リファレンス・マニュアル

Open Client™

15.7

ドキュメント ID：DC32605-01-01-1570-01

改訂：2012 年 4 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

はじめに

このマニュアルは、DB-Library™ の C バージョンのリファレンス情報について説明しています。

対象読者

このマニュアルは、DB-Library アプリケーションを作成するプログラマ向けのリファレンス・マニュアルです。このマニュアルは、C プログラミング言語に精通したアプリケーション・プログラマを対象としています。

このマニュアルの内容

このマニュアルには、以下の章があります。

- 「第 1 章 DB-Library の概要」では、DB-Library の概要について説明します。
- 「第 2 章 ルーチン」では、指定できるパラメータや戻り値など、個々の DB-Library ルーチンに固有の情報について説明します。
- 「第 3 章 バルク・コピー・ルーチン」では、バルク・コピーの概要および個々のバルク・コピー・ルーチンに固有の情報について説明します。
- 「第 4 章 2 フェーズ・コミット・サービス」では、2 フェーズ・コミット・サービスの概要および個々の 2 フェーズ・コミット・サービス・ルーチンに固有の情報について説明します。
- 「付録 A カーソル」では、DB-Library のカーソル・ルーチンについて説明します。
- 「付録 B DB-Library エラー・メッセージ」では、DB-Library エラー・メッセージに関する情報について説明します。

関連マニュアル

詳細については、これらのマニュアルを参照できます。

- 『Open Server および SDK 新機能』(各 Windows、Linux、UNIX 版) では、Open Server と Software Developer's Kit の新機能について説明しています。このマニュアルは、新機能の提供に伴って改訂されます。
- 使用しているプラットフォームの Open Server の『リリース・ノート』には、Open Server に関する重要な最新情報が記載されています。

- 使用しているプラットフォームの『Software Developer's Kit リリース・ノート』には、Open Client™ および SDK に関する重要な最新情報が記載されています。
- 『jConnect™ for JDBC™ リリース・ノート』には、jConnect に関する重要な最新情報が記載されています。
- 使用しているプラットフォームの『Open Client/Server 設定ガイド』では、システムを設定して Open Client/Server 製品を実行する方法について説明しています。
- 『Open Client Client-Library/C プログラマーズ・ガイド』では、Client-Library アプリケーションの設計方法および実装方法について説明しています。
- 『Open Client Client-Library/C リファレンス・マニュアル』では、Open Client Client-Library™ のリファレンス情報について説明しています。
- 『Open Server Server-Library/C リファレンス・マニュアル』では、Open Server Server-Library のリファレンス情報について説明しています。
- 『Open Client および Open Server Common Libraries リファレンス・マニュアル』では、CS-Library のリファレンス情報について説明しています。CS-Library は、Client-Library と Server-Library の両方のアプリケーションで役に立つユーティリティ・ルーチンの集まりです。
- 『Open Client/Server プログラマーズ・ガイド補足』では、Open Client/Server を使用するプログラマのために、プラットフォーム固有の情報について説明しています。このマニュアルには、次の情報が含まれています。
  - アプリケーションのコンパイルおよびリンク
  - Open Client/Server に含まれているサンプル・プログラム
  - プラットフォーム固有の動作をするルーチン
- 『Sybase® SDK DB-Library Kerberos 認証オプションのインストールおよびリリース・ノート』では、DB-Library で使用する MIT Kerberos セキュリティ メカニズムをインストールして有効にする方法について説明しています。DB-Library でサポートされる Kerberos セキュリティ・メカニズムの機能は、ネットワーク認証サービスと相互認証サービスのみです。

- 『Open Client/Server 開発者用国際化ガイド』では、国際化されたアプリケーションとローカライズされたアプリケーションを作成する方法について説明しています。
- 『Open Client Embedded SQL™/C プログラマーズ・ガイド』では、C アプリケーションで Embedded SQL および Embedded SQL プリコンパイラを使用する方法について説明しています。
- 『Open Client Embedded SQL™/COBOL プログラマーズ・ガイド』では、COBOL アプリケーションで Embedded SQL および Embedded SQL プリコンパイラを使用する方法について説明しています。
- 『jConnect for JDBC プログラマーズ・リファレンス』では、jConnect for JDBC 製品について説明し、リレーショナル・データベース管理システムに保管されているデータにアクセスする方法について説明しています。
- 『Adaptive Server® Enterprise ADO.NET Data Provider ユーザーズ・ガイド』では、C#、Visual Basic .NET、マネージ拡張を備えた C++、J# など、.NET でサポートされる任意の言語を使用して Adaptive Server 内のデータにアクセスする方法について説明しています。
- Sybase® 製 Adaptive Server Enterprise ODBC ドライバの『ユーザーズ・ガイド』(Microsoft Windows および UNIX 版) では、Microsoft Windows および UNIX プラットフォームの Adaptive Server から、Open Database Connectivity (ODBC) ドライバを使用してデータにアクセスする方法について説明します。
- Sybase 製 Adaptive Server Enterprise OLE DB プロバイダの『ユーザーズ・ガイド』(Microsoft Windows 版) では、Microsoft Windows プラットフォームの Adaptive Server から、Adaptive Server OLE DB プロバイダを使用してデータにアクセスする方法について説明します。
- 『Perl 用 Adaptive Server Enterprise データベース・ドライバ・プログラマーズ・ガイド』では、Perl 開発者が Perl スクリプトを使用して Adaptive Server のデータベースに接続し、情報をクエリまたは変更する方法について説明しています。
- 『PHP 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』では、PHP 開発者が Adaptive Server データベースに対してクエリを実行する方法について説明しています。
- 『Python 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』では、Adaptive Server データベースに対してクエリを実行するときに使用できる Sybase 固有の Python インタフェースについて説明しています。

その他の情報

Sybase Product Documentation Web サイトを使用して製品について詳しく知ることができます。

- Sybase Product Documentation Web サイトには、標準の Web ブラウザを使用してアクセスできます。また、製品ドキュメントのほか、EBFs/Maintenance、Technical Documents、Case Management、Solved Cases、Newsgroups、Sybase Developer Network へのリンクもあります。

  Sybase Product Documentation Web サイトは、Product Documentation http://www.sybase.com/support/manuals/ にあります。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品認定の最新情報にアクセスする**

1 Web ブラウザで Technical Documents http://www.sybase.com/support/techdocs/ を指定します。

2 [Partner Certification Report] をクリックします。

3 [Partner Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。

4 [Partner Certification Report] のタイトルをクリックして、レポートを表示します。

❖ **コンポーネント認定の最新情報にアクセスする**

1 Web ブラウザで Availability and Certification Reports http://certification.sybase.com/ を指定します。

2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。

3 [Search] をクリックして、入手状況と認定レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1 Web ブラウザで Technical Documents http://www.sybase.com/support/techdocs/ を指定します。

2 [MySybase] をクリックし、MySybase プロファイルを作成します。

Sybase EBF とソフトウェア・メンテナンス

❖ EBF とソフトウェア・メンテナンスの最新情報にアクセスする

1 Web ブラウザで the Sybase Support Page http://www.sybase.com/support を指定します。

2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。

3 製品を選択します。

4 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。

5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

表記規則

表 1：構文の表記規則

凡例	定義
コマンド	コマンド名、コマンドのオプション名、ユーティリティ名、ユーティリティのフラグ、キーワードは sans serif で示す。
変数	変数 (ユーザが入力する値を表す語) は*斜体*で表記する。
{ }	中カッコは、その中から必ず 1 つ以上のオプションを選択しなければならないことを意味する。コマンドには中カッコは入力しない。
[ ]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには中カッコは入力しない。
( )	このカッコはコマンドの一部として入力する。
|	中カッコまたは角カッコの中の縦線で区切られたオプションのうち 1 つだけを選択できることを意味する。
,	中カッコまたは角カッコの中のカンマで区切られたオプションをいくつでも選択できることを意味する。複数のオプションを選択する場合には、オプションをカンマで区切る。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Open Client および Open Server のマニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

注意　アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。

第 1 章

DB-Library の概要

この章では、DB-Library の概要について説明します。

クライアント／サーバ・アーキテクチャ

クライアント／サーバ・アーキテクチャでは、「クライアント」と「サーバ」がコンピューティング作業を分担します。

クライアントはサーバに対して要求を行い、サーバから返された要求の結果を処理します。たとえば、データベース・サーバのデータを要求するクライアント・アプリケーションもあれば、部屋の温度を下げるよう環境制御サーバに要求するアプリケーションもあります。

サーバは、データやその他の情報をクライアントへ返したり、何らかのアクションをとったりして、要求に応答します。たとえば、データベース・サーバは、表形式のデータとそのデータについての情報をクライアントに返し、電子メール・サーバは、受け取ったメールを最終的な宛先に転送します。

クライアント／サーバ・アーキテクチャは、次の点が従来のプログラム・アーキテクチャよりも優れています。

- 共通のサービスが 1 か所で、つまり 1 つのサーバで処理されるため、アプリケーションのサイズと複雑さが大幅に軽減されます。これによって、クライアント・アプリケーションが単純化され、コードの重複が減り、アプリケーションの保守が容易になります。
- クライアント／サーバ・アーキテクチャは、さまざまなアプリケーションの間の通信を容易にします。類似性のない通信プロトコルを使用するクライアント・アプリケーションは直接通信することはできませんが、両方のプロトコルを「理解できる」サーバを介せば通信できます。
- クライアント／サーバ・アーキテクチャにより、アプリケーションを独立したコンポーネントの集合として開発することが可能になります。これにより、アプリケーションの他の部分に影響を及ぼさずに、個々のコンポーネントを変更または置換できます。

クライアントのタイプ

クライアントとは、サーバへ要求を行うアプリケーションのことです。クライアントには次のものが含まれています。

- isql や bcp といった Adaptive Server Enterprise が提供する独立したユーティリティ
- Open Client ライブラリを使用して作成されたアプリケーション
- Open Client Embedded SQL™ を使用して作成されたアプリケーション

サーバのタイプ

Sybase の製品群には、次に示すように、サーバとサーバを構築するためのツールが含まれています。

- Adaptive Server Enterprise はデータベース・サーバです。Adaptive Server Enterprise は、1 つまたは複数のデータベースに格納された情報を管理します。
- Open Server は、カスタム・サーバ (別名「Open Server アプリケーション」) を作成するために必要なツールとインタフェースの集まりです。

Open Server アプリケーションは、あらゆる種類のサーバとして機能します。たとえば、Open Server アプリケーションで特殊な計算を行ったり、リアルタイム・データにアクセスしたり、電子メールなどのサービスにインタフェースしたりすることができます。Open Server アプリケーションは、Open Server Server-Library で提供されるビルディング・ブロックを使用して個別に開発します。

Adaptive Server Enterprise と Open Server アプリケーションには、次のような類似点があります。

- Adaptive Server Enterprise と Open Server はどちらもサーバであり、クライアントの要求に応答します。
- クライアントは、Adaptive Server Enterprise と Open Server アプリケーションのどちらとも、Open Client 製品を介して通信します。

しかし、次のような相違点もあります。

- Open Server アプリケーションは、Server-Library のビルディング・ブロックを使用してカスタム・コードを開発するという方法で作成する必要があります。Adaptive Server Enterprise は完成されており、カスタム・コードは必要ありません。
- Open Server アプリケーションは、あらゆる種類のサーバとして使用できるため、さまざまなプログラム言語を使用できるように作成できる。Adaptive Server Enterprise はデータベース・サーバであり、Transact-SQL のみを認識します。
- Open Server アプリケーションは、Sybase のアプリケーションとサーバだけでなく、TDS プロトコルに準拠していない「外部の」アプリケーションやサーバとも通信できます。Adaptive Server Enterprise は、直接通信できるのは Sybase のアプリケーションとサーバだけですが、Open Server ゲートウェイ・アプリケーションを中継すれば、外部のアプリケーションやサーバと通信できます。

Open Client および Open Server 製品

Sybase では、カスタマによるクライアントおよびサーバ・アプリケーション・プログラムの作成を可能にする次の 2 つの製品群を提供しています。Open Client と Open Server。

Open Client

Open Client は、カスタマ・アプリケーション、サードパーティ製品、および他の Sybase 製品が Adaptive Server Enterprise および Open Server と通信するためのインタフェースです。

Open Client には 2 つのコンポーネントがあると考えることができます。プログラミング・インタフェースとネットワーク・サービスです。

Open Client のプログラミング・インタフェース・コンポーネントは、クライアント・アプリケーションを作成するときに使用するライブラリで構成されます。このライブラリとは、Client-Library、DB-Library、および CS-Library です (CS-Library は Open Client と Open Server の両方にあり、クライアント・アプリケーションとサーバ・アプリケーションの両方に使用できるユーティリティ・ルーチンが含まれています)。

Open Client ネットワーク・サービスには Net-Library があり、TCP/IP などの、特定のネットワーク・プロトコルをサポートします。

Open Server

Open Server では、カスタム・サーバ・アプリケーションの作成に必要なツールとインタフェースを提供します。Open Client と同様に、Open Server にもプログラミング・インタフェース・コンポーネントとネットワーク・サービス・コンポーネントがあります。

Open Server のプログラミング・インタフェースには、Server-Library と CS-Library があります (CS-Library は Open Client と Open Server の両方にあり、クライアント・アプリケーションとサーバ・アプリケーションの両方に使用できるユーティリティ・ルーチンが含まれています)。

Open Server のネットワーク・サービスは、ほとんどが透過的です。

Open Client ライブラリ

Open Client を構成するライブラリは、次のとおりです。

- DB-Library：クライアント・アプリケーションを作成するときに使用するルーチンの集合です。DB-Library には、バルク・コピー・ライブラリと 2 フェーズ・コミット・ライブラリが含まれています。DB-Library は、古い Sybase アプリケーションのソース・コード互換性を維持するためのものです。

- Client-Library：クライアント・アプリケーションを作成するときに使用するルーチンの集合です。Client-Library は、カーソルとその他の新しい機能に対応するように設計されているライブラリです。
- CS-Library：クライアント・アプリケーションとサーバ・アプリケーションの両方に役立つユーティリティ・ルーチンの集まりです。Client-Library ルーチンは CS-Library 内で割り付けられる構造体を使用するため、すべての Client-Library アプリケーションには、CS-Library への呼び出しが少なくとも 1 つ含まれます。

DB-Library/C の内容

注意　DB-Library は、古い Sybase アプリケーションのソース・コード互換性を維持するためのものです。新しいアプリケーションの実装には、Client-Library または Embedded SQL を使用することをおすすめします。

DB-Library/C の C ルーチンやマクロをアプリケーションで利用すれば、Adaptive Server Enterprise アプリケーションや Open Server アプリケーションと対話できるようになります。

DB-Library/C には、Adaptive Server Enterprise アプリケーションや Open Server アプリケーションにコマンドを送るルーチン、およびこれらのコマンドの結果を処理するルーチンが含まれています。その他に、エラー状態の処理やデータの変換を行うルーチン、およびアプリケーションとサーバとの対話に関するさまざまな情報を返すルーチンがあります。

DB-Library/C には、さらに、ルーチンが使用する値や構造体を定義する多数のヘッダ・ファイルも含まれています。C 言語の他に、COBOL、FORTRAN、Ada、Pascal などの言語用の DB-Library が開発されています。

Embedded SQL とライブラリ・アプローチの比較

Open Client ライブラリ・アプリケーションと Embedded SQL アプリケーションのどちらも、Adaptive Server Enterprise に SQL コマンドを送信できます。

一般的に、Embedded SQL は Transact-SQL のスーパーセットです。Embedded SQL のアプリケーションでは、アプリケーションのホスト言語文の中に Embedded SQL コマンドが混在しています。ホスト言語のプリコンパイラは、Embedded SQL コマンドを処理して Client-Library ルーチンへの呼び出しに変換し、既存のホスト言語文はそのまま残します。リリース 10.0 以降のプリコンパイラはいずれも、Client-Library と CS-Library 呼び出しだけで構成されるランタイム・ライブラリを使用します。

つまり、ある意味では、プリコンパイラによって Embedded SQL アプリケーションが Client-Library アプリケーションに変換されると言えます。

Open Client ライブラリ・アプリケーションはライブラリ・ルーチンを使用して SQL コマンドを送り、プリコンパイラを必要としません。

一般に、Embedded SQL アプリケーションの方が作成とデバッグは簡単ですが、ライブラリ・アプリケーションの方が Open Client ルーチンの柔軟性と機能をより十分に活用できます。

サーバとの通信のためのデータ構造体

DB-Library/C アプリケーションは、1 つまたは複数の DBPROCESS 構造体を介してサーバと通信します。DBPROCESS を介して、コマンドがサーバに送信され、クエリ結果がアプリケーションに返されます。一般にアプリケーションが最初に呼び出すルーチンの 1 つに、dbopen があります。このルーチンはアプリケーションをサーバにログインさせ、DBPROCESS を割り付けて初期化します。このとき、この DBPROCESS は、アプリケーションとサーバを接続する役割を果たします。ほとんどの DB-Library/C ルーチンが、第 1 パラメータとして DBPROCESS を必要とします。

1 つのアプリケーションで、複数の DBPROCESS をオープンして 1 つ以上のサーバに接続できます。たとえば、クエリの結果を処理する途中でデータベースの更新を行うアプリケーションは、タスクごとに別の DBPROCESS が必要です。また、あるサーバからデータを検索し、別のサーバのデータベースを更新するには、サーバごとに 1 つずつ、つまり 2 つの DBPROCESS が必要です。アプリケーションの DBPROCESS は、それぞれ他の DBPROCESS とは独立して機能します。

DBPROCESS 構造体は、サーバに送信するための言語コマンドが格納されているコマンド・バッファを指し示します。また、サーバから返された結果ローも指しています。結果ローは、単一ローか、バッファリングが指定されている場合はローのバッファです。さらに、サーバから返されたエラー・メッセージと情報メッセージが格納されているメッセージ・バッファも指しています。

DBPROCESS は、サーバの対話に関するさまざまな角度からの豊富な情報も保持しています。DB-Library/C ルーチンの多くは、DBPROCESS から情報を抽出しています。アプリケーションで DBPROCESS 構造体のコンポーネントにアクセスしたりコンポーネントを操作したりするときは、直接ではなく DB-Library/C ルーチンを介して行ってください。

もう 1 つの重要な構造体に LOGINREC があります。この構造体には、ユーザ名やパスワードといった一般的なログイン情報が格納され、dbopen ルーチンはこの情報を使用してサーバにログインします。DB-Library/C ルーチンは、LOGINREC 内の情報を指定できます。

DB-Library/C のプログラミング

アプリケーション・プログラマは、DB-Library 構造体の設定、サーバとの接続、コマンドの送信、結果の処理、終了処理を行う DB-Library ルーチンへの呼び出しを使用して、DB-Library プログラムを作成します。DB-Library プログラムは、他の C 言語プログラムと同様にコンパイルおよび実行されます。

DB-Library/C を使用するプログラムでは、一般に、次の基本手順を実行します。

1　サーバにログインします。

2　言語コマンドをバッファに入れ、サーバへ送信します。

3　サーバから返された結果がある場合は、その結果を処理します。一度に 1 コマンドずつ、結果ローを 1 つずつ処理します。結果はプログラム変数に入れることができ、アプリケーションはこの変数を使用して結果を操作します。

4　DB-Library/C のエラー・メッセージとサーバ・メッセージを処理します。

5　サーバとの接続をクローズします。

次の例では、DB-Library/C アプリケーションの基本構造を示します。このプログラムは、Adaptive Server Enterprise との接続をオープンし、Transact-SQL の select コマンドをサーバに送り、select の結果として返されるローの集合を処理します。このプログラムにはエラー処理とメッセージ処理のルーチンがありません。それらのルーチンの使用方法については、DB-Library に含まれているサンプル・プログラムを参照してください。

```
#include <sybfront.h>
#include <sybdb.h>
#include <syberror.h>

/* Forward declarations of the error handler and message
** handler.
*/
interr_handler();
intmsg_handler();

main()
{
     DBPROCESS    *dbproc;     /* The connection with */
                               /* Adaptive Server Enterprise */
     LOGINREC     *login;      /* The login information */
     DBCHAR       name[40];
     DBCHAR       city[20];
     RETCODE      return_code;

     /* Initialize DB-Library */
     if (dbinit() == FAIL)
     exit(ERREXIT);

     /*
     ** Install user-supplied error-handling and message-
     ** handling routines. The code for these is omitted
     ** from this example for conciseness.
     */
     dberrhandle(err_handler);
     dbmsghandle(msg_handler);

     /* Get a LOGINREC */
     login = dblogin();
     DBSETLPWD(login, "server_password");
     DBSETLAPP(login, "example");
```

```
                /* Get a DBPROCESS structure for communication */
                /* with Adaptive Server Enterprise.*/
                dbproc = dbopen(login, NULL);

                /*
                ** Retrieve some columns from the "authors" table
                ** in the "pubs2" database.
                */

                /* First, put the command into the command buffer.*/
                dbcmd(dbproc, "select au_lname, city from
                      pubs2..authors");
             dbcmd(dbproc, "
                  where state = 'CA' ");

                /*
                ** Send the command to Adaptive Server Enterprise and start
execution
                */
                dbsqlexec(dbproc);

                /* Process the command */
                while ((return_code = dbresults(dbproc)) !=
                     NO_MORE_RESULTS)
                {
                     if (return_code == SUCCEED)
                {
                     /* Bind results to program variables.*/
                     dbbind(dbproc, 1, STRINGBIND, (DBINT)0, name);
                     dbbind(dbproc, 2, STRINGBIND, (DBINT)0, city);

                     /* Retrieve and print the result rows.*/
                     while (dbnextrow(dbproc) != NO_MORE_ROWS)
                     {
                     printf("%s:%s¥n", name, city);
                     }
                 }
             }

             /* Close the connection to Adaptive Server Enterprise */
             dbexit();
         }
```

この例の機能は、ほとんどの DB-Library/C アプリケーションに共通するものです。

- ヘッダ・ファイル ― DB-Library/C ルーチンへの呼び出しが含まれるソース・ファイルには、2 つのヘッダ・ファイル *sybfront.h* と *sybdb.h* が必要です。*sybfront.h* はファイルの先頭になければなりません。このファイルは、「第 2 章 ルーチン」のリファレンス・ページで説明する関数の戻り値や、終了値である STDEXIT と ERREXIT などの記号定数を定義します。これらの終了値は C 標準ライブラリ関数 exit の引数として使用できます。これらの値はプログラムを実行しているオペレーティング・システム用に定義されているため、システムに依存しない方法でプログラムを終了できます。*sybfront.h* にはプログラム変数宣言で使用するデータ型に対する型定義もあります。これらのデータ型については後で説明します。

  sybdb.h に含まれるその他の定義のほとんどは、DB-Library/C ルーチンが使用するためのもので、プログラムが直接アクセスするものではありません。*sybdb.h* で最も重要なのは、DBPROCESS 構造体の定義です。すでに説明したように、DBPROCESS 構造体の操作は、必ず DB-Library/C ルーチンを介して行います。この構造体のコンポーネントに直接アクセスしないでください。DB-Library/C の今後のリリースとの互換性を保つため、*sybdb.h* の内容は、「第 2 章 ルーチン」のリファレンス・ページで説明しているとおりに使用してください。

  例の中の 3 番目のヘッダ・ファイルである *syberror.h* には、エラー重大度の値があります。プログラムでこれらの値を参照する場合は、このヘッダ・ファイルをインクルードしてください。

- dbinit ― このルーチンは DB-Library/C を初期化します。これは、プログラム内で最初に呼び出す DB-Library/C ルーチンでなければなりません。現時点では、すべての DB-Library/C 環境に dbinit 呼び出しが必要なわけではありません。しかし、将来の互換性と移植性を保証するために、DB-Library/C プログラムの開始時に、必ずこのルーチンを呼び出してください。

- dberrhandle と dbmsghandle ― dberrhandle は、ユーザが作成したエラー処理ルーチンをインストールします。アプリケーションで DB-Library/C のエラーが発生すると、このエラー処理ルーチンが自動的に呼び出されます。同様に、dbmsghandle は、メッセージ処理ルーチンをインストールします。これはサーバから返される情報メッセージおよびエラー・メッセージへの応答として呼び出されます。エラーやメッセージを処理するルーチンは、ユーザが作成するものです。この例ではハンドラの例は挙げていませんが、DB-Library のサンプル・プログラムには含まれています。詳細については、使用しているプラットフォームの『Open Client/Server プログラマーズ・ガイド補足』を参照してください。
- dblogin ― このルーチンは、LOGINREC 構造体を割り付けます。DB-Library/C は、サーバにログインするときにこの構造体を使用します。その後で実行する 2 つのマクロによって、LOGINREC のコンポーネントを設定します。DBSETLUSER と DBSETLPWD は、ログインするときに DB-Library/C が使用するユーザ名とパスワードを設定します。DBSETLAPP は、アプリケーションの名前を設定します。このアプリケーション名は Adaptive Server Enterprise の sysprocesses テーブルに登録されます。ルーチンは、LOGINREC の他の部分の設定に使用できます。ただし、大部分の環境において、これらのルーチンはオプションであるため、LOGINREC には、設定されている各値のデフォルト値が含まれます。
- dbopen ― dbopen ルーチンは、アプリケーションとサーバの間の接続をオープンします。これは、dblogin によって作成される LOGINREC を使用してサーバにログインし、DBPROCESS 構造体を返します。DBPROCESS 構造体は、アプリケーションとサーバとの間の情報路となります。このルーチンが呼び出されると、アプリケーションは Adaptive Server Enterprise と接続され、Transact-SQL コマンドを Adaptive Server Enterprise へ送り、結果を処理できるようになります。
- dbcmd ― このルーチンは、コマンド・バッファに Transact-SQL コマンドを格納します。バッファに格納されたコマンドは、Adaptive Server Enterprise に送ることができます。dbcmd を連続して呼び出すと、すでにバッファにあるテキストの最後に、指定のテキストが付加されます。プログラム例の 2 番目の dbcmd の呼び出しテキストの先頭のブランクのように、語と語の間に必要なブランクを挿入するのはプログラマが行います。バッファには複数のコマンドを入れることができます。この例では、1 つのコマンドを送って処理する方法しか示していませんが、DB-Library/C は、アプリケーションが複数のコマンドをサーバに送り、各コマンドの結果セットを個別に処理できるように設計されています。

- dbsqlexec ― このルーチンは、コマンド・バッファを実行します。つまり、バッファの内容を Adaptive Server Enterprise に送り、Adaptive Server Enterprise はこれを解析して実行します。
- dbresults ― このルーチンは、現在処理可能な状態にある Transact-SQL コマンドの結果を受け取ります。この例では、バッファには、ローを返すコマンドが 1 つだけあります。したがって、プログラムは dbresults を 1 回呼び出す必要があります。ただし、プログラミングの練習のためにループの中で dbresults を呼び出しています。実際には不要な場合でも、この例のように、dbresults を常にループで呼び出すことをおすすめします。
- dbbind ― dbbind は、結果カラムをプログラム変数にバインドします。この例では、dbbind への最初の呼び出しが、最初の結果カラムをプログラム変数 *city* にバインドします。つまり、プログラムが dbnextrow を呼び出して結果ローを読み込むと、最初の結果カラム (au_lname) の内容がプログラム変数 *name* に格納されます。2 番目の dbbind 呼び出しは、2 番目の結果カラムを変数 *city* にバインドします。

  どちらのバインドも、バインド型は STRINGBIND です。これは、文字データで使用できるバインド型の 1 つです。バインド型は、指定したプログラム変数のデータ型に対応していなければなりません。この例では、変数のデータ型は DBCHAR です。これは、STRINGBIND の結果を受け取ることができる、DB-Library/C で定義されたデータ型の 1 つです。バインド型パラメータにより、dbbind は多様な型変換をサポートするので、結果カラムのデータ型と受け取り側の変数のデータ型は同じである必要はありません。
- dbnextrow ― このルーチンは 1 つのローを読み込み、その結果を前の dbbind 呼び出しで指定されたプログラム変数に格納します。dbnextrow を連続して呼び出すと、次の結果ローが読み込まれます。最後のローが読み込まれると、NO_MORE_ROWS が返されます。dbnextrow を呼び出すたびにプログラム変数の値は上書きされるため、結果の処理は dbnextrow のループ内で行う必要があります。このプログラム例は、単に各ローの内容を出力しています。
- dbexit ― このルーチンは、サーバ接続をクローズし、DBPROCESS の割り付けを解除します。また、dbinit で初期化された構造体もすべてクリーンアップします。これはプログラム内で最後に呼び出す DB-Library/C ルーチンです。

DB-Library/C には多くのルーチンがありますが、ほとんどのプログラムは、この例に示した少数のルーチンで作成できます。

DB-Library/C のデータ型

DB-Library/C は、Adaptive Server Enterprise のデータに対応するデータ型を定義しています。これらのデータ型は「SYB」で始まります (SYBINIT4、SYBCHAR、SYBMONEY など)。多くのルーチンには、パラメータとしてこれらのデータ型が必要です。DB-Library/C と Server-Library/C には、プログラム変数の宣言で使用する型定義も含まれています。このような型は、DB-Library/C では「DB」で始まり (DBINT、DBCHAR、DBMONEY など)、Server-Library/C では『SRV_』で始まります (SRV_INT4、SRV_CHAR、SRV_MONEY など)。これらを使用すると、プログラム変数の互換性が保証されます。

Adaptive Server Enterprise のデータ型と対応する DB-Library/C プログラム変数データ型のリストについては、「データ型」(443 ページ) を参照してください。また、Server-Library のデータ型のリストについては『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

dbconvert_ps ルーチンを利用すると、あるサーバ・データ型から別のサーバ・データ型にデータを変換できます。これは、ほとんどのデータ型間の変換をサポートします。Adaptive Server Enterprise と Open Server のデータ型は DB-Library/C のデータ型に直接対応するため、アプリケーションのさまざまな場面で dbconvert_ps を使用できます。サーバの結果カラムをプログラム変数にバインドするルーチンである dbbind と dbaltbind も、型変換を行います。

DB-Library/C のルーチン

DB-Library/C のルーチンとマクロは、さまざまなタスクを処理します。この項の説明は、以下のカテゴリに分かれています。

- 初期化
- コマンドの処理
- 結果の処理
- メッセージとエラーの処理
- 情報の検索
- ブラウズ・モード
- text と image の処理

- データ型変換
- プロセス制御フロー
- リモート・プロシージャ・コールの処理
- レジスタード・プロシージャ・コールの処理
- ゲートウェイ・パススルー・ルーチン
- datetime と money
- 終了処理
- 機密保護のサポート
- その他のルーチン

ルーチンとマクロの説明は、「第 2 章 ルーチン」のそれぞれのリファレンス・ページにあります。ルーチンとマクロは、すべてプレフィクス「db」が付きます。ルーチン名は小文字、マクロ名は大文字です。

さらに、DB-Library/C には次の 2 つの特殊ライブラリが含まれています。

- バルク・コピー (「第 3 章 バルク・コピー・ルーチン」を参照)
- 2 フェーズ・コミット・サービス (「第 4 章 2 フェーズ・コミット・サービス」を参照)

バルク・コピー・ルーチンには、プレフィクス「bcp」が付きます。2 フェーズ・コミット・ルーチンには、標準のプレフィクスはありません。

初期化

これらのルーチンは、アプリケーション・プログラムとサーバの間での接続の設定と定義を行います。LOGINREC 構造体の割り付けと定義、サーバへの接続のオープン、DBPROCESS 構造体の割り付けなどのタスクを扱います。すべての DB-Library/C プログラムに必要なルーチンはごく少数です。特に、dbinit、dblogin、および dbopen がアプリケーションに必要です。プログラムで初期化ルーチンを呼び出す順序は、ここで示す順にほぼ一致します。

DB-Library/C の初期化

DB-Library の内部環境を設定する最上位レベルのルーチンです。

- dbinit － DB-Library/C で使用される基本構造体を初期化します。
- dbsetversion － DB-Library のバージョン・レベルを指定します。
- dbsetmaxprocs － 同時にオープンできる DBPROCESS 構造体の最大数を設定します。
- dbgetmaxprocs － 現時点で同時にオープンできる DBPROCESS 構造体の最大数を示します。

LOGINREC の設定

これらのルーチンは、LOGINREC 内にデータを格納します。LOGINREC に格納されたユーザ情報は、プログラムが dbopen を呼び出して接続をオープンするときに DB-Library によってサーバに送信されます。

- dblogin － 以降の dbopen で使用される LOGINREC 構造体を割り付けます。
- DBSETLUSER － LOGINREC 構造体内のサーバ・ユーザ名を設定します。
- DBSETLPWD － LOGINREC 構造体内のサーバ・パスワードを設定します。
- DBSETLAPP － LOGINREC 構造体内のアプリケーション名を設定します。
- DBSETLHOST － LOGINREC 構造体内のホスト名を設定します。
- DBSETLCHARSET － LOGINREC 構造体内の文字セットを設定します。
- DBSETLPACKET － アプリケーションの Tabular Data Stream™ (TDS) パケット・サイズを設定します。
- dbgetpacket － 現在の TDS パケット・サイズを返します。
- dbrpwset － リモート・パスワードを LOGINREC 構造体に追加します。サーバは、別のサーバでリモート・プロシージャ・コールを実行するときに、このパスワードを使用します。
- dbrpwclr － LOGINREC 構造体からリモート・パスワードをすべてクリアします。
- dbloginfree － LOGINREC 構造体を解放します。

サーバ接続の確立

アプリケーションは、以下のルーチンを呼び出してリモート・サーバへの接続を設定し、オープンします。

- dbsetifile － dbopen がサーバとの接続に使用する interfaces ファイルを指定します。
- dbsetlogintime － DB-Library/C が dbopen による DBPROCESS 接続の要求に対するサーバの応答を待つ秒数を設定します。
- dbopen － ネットワークとの通信を設定し、LOGINREC を使用してサーバにログインし、LOGINREC で指定されているすべてのオプションを初期化して、DBPROCESS を割り付けます。アプリケーションは 1 つのサーバに対して複数の接続をオープンすることができ、接続ごとに別の DBPROCESS が存在します。また、1 つのアプリケーションで、複数のサーバに対する複数の接続をオープンすることもできます。
- dbuse － 現在のデータベースを設定します。このルーチンは Transact-SQL の use コマンドに相当し、接続がオープンされていればいつでも、同じアプリケーションの中で繰り返し呼び出すことができます。

コマンドの処理

アプリケーションは、言語コマンドを通してサーバと通信できます。Adaptive Server Enterprise での言語は、Transact-SQL です。Open Server では、Open Server が理解できるようにプログラムされたものであれば、任意の言語を使用できます。アプリケーションは、DBPROCESS が指しているコマンド・バッファにコマンドを入れます。コマンド・バッファには複数のコマンドを格納することができ、バッファの中のコマンドのセットはコマンド・バッチとして認識されます。アプリケーションは、このコマンド・バッチをサーバに送信し、サーバは、バッファに入力されている順序でコマンドを実行します。

コマンド・バッチの構築

次のルーチンは、バッファへのコマンドの追加やバッファのクリアを行います。

- dbcmd ― テキストをコマンド・バッファに追加します。複数のコマンドや、コマンドの一部分を追加するために、繰り返し呼び出すことができます。連続した呼び出しで追加されたテキストは、前のテキストに連結されます。
- dbfcmd ― sprintf タイプのフォーマットを使用して、テキストをコマンド・バッファに追加します。このルーチンは、テキストが代入される引数を利用できる点を除いて、dbcmd と同じです。
- dbfreebuf ― コマンド・バッファをクリアします。コマンドのバッチが入力される前に、自動的にコマンド・バッファがクリアされます。それ以外の時点でコマンド・バッファをクリアする場合、または DBNOAUTOFREE オプションが設定されているときにコマンド・バッファをクリアする場合は、dbfreebuf を使用してください。

コマンド・バッチのアクセス

次のルーチンは、コマンド・バッファの一部の検査やコピーに使用します。

- dbgetchar ― コマンド・バッファ内の特定の文字へのポインタを返します。
- dbstrlen ― コマンド・バッファの長さを返します。
- dbstrcpy ― コマンド・バッファの一部をプログラム変数にコピーします。このルーチンにより、サーバに何が送られたかを正確に把握することができるため、特にデバッグ時に便利です。

コマンド・バッチの実行

言語コマンドがバッファに入力されたら、これらのコマンドを実行するためにサーバに送信します。

- dbsqlsend ― 実行のためにコマンド・バッファの内容をサーバに送ります。dbsqlexec とは異なり、このルーチンはサーバからの応答を待ちません。dbsqlsend が SUCCEED を返した場合は、コマンド・バッチの正当性を検証するために、dbsqlok を呼び出す必要があります。
- dbpoll ― dbsqlsend (または dbrpcsend) と dbsqlok の間で呼び出され、サーバの応答が DBPROCESS に到着したかどうかを調べます。

- dbsqlok ― サーバからの結果を待って、サーバが応答した命令の正当性を検証します。このルーチンは、dbsqlsend、dbrpcsend、dbmoretext とともに使用します。dbsqlok の呼び出しが正常に終了した後、アプリケーションは結果を処理するために、dbresults を呼び出します。
- dbsqlexec ― 実行のためにコマンド・バッファの内容をサーバに送ります。dbsqlexec が SUCCEED を返したときは、結果を処理するために dbresults を呼び出す必要があります。dbsqlexec を呼び出すことは、dbsqlsend と dbsqlok を続けて呼び出すことと同じです。

コマンド・オプションの設定およびクリア

アプリケーションは、Adaptive Server Enterprise および DB-Library/C のさまざまなコマンド・オプションを設定できます。このオプションには、Adaptive Server Enterprise にコマンド・バッチの実行ではなく解析だけを行わせる DBPARSEONLY や、結果ローをバッファに格納するための DBBUFFER などがあります。使用可能なオプションとその意味については、「オプション」(436 ページ) を参照してください。

- dbsetopt ― オプションを設定します。
- dbclropt ― オプションをクリアします。
- dbisopt ― 特定のオプションが設定されているかどうかを判断します。

結果の処理

サーバでコマンド・バッチが実行され、dbsqlexec または dbsqlok から SUCCEED が返されると、アプリケーションは結果の処理を行います。結果には次の 2 種類があります。

- サーバからの、正常終了または失敗の表示
- 結果ロー

結果ローは、select コマンドを実行したときや、select コマンドが含まれるストアド・プロシージャを execute コマンドで実行したときに返されます。

結果ローには 2 種類あります。通常ローと計算ローです。通常ローは select コマンドの select リストにあるカラムから生成されます。計算ローは select コマンドの compute 句にあるカラムから生成されます。この 2 種類のローに含まれるデータはまったく異なるため、アプリケーションはこの 2 つを別々に処理する必要があります。

バッチ内の各 Transact-SQL コマンドに対する結果は、アプリケーションに個別に返されます。各コマンドの結果セット内では、結果ローは 1 つずつ処理されます。

コマンド・バッチに Transact-SQL コマンドが 1 つしか含まれておらず、そのコマンドが select コマンドなどのローを返すコマンドの場合は、アプリケーションで dbresults を呼び出してそのコマンドの結果を処理する必要があります。

コマンド・バッチに Transact-SQL コマンドが 1 つしか含まれておらず、そのコマンドが use database コマンドや insert コマンドなどのローを返さないコマンドの場合は、アプリケーションでコマンドの結果を処理するために dbresults を呼び出す必要はありません。しかし、このような状況で dbresults を呼び出しても問題はありません。どのコマンドの後でも dbresults が NO_MORE_RESULTS を返すまで繰り返し dbresults を呼び出すようにすれば、コードの管理がより簡単になります。

コマンド・バッチに複数の Transact-SQL コマンドが含まれる場合は、各コマンドがローを返すかどうかに関係なく、バッチ内のコマンドごとに dbresults を一度ずつ呼び出す必要があります。このため、DB-Library/C アプリケーションでは、1 つまたは複数のコマンドをサーバに送った後に、ループの中で dbresults を呼び出すことをおすすめします。

表 1-1 は、Transact-SQL コマンドと、コマンドが返す結果の処理に必要な DB-Library/C 関数を示します。

表 1-1：Transact-SQL コマンドを処理するために必要な DB-Library/C 関数

Transact-SQL コマンド	必要な DB-Library/C 関数
この表に記載されていないいすべての Transact-SQL コマンド	dbresults：ただし、dbcc などのコマンドの場合は、コマンドの通常の出力がエラーとメッセージから構成されると DB-Library/C がみなすことがある。このため、出力は、メイン・プログラムで dbnextrow などの DB-Library/C ルーチンを使用して処理されるのではなく、DB-Library/C アプリケーションのエラー・ハンドラやメッセージ・ハンドラで処理される。
execute	DB-Library/C アプリケーションは、ストアド・プロシージャが返す結果セットごとに dbresults を一度ずつ呼び出す必要がある。さらに、ストアド・プロシージャがローを返す場合は、dbnextrow または他の DB-Library/C の結果ロー・ルーチンを呼び出さなければならない。
select	dbresults：さらに、dbnextrow または他の DB-Library/C の結果ロー・ルーチンを呼び出さなければならない。

結果の設定

dbresults はバッチ内の次のコマンドの結果を設定します。dbresults は、dbsqlexec または dbsqlok から SUCCEED が返されてから、dbbind または dbnextrow を呼び出す前に呼び出します。

結果データの取得

結果データを取得する最も簡単な方法は、dbbind または dbaltbind を使用して結果カラムをプログラム変数にバインドすることです。このようにすれば、アプリケーションが dbnextrow を呼び出して結果ローを読み込むと (「結果ローの読み込み」(22 ページ) を参照)、DB-Library/C は、カラムがバインドされているプログラム変数にカラムのデータを自動的にコピーします。dbbind または dbaltbind は、dbresults を呼び出した後で、最初の dbnextrow の呼び出しの前に呼び出す必要があります。

dbdata と dbadata を使用して、結果カラムのデータに直接アクセスすることもできます。これらのルーチンは、データへのポインタを返します。dbdata および dbadata には、データのコピーではなく実際のデータにアクセスできるという利点があります。これらのルーチンは、dbdatlen および dbadlen とともに使用されることもあります。dbdatlen および dbadlen はデータの長さを返すルーチンです。詳細については、「情報の検索」(27 ページ) を参照してください。これらのルーチンを使用してデータに直接アクセスする場合は、あらかじめ結果カラムをプログラム変数にバインドしておく必要はありません。dbnextrow の後で、dbdata または dbadata を呼び出すだけでアクセスできます。

結果カラムを取得するには、次のルーチンを使用します。

- dbbind ― 通常ローの結果カラムをプログラム変数にバインドします。
- dbbind_ps ― 通常ローの結果カラムをプログラム変数にバインドします。numeric 変数および decimal 変数の精度と位取りをサポートします。
- dbaltbind ― 計算ローの結果カラムをプログラム変数にバインドします。
- dbaltbind_ps ― 計算ローの結果カラムをプログラム変数にバインドします。numeric 変数および decimal 変数の精度と位取りをサポートします。
- dbdata ― 通常ローの結果カラムのデータへのポインタを返します。
- dbadata ― 計算ローの結果カラムのデータへのポインタを返します。
- dbnullbind ― 通常ローの結果カラムにインジケータ変数を関連付けます。
- dbanullbind ― 計算ロー・カラムにインジケータ変数を関連付けます。
- dbsetnull ― null 値をバインドするときに使用する代入値を定義します。
- dbprtype ― サーバ型トークンを、判読可能な文字列に変換します。トークンは、dbcoltype や dbaltop などのさまざまなルーチンによって返されます。

結果ローの読み込み

dbresults から SUCCEED が返され、カラムからプログラム変数へのバインドが指定されていれば、アプリケーションで結果の処理を開始できます。初めに行う処理は、結果ローをアプリケーションで使用できるようにすることです。これは、dbnextrow ルーチンによって行います。dbnextrow を呼び出すたびに、サーバから返された次のローが読み込まれます。ローはネットワークから直接読み込まれます。

dbnextrow によってローが読み込まれると、アプリケーションで、そのロー内のデータを自由に処理できます。結果カラムがプログラム変数にバインドされている場合、ロー内のデータは自動的に変数にコピーされます。そうでない場合は、データにアクセスするには dbdata または dbadata を使用します。

必要であれば、dbnextrow で読み込んだローを自動的にロー・バッファに保存することもできます。このようにするには、dbsetopt ルーチンを使用して DBBUFFER オプションを設定します。ロー・バッファリングは、結果ローを順次処理以外の方法で処理する必要があるアプリケーションに便利です。ロー・バッファリングを行わない場合は、dbnextrow でローを読み込んだときにそのローを処理しなければなりません。これは、次に dbnextrow を呼び出したときにローが上書きされるためです。アプリケーションでロー・バッファリングを有効にすると、dbnextrow で読み込まれたローはロー・バッファに追加されます。このようにすれば、dbgetrow ルーチンを使用してバッファ内を移動したり、これまでに読み込んだローに戻ったりすることができます。ロー・バッファリングは、メモリへの影響やパフォーマンスの低下を伴うため、使用するときは十分に注意してください。ロー・バッファリングはネットワーク・バッファリングとは無関係であり、まったく別の問題である点に注意してください。

デフォルト・フォーマットで結果ローを出力するルーチンもあります。フォーマットは事前に決定されているため、これらのルーチンの効用は限られており、主にデバッグに適しています。

DB-Library/C は、コマンドの結果を 1 つずつ処理します。アプリケーションで 1 つのコマンドの結果をすべて読み込んだときは、コマンド・バッファ内の次のコマンドに対する結果を設定するために、もう一度 dbresults を呼び出す必要があります。すべての結果を処理できるように、dbresults をループの中で呼び出すことをおすすめします。

結果ローを処理するには、次のルーチンを使用します。

- dbnextrow ― 次のローを読み込みます。dbnextrow からの戻り値は、ローが通常ローか計算ローか、ロー・バッファが満杯かどうか、および最後の結果ローが読み込まれているかどうかをアプリケーションに通知します。
- DBCURROW ― 現在読み込まれているローの番号を返します。
- dbprhead ― 結果ローに対するデフォルトのカラム見出しを出力します。このルーチンは dbprrow とともに使用します。
- dbprrow ― すべての結果ローをデフォルト・フォーマットで出力します。このルーチンを使用するために、結果をバインドしたり、dbnextrow を呼び出したりする必要はありません。

結果のキャンセル

次のルーチンで結果をキャンセルします。

- dbcancel ― 現在のコマンド・バッチの結果をキャンセルします。このルーチンは、現在のバッチのすべてのコマンドをキャンセルします。
- dbcanquery ― 最後に実行されたクエリからの保留中のローをキャンセルします。

この 2 つのルーチンの違いを、言語バッチの結果を処理するアプリケーションの例を通して示します。

```
select * from pubs.titles
select * from pubs.authors
```

titles のローの処理中に dbcanquery を呼び出すと、titles のローが破棄されます。アプリケーションは、引き続き dbresults を呼び出して次の文からのローを処理しなければなりません。titles のローの処理中に dbcancel を呼び出すと、titles のローと、バッチ内に残っている未処理のコマンドの結果がすべて廃棄されます。アプリケーションは、dbcancel を呼び出した後に dbresults を呼び出し続ける必要はありません。

ストアド・プロシージャの結果の処理

ストアド・プロシージャの呼び出しは、「リモート・プロシージャ・コールの処理」(34 ページ) で説明するリモート・プロシージャ・コール、または Transact-SQL の execute コマンドによって行います。呼び出しによって生成される結果には、いくつかの種類があります。第 1 に、select 文が含まれるストアド・プロシージャは、通常の形態で結果ローを返します。後続の dbresults 呼び出しは、ストアド・プロシージャ内の次の select 文からのローのセットにアクセスします。このローは、dbnextrow を使用して通常の方法で処理できます。

第 2 に、ストアド・プロシージャには「リターン・パラメータ」を持つものがあります。リターン・パラメータは出力パラメータとも呼ばれ、ストアド・プロシージャの「参照による呼び出し」を可能にします。ストアド・プロシージャ内部で出力パラメータの値を変更した結果を、呼び出しプログラムで使用できます。呼び出しプログラムでは、dbresults と dbnextrow を呼び出してストアド・プロシージャの結果ローをすべて処理した後で、出力パラメータ値を取り出すことができます。以下に示すルーチンを使用して、リターン・パラメータ値の処理を行います。

第 3 に、ストアド・プロシージャはステータス番号を返すことができます。

ストアド・プロシージャの出力パラメータおよびリターン・ステータスにアクセスするには、次のルーチンを使用します。

- dbnumrets ― ストアド・プロシージャが生成するリターン・パラメータ値の数を返します。dbnumrets から返された値が 0 以下の場合は、使用可能なリターン・パラメータ値はありません。
- dbretdata ― リターン・パラメータ値へのポインタを返します。
- dbretlen ― リターン・パラメータ値の長さを返します。
- dbrettype ― リターン・パラメータ値のデータ型を返します。
- dbretname ― 特定の値に関連するリターン・パラメータの名前を返します。
- dbretstatus ― ストアド・プロシージャのステータス番号を返します。
- dbhasretstat ― 現在のコマンドまたはリモート・プロシージャ・コールが、ストアド・プロシージャのステータス番号を生成したかどうかを示します。dbhasretstat から 「FALSE」が返された場合は、ストアド・プロシージャのステータス番号は生成されていません。

結果タイムアウトの設定

デフォルトでは、DB-Library はサーバ・コマンドの結果が到着するのを無期限で待ちます。アプリケーションでは、次のルーチンを使用してタイムアウトの期間を指定できます。

- dbsettime － DB-Library/C がサーバの応答を待つ時間を秒単位で設定します。
- DBGETTIME － DB-Library/C がサーバの応答を待つ時間を秒単位で取得します。

メッセージとエラーの処理

DB-Library/C アプリケーションでは、次の 2 種類のメッセージとエラーを処理する必要があります。

- サーバのメッセージとエラーは、情報メッセージから致命的エラーまで、さまざまな重大度があります。DB-Library/C アプリケーションは、サーバのメッセージとエラーを「メッセージ」として認識します。すべての Adaptive Server Enterprise メッセージのリストを表示するには、次の Transact-SQL コマンドを実行します。

  ```
  select * from sysmessages
  ```

  Adaptive Server Enterprise メッセージのリストについては、『ASE システム管理ガイド』を参照してください。Open Server メッセージのリストについては、『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。
- DB-Library/C の警告とエラーは、DB-Library/C アプリケーションでは「エラー」として認識されます。DB-Library/C のエラーのリストについては、「エラー」(418 ページ) を参照してください。

また、ほとんどの DB-Library/C ルーチンが、正常に終了したか失敗したかの表示を返します。

DB-Library/C アプリケーションでサーバ・メッセージ、DB-Library/C エラー、および正常終了と失敗の表示を処理するには、次の方法があります。

- メインライン・コード内で DB-Library/C ルーチンのリターン・コードをテストし、状況に応じて失敗を処理します。
- メッセージ・ハンドラとエラー・ハンドラをインストールして、メッセージとエラーの処理を集中化します。メッセージまたはエラーが発生すると、DB-Library/C によってハンドラが自動的に呼び出されます。

すべての DB-Library/C アプリケーションで、メインライン・コードでのエラーのテストに加えて、メッセージとエラーの集中処理を使用することをおすすめします。メッセージとエラーの処理の集中化は、大規模で複雑なアプリケーションの場合に非常に効果的です。次に例を示します。

- メッセージとエラーの処理を集中化すれば、メインラインでのエラー処理論理の必要性は減少します。これは、メッセージまたはエラーが起きるたびに、DB-Library/C がアプリケーションのメッセージ・ハンドラおよびエラー・ハンドラを自動的に呼び出すためです。

  しかし、メッセージとエラーの集中処理を使用しているアプリケーションにおいても、その性質に応じて、メインラインでのエラー論理が必要となる点に注意してください。

- メッセージとエラーの処理を集中化すれば、予想外のエラーをスムーズに処理できます。メインラインでのエラー処理論理だけを使用しているアプリケーションでは、予想外のエラーをトラップできないことがあります。

DB-Library/C アプリケーションでのメッセージとエラーの処理を集中化するには、メッセージ・ハンドラとエラー・ハンドラのプログラムを作成し、dbmsghandle と dberrhandle を使用してインストールする必要があります。

メッセージとエラー処理のための DB-Library/C ルーチンを次に示します。

- dbmsghandle － サーバの情報メッセージおよびエラー・メッセージを処理するためのユーザ関数をインストールします。
- dberrhandle － DB-Library/C のエラー・メッセージを処理するためのユーザ関数をインストールします。
- DBDEAD － 特定の DBPROCESS が dead であるかどうかを判断します。DBPROCESS のステータスが dead のときは、現在の DB-Library/C ルーチンは失敗し、エラー・ハンドラが呼び出されます。

情報の検索

通常の結果カラム、計算結果カラム、ロー・バッファ、コマンド・ステータスなど、多岐にわたる情報を、DBPROCESS 構造体から検索できます。前述のように、通常の結果カラムは select コマンドの select リストの中のカラムに対応し、計算結果カラムは select コマンドのオプションの compute 句の中のカラムに対応します。

通常の結果カラムの情報

次のルーチンは、dbsqlexec から SUCCEED が返された後で呼び出すことができます。

- dbnumcols ― 現在の結果セット内のカラム数を調べます。
- dbcolname ― 通常の結果カラムの名前を返します。
- dbcollen ― 通常カラムのデータの最大長を返します。
- dbcoltype ― 通常の結果カラムのサーバ・データ型を返します。
- dbdatlen ― 通常カラムのデータの実際の長さを返します。このルーチンは、一般に dbdata とともに使用します。dbdatlen によって返される値は、dbnextrow によって読み込まれる通常ローごとに異なります。
- dbvarylen ― カラムのデータが可変長かどうかを示します。

計算結果カラムの情報

次のルーチンは、dbsqlexec から SUCCEED が返された後で呼び出すことができます。

- DBROWTYPE ― 現在の結果ローが通常ローか計算ローかを示します。
- dbnumcompute ― 現在の結果セット内の compute 句の数を返します。
- dbnumalts ― 計算ローのカラム数を返します。
- dbbylist ― 計算ローの bylist を返します。
- dbaltop ― 計算カラムの集合演算子のタイプを返します。
- dbalttype ― 計算カラムのデータ型を返します。
- dbaltlen ― 計算カラムのデータの最大長を返します。
- dbaltcolid ― 計算カラムのカラム ID を返します。

- dbadlen ― 計算カラムのデータの実際の長さを返します。このルーチンは、一般に dbadata とともに使用します。dbadlen によって返される値は、dbnextrow によって読み込まれる計算ローごとに異なります。

ロー・バッファ情報

次のマクロは、バッファ内の結果ローを操作するときに役立つ情報を返します。

- DBFIRSTROW ― バッファの最初のローの番号を返します。
- DBLASTROW ― バッファの最後のローの番号を返します。
- dbgetrow ― ロー・バッファ内の指定のローを読み込みます。アプリケーションでこのルーチンを利用すれば、dbnextrow によってすでに読み込まれているバッファ内のローにアクセスできます。
- dbclrbuf ― ロー・バッファからローを削除します。

コマンド・ステータスの情報

次のルーチンは、コマンド・バッチの現在の状態についての情報を返します。この中には、dbresults によって現在処理中のコマンドである、「現在のコマンド」についての情報を返すものもあります。

- DBCURCMD ― バッチ内の現在のコマンドの番号を返します。
- dbgetoff ― 指定の Transact-SQL 構成体がコマンド・バッファ内にあるかどうかを調べます。このルーチンは、DBOFFSET オプションとともに使用します。
- DBMORECMDS ― バッチ内に他にもコマンドがあるかどうかを示します。
- DBCMDROW ― 現在のコマンドがローを返すコマンド (つまり、select コマンドまたは select を含むストアド・プロシージャ) かどうかを示します。
- DBROWS ― 現在のコマンドが実際にローを返したかどうか示します。
- DBCOUNT ― コマンドによって影響を受けたローの数を返します。
- DBNUMORDERS ― select コマンドの order by 句に指定されているカラムの数を返します。
- dbordercol ― select コマンドの order by 句にあるカラムの ID を返します。

ブラウズ・モード

ブラウズ・モードを利用すると、データベースのローをブラウズし、1 つずつローの値を更新することができます。プログラムの観点からは、このプロセスはいくつかのステップで構成されます。ローをブラウズして更新できるようにするには、そのローをデータベースからプログラム変数に転送しなければならないからです。

ブラウズするローはデータベースにある実際のローではなく、プログラム変数にあるコピーであるため、プログラムでの変数の値に対する変更が、元のデータベースのローの更新に使用できることを保証しなければなりません。特にマルチユーザの場合は、あるユーザが行ったデータベースへの更新を、そのすぐ後に別のユーザが無意識のうちに上書きすることがないようにする必要があります。一般に、アプリケーションはデータベースから複数のローを一度に選択しますが、アプリケーションのユーザがブラウズおよび更新できるのは一度に 1 ローだけであるため、このことが問題になります。ブラウズ可能なデータベース・テーブル内の timestamp カラムの情報を利用すれば、この種のマルチユーザによる更新を規制できます。

アプリケーションでブラウズ・モード・ルーチンを使用すれば、アドホック・クエリも扱うことができます。データベース・テーブルを更新する複雑なアドホック・クエリの構造を検査するために、さまざまなルーチンから返される情報を利用できます。

概念的には、ブラウズ・モードは次の 3 つの手順から構成されます。

1　1 つあるいは複数のデータベース・テーブルから導出されたカラムを含む結果ローを選択します。

2　該当する場合は、結果ローのカラムの値を 1 ローずつ変更します (これは実際のデータベースのローではありません)。

3　結果ローの新しい値を使用して、元のデータベース・テーブルを一度に 1 ローずつ更新します。

プログラムでは、上記の手順は次のように実装されます。

1　select コマンドを実行して、結果カラムが含まれる結果ローを生成します。この select コマンドには、for browse オプションが含まれている必要があります。

2　結果カラムの値をプログラム変数に 1 ローずつコピーします。

3　必要に応じて、変数の値を変更します (たとえばユーザの入力への応答として)。

4 必要に応じて、update コマンドを実行し、現在の結果ローに対応するデータベース・ローを更新します。マルチユーザによる更新を処理するには、update コマンドの where 句でタイムスタンプ・カラムを参照する必要があります。この場合の where 句は、dbqual 関数によって取得できます。

5 結果ローごとに上記の 2、3、4 を繰り返します。

ブラウズ・モードを使用するには、次の条件が満たされていなければなりません。

- select コマンドは、キーワード for browse で終了しなければなりません。
- 更新するテーブルは「ブラウズ可能」でなければなりません (つまり、各テーブルにユニーク・インデックスとタイムスタンプ・カラムが必要です)。ブラウズ・モードのテーブルのローはユニークであるので、ブラウズ・モード・テーブルに対する select では、キーワード distinct は意味はありません。
- 更新で使用する結果カラムは「更新可能」、つまりブラウズ可能なテーブルから導出されたカラムでなければなりません。max(colname) などの SQL 式の結果は使用できません。つまり、結果カラムと更新されるデータベース・カラムの間に正しい対応関係がなければなりません。さらに、ブラウズ・モードを使用するには、通常は 2 つの接続 (DBPROCESS ポインタ) が必要です。1 つはデータの選択、もう 1 つは選択したデータに基づく更新に使用します。

ブラウズ・モードのプログラミングの例については、DB-Library に含まれるサンプル・プログラム、*example6.c* および *example7.c* を参照してください。「サンプル・プログラム」(42 ページ) を参照してください。

次にブラウズ・モードのためのルーチンを示します。

- dbqual ― ブラウズ可能なテーブルの現在のローの更新に適した where 句へのポインタを返します。
- dbfreequal ― dbqual によって割り付けられたメモリを解放します。
- dbtsnewval ― ブラウズ・モードでの更新後のタイムスタンプ・カラムの新しい値を返します。
- dbtsnewlen ― ブラウズ・モードでの更新後のタイムスタンプ・カラムの新しい値の長さを返します。
- dbtsput ― タイムスタンプ・カラムの新しい値を、DBPROCESS 内の指定のテーブルの現在のローに格納します。

- dbcolbrowse ― 結果カラムの導出元がブラウズ・モードを介して更新可能であるかどうかを示します。
- dbcolsource ― 指定の結果カラムの導出元であるデータベース・カラムの名前へのポインタを返します。
- dbtabbrowse ― 特定のテーブルがブラウズ・モードを介して更新可能であるかどうかを示します。
- dbtabcount ― 現在の select コマンドの影響を受けるテーブルの数を返します。
- dbtabname ― 番号を基にテーブル名を返します。
- dbtabsource ― 特定の結果カラムの導出元であるテーブルの名前と番号を返します。

text と image の処理

text と image は、大きなテキストまたはイメージの値を保存するために設計された、Adaptive Server Enterpris のデータ型です。text データ型は、2,147,483,647 バイトまでの印刷可能文字を保持でき、image データ型は、2,147,483,647 バイトまでのバイナリ・データを保持できます。

text 値と image 値は、非常に大きくなることがあるため、実際にはデータベース・テーブルには格納されません。代わりに text 値または image 値へのポインタがテーブルに格納されます。このポインタを「テキスト・ポインタ」と呼びます。

競合するアプリケーションどうしが、データベースに加えた変更を互いに消去しあうことがないように、個々の text カラムや image カラムにはタイムスタンプが関連付けられています。このタイムスタンプを、「テキスト・タイムスタンプ」と呼びます。

DB-Library/C アプリケーションで、dbwritetext を使用して text データや image データをテーブルに挿入するには、次の手順を実行する必要があります。

1　insert コマンドを使用して、text 値と image 値を除くすべてのデータをローに挿入します。

2　update コマンドを使用してローを更新し、text カラムまたは image カラムの値を NULL に設定します。null 値を持つ text カラムまたは image カラムのローが、有効なテキスト・ポインタを持つのは、その null 値が update 文で明示的に入力された場合だけであるので、この手順は重要です。

3 select コマンドを使用して、このローを検索します。text または image の値が含まれているカラムを明示的に選択してください。これは、アプリケーションの DBPROCESS に正しいテキスト・ポインタとテキスト・タイムスタンプ情報を設定するために必要な手順です。アプリケーションは、この select によって返されたデータを破棄します。

4 dbtxtptr を呼び出して、DBPROCESS からテキスト・ポインタを取り出します。

5 dbtxtimestamp を呼び出して、DBPROCESS からテキスト・タイムスタンプを取り出します。

6 text 値または image 値を Adaptive Server Enterprise に書き込みます。アプリケーションは次のどちらかを実行できます。

- 一度の dbwritetext 呼び出しで値を書き込む。
- dbwritetext と dbmoretext を使用して、値をいくつかのまとまりに分けて書き込む。

7 アプリケーションがこの text または image の値に対して再度更新を行う場合は、dbwritetext オペレーションが正常に終了したときに Adaptive Server Enterprise から返される新しいテキスト・タイムスタンプを保存することもあります。新しいテキスト・タイムスタンプには、dbtxtsnewval を使用してアクセスします。また、後で取り出せるように dbtxtsput を使用して保存します。

データベース・テーブルの text カラムと image カラムを更新するときは、次のルーチンを使用できます。

- dbreadtext － Adaptive Server Enterprise から text 値または image 値を読み込みます。
- dbwritetext － Adaptive Server Enterprise へ text 値または image 値を送ります。
- dbmoretext － Adaptive Server Enterprise へ text 値または image 値の一部を送ります。
- dbtxptr － 現在の結果ロー内のカラムに対するテキスト・ポインタを返します。
- dbtxtimestamp － 現在の結果ロー内のカラムに対するテキスト・タイムスタンプの値を返します。
- dbtxtsnewval － dbwritetext を呼び出した後の新しいテキスト・タイムスタンプの値を返します。
- dbtxtsput － DBPROCESS の現在のロー内の指定のカラムに、新しいテキスト・タイムスタンプの値を格納します。

データ型変換

DB-Library/C の dbconvert ルーチンと dbconvert_ps ルーチンによって、ほとんどのサーバ・データ型の間の変換がサポートされます。サーバのデータ型については、「データ型」(443 ページ) を参照してください。

結果カラムをプログラム変数にバインドする dbbind ルーチン、dbbind_ps ルーチン、dbaltbind ルーチン、dbaltbind_ps ルーチンは、型変換にも使用できます。これらのルーチンには、それぞれ受け取るプログラム変数のデータ型を指定するパラメータがあります。サーバから返されるデータのデータ型が異なる場合は、通常は、パラメータによって指定されているデータ型に自動的に変換されます。

これらのルーチンを使用して、データ型変換を行います。

- dbconvert_ps ― あるサーバ・データ型から別のデータ型にデータを変換します。numeric データ型および decimal データ型に対する精度と位取りをサポートします。
- dbconvert ― あるサーバ・データ型から別のデータ型にデータを変換します。
- dbwillconvert ― 指定されたデータ型変換をサポートしているかどうかを示します。

プロセス制御フロー

アプリケーションは、次のルーチンによって、サーバとの対話における動作をスケジューリングできます。

- dbsetbusy ― DB-Library/C が読み込みを行っているとき、またはサーバから結果を読み込むために待機しているときに、ユーザ提供関数を呼び出します。
- dbsetidle ― DB-Library/C がサーバからの読み込みを終了したときに、ユーザ提供関数を呼び出します。
- dbsetinterrupt ― サーバからの読み込みを待つ間に、ユーザ提供関数を呼び出して割り込みを処理します。
- DBIORDESC (UNIX のみ) ― サーバからのデータの読み込みに使用する UNIX ファイル記述子にアクセスします。これにより、アプリケーションは複数の入力データ・ストリームに応答できます。

- DBIOWDESC (UNIX のみ) ― サーバへのデータの書き込みに使用する UNIX ファイル記述子にアクセスします。これにより、アプリケーションは複数の入力および出力データ・ストリームを効率的に利用できます。
- DBRBUF (UNIX のみ) ― DB-Library/C ネットワーク・バッファに、まだ読み込まれていないバイトがあるかどうかを調べます。

リモート・プロシージャ・コールの処理

リモート・プロシージャ・コールは、リモート・サーバに存在するストアド・プロシージャの呼び出しです。アプリケーションと他のサーバのどちらかが呼び出しを行います。アプリケーションが行うリモート・プロシージャ・コールの効果は、execute コマンドと同じです。つまり、ストアド・プロシージャを実行し、生成された結果は dbresults によるアクセスが可能です。ただし、多くの場合はリモート・プロシージャ・コールの方が execute コマンドよりも効率的です。実行するプロシージャが、アプリケーションが直接接続しているサーバ以外のサーバに存在する場合は、そのプロシージャ内で実行したコマンドはロールバックできないことに注意してください。

サーバは、別のサーバに対してリモート・プロシージャ・コールを行うことができます。これは、あるサーバ上で実行するストアド・プロシージャに、別のサーバ上のストアド・プロシージャを実行する execute コマンドが含まれている場合に行われます。execute コマンドが実行されると、最初のサーバは 2 番目のサーバにログインし、プロシージャに対するリモート・プロシージャ・コールを実行します。この動作はアプリケーションからの介入なしに行われますが、最初のサーバがログインに使用するリモート・パスワードをアプリケーションで指定することはできます。

リモート・プロシージャ・コールを実行するには、次のルーチンを使用します。

- dbrpcinit ― ストアド・プロシージャへのリモート・プロシージャ・コールを初期化します。
- dbrpcparam ― リモート・プロシージャ・コールにパラメータを追加します。
- dbrpcsend ― リモート・プロシージャ・コールの終了を示します。これによって、サーバは指定されたプロシージャの実行を開始します。

- dbpoll ― dbsqlsend (または dbrpcsend) と dbsqlok の間で呼び出され、サーバの応答が DBPROCESS に到着したかどうかを調べます。
- dbsqlok ― サーバからの結果を待って、サーバが応答した命令の正当性を検証します。このルーチンは、dbsqlsend、dbrpcsend、dbmoretext とともに使用します。dbsqlok 呼び出しが正常に終了したときは、結果を処理するために dbresults を呼び出す必要があります。

レジスタード・プロシージャ・コールの処理

レジスタード・プロシージャは、稼働中の Open Server 内で定義されてインストールされるプロシージャです。レジスタード・プロシージャを使用するには、Open Server のバージョンが 2.0 以降でなければなりません。現時点では、Adaptive Server Enterprise はレジスタード・プロシージャをサポートしていません。

DB-Library/C アプリケーションでは、レジスタード・プロシージャはアプリケーション間の通信と同期の手段となります。これは、Open Server に接続している DB-Library/C アプリケーションがレジスタード・プロシージャの実行を「監視」できるためです。レジスタード・プロシージャが実行されると、監視しているアプリケーションは、プロシージャ名と呼び出し時の引数が含まれるノーティフィケーション (通知) を受け取ります。

注意　DB-Library/C アプリケーションで作成できるレジスタード・プロシージャは、「ノーティフィケーション・プロシージャ」と呼ばれる特殊なタイプのものだけです。ノーティフィケーション・プロシージャは、通常の Open Server レジスタード・プロシージャとは異なり、実行可能な文は含まれません。

たとえば、次のような状況を想定します。

- stockprice は、リアルタイムに株価をモニタする DB-Library/C アプリケーションである。
- price_change は、stockprice アプリケーションが Open Server 内に作成したノーティフィケーション・プロシージャで、株式銘柄名と株価変動をパラメータとして持つ。
- sellstock は、株式を売るアプリケーションで、price_change が実行されたときにノーティフィケーションを受けるよう要求している。

モニタリング・アプリケーションである stockprice は、Extravagant Auto Parts の株価が $1.10 上昇したことを検知すると、「Extravagant Auto Parts」と「+1.10」をパラメータとして渡して price_change を実行します。

price_change が実行されると、Open Server は sellstock にノーティフィケーション (通知) を送ります。これには、プロシージャ名 (price_change) とプロシージャに渡す引数 (「Extravagant Auto Parts」と「+1.10」) が含まれます。sellstock は、このノーティフィケーションに含まれる情報を使用して、Extravagant Auto Parts の株式を 100 株売却することを決定します。

price_change は、stockprice アプリケーションと sellstock アプリケーションが通信を行うための手段です。

通信手段としてのレジスタード・プロシージャには、次の利点があります。

- レジスタード・プロシージャを実行するための 1 回の呼び出しにより、プロシージャの実行を多くのアプリケーションに通知できます。プロシージャを実行するアプリケーションは、ノーティフィケーションを要求しているクライアントの数や、どのクライアントが要求しているかを知る必要はありません。
- レジスタード・プロシージャによる通信メカニズムは、サーバベースです。Open Server は、接続アドレスの集中レポジトリとなります。これにより、クライアント・アプリケーションどうしが通信するとき、互いに直接接続するのではなく、クライアントはそれぞれサーバに接続します。

DB-Library/C アプリケーションでは、次のことが可能です。

- Open Server 内にレジスタード・プロシージャを作成する
- レジスタード・プロシージャを削除する
- Open Server 内に定義されているすべてのレジスタード・プロシージャのリストを作成する
- 特定のレジスタード・プロシージャが実行されたときにノーティフィケーションを受け取ることを要求する
- 特定のレジスタード・プロシージャが実行されたときにノーティフィケーションを受け取ることの要求を取り消す
- すべてのレジスタード・プロシージャ・ノーティフィケーションのリストを作成する
- レジスタード・プロシージャを実行する

- レジスタード・プロシージャの実行ノーティフィケーションをアプリケーションが受け取ったときに呼び出されるユーザ提供ハンドラをインストールする
- Open Server をポーリングして、保留中のレジスタード・プロシージャ・ノーティフィケーションがあるかどうかを確認する

レジスタード・プロシージャ・ルーチンを次に示します。

- dbnpcreate ― ノーティフィケーション・プロシージャを作成します。
- dbnpdefine ― ノーティフィケーション・プロシージャを定義します。
- dbregdrop ― レジスタード・プロシージャを削除します。
- dbreglist ― 現在 Open Server 内に定義されているすべてのレジスタード・プロシージャのリストを返します。
- dbreghandle ― レジスタード・プロシージャ・ノーティフィケーションのハンドラ・ルーチンをインストールします。
- dbreginit ― レジスタード・プロシージャの実行を開始します。
- dbregnowatch ― レジスタード・プロシージャが実行されたときにノーティフィケーションを受け取ることの要求をキャンセルします。
- dbregparam ― レジスタード・プロシージャのパラメータを定義します。
- dbregexec ― レジスタード・プロシージャを実行します。
- dbregwatch ― レジスタード・プロシージャが実行されたときにノーティフィケーションを受け取ることを要求します。
- dbregwatchlist ― DBPROCESS が監視しているレジスタード・プロシージャのリストを返します。
- dbpoll ― レジスタード・プロシージャ・ノーティフィケーションを使用するアプリケーションで、ノーティフィケーションが到着しているかどうかを確認するために使用します。

ゲートウェイ・パススルー・ルーチン

パススルー・ルーチンは、Open Server ゲートウェイ・アプリケーションで呼び出すことができます。このルーチンを使用すると、DB-Library/C アプリケーションで Tabular DataStream™ (TDS) パケット全体の送受信と TDS パケット・サイズの設定を行うことができます。

TDS は、クライアントとサーバの間で、要求と要求結果を転送するために使用されるアプリケーション・プロトコルです。次のルーチンは、Open Server Server-Library ルーチンである srvrecvpassthru および srvsendpassthru とともに使用します。

- dbrecvpassthru － Open Server から TDS パケットを受信します。
- dbsendpassthru － Open Server へ TDS パケットを送信します。

srvrecvpassthru および srvsendpassthru については、『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

datetime と money

次のルーチンは、datetime データ型と money データ型を操作します。datetime データ型と money データ型には、長いバージョンの DBDATETIME および DBMONEY と、短いバージョン (4 バイト) の DBDATETIME4 および DBMONEY4 があります。次のリストに示す DBDATETIME4 のルーチンはいずれも、DBDATETIME にも使用できます。また、DBMONEY4 のルーチンはいずれも DBMONEY にも使用できます。たとえば、リストにある dbmny4add は dbmnyadd としても使用できます。

- dbdate4cmp － 2 つの DATETIME4 の値を比較します。
- dbdate4zero － DBDATETIME4 の値を初期化します。
- dbmny4add － 2 つの DBMONEY4 の値を加算します。
- dbmny4cmp － 2 つの DATETIME4 の値を比較します。
- dbmny4copy － DBMONEY4 の値をコピーします。
- dbmny4divide － DBMONEY4 の値を別の値で除算します。
- dbmny4minus － DBMONEY4 の値の符号を反転します。
- dbmny4mul － DBMONEY4 の値を乗算します。
- dbmny4sub － DBMONEY4 の値を減算します。
- dbmny4zero － DBMONEY4 の値を初期化します。

- dbmnydec － DBMONEY の値を減らします。
- dbmnydown － DBMONEY の値を正の整数で除算します。
- dbmnyinc － DBMONEY の値を増やします。
- dbmnyinit － dbmnyndigit を呼び出すために DBMONEY の値を準備します。
- dbmnymaxneg － DBMONEY の負の最大値を返します。
- dbmnymaxpos － DBMONEY の正の最大値を返します。
- dbmnyndigit － DBMONEY の値の右端の桁を DBCHAR として返します。
- dbmnyscale － DBMONEY の値を乗算し、指定した数を加算します。

終了処理

次のルーチンは、アプリケーションとサーバとの間の接続を切断します。

- dbexit － すべての DBPROCESS 構造体をクローズし、割り付けを解除します。また、dbinit によって初期化された構造体をすべてクリーンアップします。
- dbclose － 1 つの DBPROCESS 構造体をクローズし、割り付けを解除します。

機密保護のサポート

次のルーチンは、Adaptive Server Enterprise に対して実行される DB-Library アプリケーションをセキュリティで保護します。

- DBSETLENCRYPT － Adaptive Server Enterprise にログインするときにパスワードの暗号化を使用するかどうかを指定します。
- dbsechandle － セキュア・ログインを処理するためのユーザ関数をインストールします。

- bcp_options ― バルク・コピーの、セキュリティ・ラベル・オプションである BCPLABELED などのオプションを設定します。

注意　最初に DB-Library のリリースを 10.0 に設定しなければ、DBSETLENCRYPT を呼び出したときにエラーが発生します。DBSETLENCRYPT を呼び出す前に、dbsetversion を使用して DB-Library のリリースを 10.0 に設定してください。

その他のルーチン

アプリケーションによっては、次のルーチンが便利です。

- dbsetavail ― DBPROCESS に対して、通常の使用が可能というマーク (available) を付けます。
- DBISAVAIL ― DBPROCESS が通常使用可能かどうかを示します。
- dbname ― 現在のデータベースの名前を返します。
- dbchange ― コマンド・バッチが現在のデータベースを変更したかどうかを示します。
- dbsetuserdata ― DBPROCESS 構造体を使用して、ユーザが割り付けたデータへのポインタを保存します。このルーチンと dbgetuserdata を使用すると、ユーザ・データを特定の DBPROCESS に関連付けることができます。これらのルーチンの重要な用途として、サーバ・メッセージ・ハンドラとハンドラをトリガしたプログラム・コードとの間の情報転送があります。
- dbgetuserdata ― DBPROCESS 構造体からユーザ割り付けのデータを指すポインタを返します。
- dbreadpage ― Adaptive Server Enterprise から 1 ページ分のバイナリ・データを読み込みます。
- dbwritepage ― 1 ページ分のバイナリ・データを Adaptive Server Enterprise に書き込みます。
- dbsetconnect ― このルーチンのサーバ接続情報を設定します。

2 フェーズ・コミット・サービス・ライブラリ

このライブラリのルーチンにより、アプリケーションは、2 つ以上の Adaptive Server Enterprise 間での更新を調整できます。

「第 4 章 2 フェーズ・コミット・サービス」を参照してください。

DB-Library 版 MIT Kerberos

DB-Library では、MIT Kerberos のセキュリティ・メカニズムを使用して、ネットワーク・サービスと相互認証サービスを提供します。この機能を使用すると、古い Sybase アプリケーションで Kerberos 認証サービスを使用できます。しかも、変更や再コンパイルの必要性は多くありません。

次の DB-Library マクロにより、Kerberos がサポートされます。

- DBSETLNETWORKAUTH ― ネットワーク・ベースの認証を有効または無効にします。
- DBSETLMUTUALAUTH ―接続のセキュリティ・メカニズムの相互認証を有効または無効にします。
- DBSETLSERVERPRINCIPAL ― 必要に応じてサーバのプリンシパル名を設定します。

注意　DB-Library でサポートされる Kerberos セキュリティ・メカニズムの機能は、ネットワーク認証サービスと相互認証サービスのみです。

❖ DB-Library への MIT-Kerberos のインストール

次の手順では、DB-Library への MIT Kerberos のインストールに関する基本的な事項を説明します。詳細については、Sybase SDK DB-Lib Kerberos Authentication Option 15.5 の『Installation and Release Bulletin』を参照してください。

1　Sybase SDK DB-Lib Kerberos Authentication Option 15.5 を購入します。

2　SDK 15.5 を使用して Sybase SDK DB-Lib Kerberos Authentication Option 15.5 をインストールします。

3　DB-Library に *sybdb.h* ではなく *sybdbn.h* をインクルードします。

4 dbsetversion を使用して DB-Library のバージョンを DBVERSION_100 以上に設定します。

5 次の API のうち 1 つ以上を呼び出します。

DBSETLNETWORKAUTH(LOGINREC **loginrec*, DBBOOL *enable*)

DBSETLMUTUALAUTH(LOGINREC **loginrec*, DBBOOL *enable*)

DBSETLSERVERPRINCIPAL(LOGINREC **loginrec*, char **name*)

6 DB-Library をリコンパイルします。

サンプル・プログラム

DB-Library ルーチンの使用方法とその機能を示すサンプル・プログラムもあります。サンプル・プログラムは次のディレクトリに格納されています。

- UNIX では *$SYBASE/$SYBASE_OCS/sample/dblibrary*
- Windows では *%SYBASE%¥%SYBASE_OCS%¥sample¥dblib*

詳細については、使用しているプラットフォームの『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

第 2 章

ルーチン

この章では、DB-Library の各ルーチンについて説明します。

ルーチン	説明	ページ
dbcmd	DBPROCESS コマンド・バッファにテキストを追加する。	96
DBCMDROW	現在のコマンドがローを返せるかどうかを調べる。	98
dbcolbrowse	通常の結果カラムのソースが、DB-Library のブラウズ・モード機能を介して更新可能かどうかを調べる。	99
dbcollen	通常の結果カラム内のデータの最大長を返す。	100
dbcolname	通常の結果カラムの名前を返す。	101
dbcolsource	指定された通常の結果カラムの導出元であるデータベース・カラムの名前を指すポインタを返す。	102
dbcoltype	通常の結果カラムのデータ型を返す。	104
dbcoltypeinfo	numeric 型または decimal 型の通常の結果カラムの精度と位取りの情報を返す。	105
dbcolutype	通常の結果カラムのユーザ定義データ型を返す。	106
dbconvert	データを別のデータ型に変換する。	108
dbconvert_ps	データを別のデータ型に変換する。numeric データ型と decimal データ型の精度と位取りをサポートする。	112
DBCOUNT	Transact-SQL コマンドによる影響を受けたローの数を返す。	118
DBCURCMD	現在のコマンドの番号を返す。	120
DBCURROW	現在読み込み中のローの番号を返す。	120
dbcursor	フェッチ・バッファ内の特定のローを挿入、更新、削除、ロック、またはリフレッシュする。	121
dbcursorbind	カーソル・カラムのバインド情報を登録する。	123
dbcursorclose	指定のハンドルに対応しているカーソルをクローズして、そのカーソルに属するすべてのデータを解放する。	125
dbcursorcolinfo	オープン・カーソル内の指定カラム番号のカラム情報を返す。	126
dbcursorfetch	ローのブロックをフェッチして、dbcursorbind でユーザが宣言したプログラム変数に出力する。	127
dbcursorinfo	キーセットが結果セットの末尾になった場合、キーセット内のカラム数およびロー数を返す。	130
dbcursoropen	カーソルをオープンして、スクロール・オプション、同時実行オプション、およびフェッチ・バッファのサイズ (1 回のフェッチで取り出すローの数) を指定する。	130
dbdata	通常の結果カラム内のデータを指すポインタを返す。	134
dbdate4cmp	2 つの DBDATETIME4 値を比較する。	136
dbdate4zero	DBDATETIME4 変数を 1900 年 1 月 1 日午前 12 に初期化する。	136

ルーチン	説明	ページ
エラー	すべての DB-Library エラーとエラー重大度のリスト。	418
オプション	すべての DB-Library オプションのリスト。	436
データ型	DB-Library で使用されるデータ型とデータ型の記号定数。	443

db12hour

説明

指定された言語が、12 時間制と 24 時間制のどちらを使用するかを調べます。

構文

```
DBBOOL db12hour(dbproc, language)

DBPROCESS    *dbproc;
char         *language;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

language

言語の名前です。

戻り値

language が 12 時間制を使用する場合は「TRUE」、24 時間制の場合は「FALSE」を返します。

使用法

- db12hour は、*language* が 12 時間制を使用している場合は「TRUE」、24 時間制を使用している場合は「FALSE」を返します。
- *language* が NULL の場合は、*dbproc* で現在使用中の言語が指示されたものとみなされます。*language* と *dbproc* の両方が NULL の場合は、DB-Library のデフォルト言語 (今後のすべての dbopen 呼び出しに使用する言語) が指定されたものとみなされます。
- db12hour は、dbsqlexec を使用して DBDATETIME 値を取得したり操作したりするときに便利です。DBDATETIME 値を文字列に変換すると、dbconvert と dbbind は、常に、DBDATETIME 値の月コンポーネントをローカル言語で返しますが、日時順については U.S. English 形式 (month-day-year、12 時間制) が使用されます。db12hour の戻り値は、12 時間制より 24 時間制の方が好ましい場合に追加操作が必要となることをアプリケーションに伝えます。

- 次のコードは、db12hour の例です。

```
DBBOOL time_format;
DBCHAR s_date[40];

/*
** Find out whether 12-hour or 24-hour time is
** used.
*/
time_format = db12hour(dbproc, "FRANCAIS");

/* Put a command into a command buffer */
dbcmd(dbproc, "select start_date from info_table");

/* Send the command to the Adaptive Server
Enterprise */
dbsqlexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/*
** Bind column data (start_date) to the program
** variable (s_date)
*/
dbbind(dbproc, 1, NTBSTRINGBIND, 0, s_date);

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
      /*
      ** If we want 24-hour time, re-format
      ** s_date accordingly.
      */
      if (time_format == TRUE)
           format_24(s_date);

      printf("Next start date:%s¥n", s_date);
}
```

参照　dbdateorder、dbdayname、dbmonthname、dbsetopt

dbadata

説明

計算カラムのデータを指すポインタを返します。

構文

BYTE *dbadata(dbproc, computeid, colnum)

```
DBPROCESS    *dbproc;
int          computeid;
int          colnum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid

対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

colnum

対象となるカラムの番号です。最初に返されるカラムの番号は 1 です。計算カラムが返される順序は、計算カラムが最初に指定されたときの順序ではなく、select リスト内の対応するカラムの順序によって決まります。たとえば、次のクエリの「sum(price)」の結果を参照するには、*colnum* の値として 2 ではなく 1 を指定します。

```
select price, advance from titles
compute sum(advance), sum(price)
```

colnum の値は、select リスト内の計算カラムの絶対位置ではなく、相対順序で決まります。たとえば、上記の select 文を次のように変えたとします。

```
select title_id, price, advance from titles
compute sum(advance), sum(price)
```

「sum(price)」の *colnum* の値は、この場合も 2 ではなく 1 です。これは、select リストの「title_id」カラムは計算カラムではないので、計算カラムの番号を決定するときに無視されるためです。

戻り値

特定の計算の特定のカラムのデータを指す BYTE ポインタです。このポインタを適切なデータ型にキャストするようにしてください。該当するカラムや計算がない場合、またはデータの値が NULL の場合は、NULL を指す BYTE ポインタが返されます。

DB-Library は、BYTE ポインタが指すデータ領域の割り付けと解放を行います。この領域は上書きしないでください。

使用法

- dbnextrow を呼び出した後で、このルーチンを使用すると、計算ロー内の特定カラムのデータを指すポインタが返されます。このデータは、NULL で終了するデータではありません。dbadlen を使用すれば、データの長さがわかります。
- 整数データのカラムの合計や平均値を算出するとき、サーバは、カラムのサイズに関係なく常に 4 バイトの整数を返します。したがって、このような計算の結果を格納する変数は、DBINT として宣言してください。
- 次のコードは、dbadata の例です。

```
DBPROCESS    *dbproc;
int           rowinfo;
DBINT         sum;

/*
** First, put the commands into the command
** buffer
*/
dbcmd(dbproc, "select db_name(dbid), dbid, size
     from sysusages");
dbcmd(dbproc, " order by dbid");
dbcmd(dbproc, " compute sum(size) by dbid");

/*
** Send the commands to Adaptive Server Enterprise
and start
** execution
*/
dbsqlexec(dbproc);

/* Process the command */
dbresults(dbproc);

/* Examine the results of the compute clause */
while((rowinfo = dbnextrow(dbproc)) !=
    NO_MORE_ROWS)
{
    if (rowinfo == REG_ROW)
       printf("regular row returned.¥n");
    else
    {
       /*
```

```
                ** This row is the result of a compute
                ** clause, and "rowinfo" is the computeid
                ** of this compute clause.
                */

                sum = *(DBINT *)(dbadata(dbproc, rowinfo,
                    1));
                printf("sum = %ld¥n", sum);
            }
        }
```

- 関数 dbaltbind は、計算データを自動的にプログラム変数にバインドします。このルーチンはデータをコピーしますが、一般に dbadata よりも簡単に使用できます。さらに、dbaltbind には便利な型変換機能があります。アプリケーションは、この機能を使用して、結果文字列に NULL ターミネータを追加したり、通貨や日時のデータを印刷可能文字列に変換したりすることができます。

参照

dbadlen、dbaltbind、dbaltlen、dbalttype、dbgetrow、dbnextrow、dbnumalts

dbadlen

説明

計算カラムのデータの実際の長さを返します。

構文

DBINT dbadlen(dbproc, computeid, column)

```
DBPROCESS    *dbproc;
int          computeid;
int          column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定の計算カラムのデータのバイト単位の長さです。このようなカラムまたは compute 句がない場合、dbadlen は -1 を返します。データが NULL 値の場合は、dbadlen は 0 を返します。

使用法

- このルーチンは、特定の計算カラムのデータの実際の長さを返します。
- データの最大長を調べるには、dbaltlen ルーチンを使用してください。このデータへのポインタを取得するには、dbadata を使用してください。
- 次のコードは、dbadlen の例です。

```
DBPROCESS    *dbproc;
char          biggest_name[MAXNAME+1];
int           namelen;
int           rowinfo;

/* put the command into the command buffer */
dbcmd(dbproc, "select name from sysobjects");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(name)");

/*
** Send the command to Adaptive Server Enterprise
and start
** execution.
*/
dbsqlexec(dbproc);

/* process the command */
dbresults(dbproc);

/* examine each row returned by the command */
while((rowinfo = dbnextrow(dbproc)) !=
     NO_MORE_ROWS)
{
     if (rowinfo == REG_ROW)
        printf("regular row returned.¥n");
     else
     {
        /*
        ** This row is the result of a compute
        ** clause, and "rowinfo" is the computeid
        ** of this compute clause.
        */
         namelen = dbadlen(dbproc, rowinfo, 1);
         strncpy(biggest_name,
```

```
                    (char *)dbadata(dbproc, rowinfo, 1),
                    namelen);

                /*
                ** Data pointed to by dbadata() is not
                ** null-terminated.
                */
                biggest_name[namelen] = '¥0';

                printf("biggest name = %s¥n",
        biggest_name);
            }
        }
```

参照 dbadata、dbaltlen、dbalttype、dbgetrow、dbnextrow、dbnumalts

dbaltbind

説明 計算カラムをプログラム変数にバインドします。

構文

```
RETCODE dbaltbind(dbproc, computeid, column, vartype,
        varlen, varaddr)

DBPROCESS    *dbproc;
int          computeid;
int          column;
int          vartype;
DBINT        varlen;
BYTE         * varaddr;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
対象となる特定の計算ローを表す ID です。1 つの select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は、1 です。

column
プログラム変数にコピーされるロー・データのカラム番号です。最初のカラムの番号は 1 です。計算カラムが返される順序は、計算カラムが最初に指定されたときの順序ではなく、select リスト内の対応するカラムの順序によって決まります。たとえば、次のクエリの「sum(price)」の結果を参照するには、*column* の値として 2 ではなく 1 を指定します。

```
select price, advance from titles
compute sum(advance), sum(price)
```

column の値は、select リスト内の計算カラムの絶対位置ではなく、相対順序で決まります。たとえば、上記の select 文を次のように変えたとします。

```
select title_id, price, advance from titles
compute sum(advance), sum(price)
```

「sum(price)」の *column* の値は、この場合も 2 ではなく 1 です。これは、select リストの「title_id」カラムは計算カラムではないので、計算カラムの番号を決定するときに無視されるためです。

vartype
バインドのデータ型を表します。このデータ型は、DBPROCESS からのデータのコピーを受け取るプログラム変数のデータ型と対応していなければなりません。次の表に、*vartype* の値とプログラム変数の型の対応を示します。

dbaltbind は、さまざまな型変換をサポートするため、*vartype* は SQL クエリから返される型と異なっていてもかまいません。たとえば、SYBMONEY 型の結果を、FLT8BIND を介して DBFLT8 型のプログラム変数にバインドできます。このとき、適切なデータ変換が自動的に行われます。DB-Library によって行われるデータ変換のリストについては、dbwillconvert のリファレンス・ページを参照してください。

注意　dbaltbind には、numeric データ型および decimal データ型に対して精度と位取りを明示的に指定する機能はありません。dbaltbind で numeric または decimal のデータを扱うとき、numeric カラムまたは decimal カラムへのバインドの場合はバインド元データの精度と位取りが使用され、それ以外の場合はデフォルトの精度 18 と位取り 0 が使用されます。精度と位取りの値を明示的に指定する場合は、dbaltbind_ps を使用します。dbaltbind を呼び出すことは、*typeinfo* の値に NULL を指定して dbaltbind_ps を呼び出すのと同じことです。

DB-Library によって使用される型定義のリストについては、「データ型」(443 ページ) を参照してください。

表 2-1 は、dbaltbind が認識する有効な *vartype* の値と、各 vartype に対応するサーバ・データ型とプログラム変数型のリストです。

表 2-1：dbaltbind のバインド型

vartype	プログラム変数タイプ	サーバ・データ型
CHARBIND	DBCHAR	SYBCHAR
STRINGBIND	DBCHAR	SYBCHAR
NTBSTRINGBIND	DBCHAR	SYBCHAR
VARYCHARBIND	DBVARYCHAR	SYBCHAR
BINARYBIND	DBBINARY	SYBBINARY
VARYBINBIND	DBVARYBIN	SYBBINARY
TINYBIND	DBTINYINT	SYBINT1
SMALLBIND	DBSMALLINT	SYBINT2
INTBIND	DBINT	SYBINT4
FLT8BIND	DBFLT8	SYBFLT8
REALBIND	DBREAL	SYBREAL
NUMERICBIND	DBNUMERIC	SYBNUMERIC
DECIMALBIND	DBDECIMAL	SYBDECIMAL
BITBIND	DBBIT	SYBBIT
DATETIMEBIND	DBDATETIME	SYBDATETIME
SMALLDATETIMEBIND	DBDATETIME4	SYBDATETIME4
MONEYBIND	DBMONEY	SYBMONEY
SMALLMONEYBIND	DBMONEY4	SYBMONEY4
BOUNDARYBIND	DBCHAR	SYBBOUNDARY
SENSITIVITYBIND	DBCHAR	SYBSENSITIVITY

警告！ライブラリのバージョンが DBVERSION_100 以上に設定されていない場合は (設定するには dbsetversion を使用します)、*vartype* に BOUNDARYBIND、DECIMALBIND、NUMERICBIND、または SENSITIVITYBIND を使用するとエラーになります。

計算ローから SYBTEXT データや SYBIMAGE データが返されることはないので、これらのデータ型は前述の表には含まれていません。

前述の表にサーバ・データ型が含まれているのは、参考のためです。指定する *vartype* は、特定のサーバ・データ型に対応しているものでなくてもかまいません。前述したように、dbaltbind によって、指定の *vartype* にサーバ・データが変換されるためです。

次の表は、使用可能な文字データ表現を示します。これらの違いは、データがブランク埋め込みであるか、NULL 文字で終了するかによって生じます。

vartype	プログラムの型	埋め込み文字	ターミネータ
CHARBIND	DBCHAR	ブランク	なし
STRINGBIND	DBCHAR	ブランク	¥0
NTBSTRINGBIND	DBCHAR	なし	¥0
VARYCHARBIND	DBVARYCHAR	なし	なし
BOUNDARYBIND	DBCHAR	なし	¥0
SENSITIVITYBIND	DBCHAR	なし	¥0

上記の表の「¥0」は、NULL ターミネータ文字を示しています。

整数または浮動小数点のデータを文字バインド型に変換するときにオーバフローが発生すると、結果の値の最初の文字は、エラーを示すアスタリスク (「*」) となります。

バイナリ・データを保管するには、2 とおりの方法があります。

vartype	プログラムの型	埋め込み文字
BINARYBIND	DBBINARY	NULL
VARYBINBIND	DBVARBINARY	なし

整数データのカラムの合計や平均値を算出するとき、サーバは、カラムのサイズに関係なく常に 4 バイトの整数を返します。したがって、このような計算の結果を格納する変数は DBINT として宣言し、バインドの *vartype* には INTBIND を指定してください。

varlen
プログラム変数の長さ (バイト単位)。

vartype の値が MONEYBIND や FLT8BIND などの固定長の型を表すものである場合は、この長さは無視されます。

文字型とバイナリ型の場合は、*varlen* には、NULL ターミネータなどの特別な終了バイトに必要な領域を含む、使用可能なコピー先バッファ領域の全長を指定してください。*varlen* が 0 の場合は、使用可能なすべてのバイトがプログラム変数にコピーされます (char と binary のサーバ・データの場合、使用可能なバイトの総数は、ブランクの埋め込み文字を含む、データベース・カラムの定義された長さと同じです。varchar と varbinary のデータの場合、使用可能なバイトの総数は、カラムに格納されている実際のデータと同じです)。したがって、プログラム変数が、結果を処理するのに十分な大きさであることが確実な場合は、*varlen* を 0 に設定してもかまいません。

varaddr
データのコピー先であるプログラム変数のアドレスです。

戻り値

SUCCEED または FAIL。

カラム番号が正しくない場合、*vartype* で指定されたデータ変換が正しくない場合、または *varaddr* が NULL の場合、dbaltbind は FAIL を返します。

使用法

- このルーチンは、サーバから返された計算カラム・データをプログラム変数にコピーするように DB-Library に指示します (計算カラムは、Transact-SQL select 文の compute 句から生成されます)。計算データが含まれている新しいローが dbnextrow または dbgetrow を介して読み込まれると、その計算ロー内の指定された *column* のデータが、アドレス *varaddr* のプログラム変数にコピーされます。コピーする計算カラムごとに dbaltbind を呼び出す必要があります。すべての計算カラムをプログラム変数にバインドする必要はありません。
- サーバは 2 つのタイプのローを返します。select 文の select リストで指定されたカラムのデータが含まれる通常ローと、compute 句から生成される計算ローです。dbaltbind は計算ローのデータをバインドします。通常ローのデータをバインドするには、dbbind を使用してください。
- dbaltbind 呼び出しは、dbresults 呼び出しの後で、dbnextrow の最初の呼び出しの前に行ってください。

- 標準的な呼び出しのシーケンスを次に示します。

```
DBCHAR   name[20];
DBINT    namecount;

 /* read the query into the command buffer */
 dbcmd(dbproc, "select name from emp compute
      count(name)");

 /* send the query to Adaptive Server Enterprise */
 dbsqlexec(dbproc);

 /* get ready to process the query results */
 dbresults(dbproc);

 /* bind the regular row data (name) */
 dbbind(dbproc, 1, STRINGBIND, (DBINT)0, name);

 /* bind the compute column data (count of name) */
dbaltbind(dbproc, 1, 1, INTBIND, (DBINT) 0,
     (BYTE *) &namecount);

 /* now process each row */
 while (dbnextrow(dbproc) != NO_MORE_ROWS)
 {
     C-code to print or process row data
 }
```

- dbaltbind はデータをプログラム変数にコピーするため、多少のオーバヘッドが発生します。このコピー作業を避けるには、返されたデータに直接アクセスする dbadata ルーチンを使用してください。
- 1 つの結果カラムをバインドできるのは 1 つのプログラム変数だけです。1 つの結果カラムを複数のプログラム変数にバインドした場合は、最後のバインドだけが有効になります。
- サーバからはカラム値として null が返されることもありますが、DB-Library では次のような方法で null 値を処理できます。
  - 各データ型に 1 つずつデフォルト値が事前に定義されています。バインドされたカラムの値が null のときは、自動的にこのデフォルト値が代入されます。dbsetnull 関数を使用することにより、ユーザは、独自の null 代入値を明示的に設定できます。デフォルト代入値のリストについては、dbsetnull 関数のリファレンス・ページを参照してください。

- dbnullbind (計算ローの場合は dbanullbind) を使用して、インジケータ変数をカラムにバインドできます。ローがフェッチされると、カラム値が null であったかどうかを示すようにインジケータ変数の値が設定されます。インジケータ値と意味については、dbnullbind 関数のリファレンス・ページを参照してください。

参照

dbadata、dbaltbind_ps、dbanullbind、dbbind、dbbind_ps、dbconvert、dbconvert_ps、dbnullbind、dbsetnull、dbsetversion、dbwillconvert、「データ型」(443 ページ)

dbaltbind_ps

説明

計算カラムをプログラム変数にバインドします。numeric データ型と decimal データ型の精度と位取りをサポートします。

構文

```
RETCODE dbaltbind_ps(dbproc, computeid, column,
                     vartype, varlen, varaddr,
                     typeinfo)

DBPROCESS     *dbproc;
int           computeid;
int           column;
int           vartype;
DBINT         varlen;
BYTE          *varaddr;
DBTYPEINFO    *typeinfo;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid

対象となる特定の計算ローを表す ID です。1 つの select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は、1 です。

column
プログラム変数にコピーされるロー・データのカラム番号です。最初のカラムの番号は 1 です。計算カラムが返される順序は、計算カラムが最初に指定されたときの順序ではなく、select リスト内の対応するカラムの順序によって決まります。たとえば、次のクエリの「sum(price)」の結果を参照するには、*column* の値として 2 ではなく 1 を指定します。

```
select price, advance from titles
compute sum(advance), sum(price)
```

column の値は、select リスト内の計算カラムの絶対位置ではなく、相対順序で決まります。たとえば、上記の select 文を次のように変えたとします。

```
select title_id, price, advance from titles
compute sum(advance), sum(price)
```

「sum(price)」の *column* の値は、この場合も 2 ではなく 1 です。これは、select リストの「title_id」カラムは計算カラムではないので、計算カラムの番号を決定するときに無視されるためです。

vartype
バインドのデータ型を表します。このデータ型は、DBPROCESS からのデータのコピーを受け取るプログラム変数のデータ型と対応していなければなりません。次の表に、*vartype* の値とプログラム変数の型の対応を示します。

dbaltbind_ps は、さまざまな型変換をサポートするため、*vartype* は SQL クエリから返される型と異なっている可能性があります。たとえば、SYBMONEY 型の結果を、FLT8BIND を介して DBFLT8 型のプログラム変数にバインドできます。このとき、適切なデータ変換が自動的に行われます。DB-Library によって行われるデータ変換のリストについては、dbwillconvert のリファレンス・ページを参照してください。

注意　dbaltbind_ps のパラメータは、DBNUMERIC または DBDECIMAL の変数の精度と位取りに関する情報を指定するパラメータ *typeinfo* が追加されている点を除いて、dbaltbind のパラメータとまったく同じです。

DB-Library によって使用される型定義のリストについては、「データ型」(443 ページ) を参照してください。

表 2-2 は、dbaltbind_ps が認識する有効な *vartype* の値と、各 vartype に対応するサーバ・データ型とプログラム変数型のリストです。

表 2-2：dbaltbind_ps のバインド型

vartype	プログラム変数タイプ	サーバ・データ型
CHARBIND	DBCHAR	SYBCHAR
STRINGBIND	DBCHAR	SYBCHAR
NTBSTRINGBIND	DBCHAR	SYBCHAR
VARYCHARBIND	DBVARYCHAR	SYBCHAR
BINARYBIND	DBBINARY	SYBBINARY
VARYBINBIND	DBVARYBIN	SYBBINARY
TINYBIND	DBTINYINT	SYBINT1
SMALLBIND	DBSMALLINT	SYBINT2
INTBIND	DBINT	SYBINT4
FLT8BIND	DBFLT8	SYBFLT8
REALBIND	DBREAL	SYBREAL
NUMERICBIND	DBNUMERIC	SYBNUMERIC
DECIMALBIND	DBDECIMAL	SYBDECIMAL
BITBIND	DBBIT	SYBBIT
DATETIMEBIND	DBDATETIME	SYBDATETIME
SMALLDATETIMEBIND	DBDATETIME4	SYBDATETIME4
MONEYBIND	DBMONEY	SYBMONEY
SMALLMONEYBIND	DBMONEY4	SYBMONEY4
BOUNDARYBIND	DBCHAR	SYBBOUNDARY
SENSITIVITYBIND	DBCHAR	SYBSENSITIVITY

警告！ライブラリのバージョンが DBVERSION_100 以上に設定されていない場合は (設定するには dbsetversion を使用します)、*vartype* に BOUNDARYBIND、DECIMALBIND、NUMERICBIND、または SENSITIVITYBIND を使用するとエラーになります。

計算ローから SYBTEXT データや SYBIMAGE データが返されることはないので、これらのデータ型は前述の表には含まれていません。

前述の表にサーバ・データ型が含まれているのは、参考のためです。指定する *vartype* は、特定のサーバ・データ型に対応しているものでなくてもかまいません。前述したように、dbaltbind_ps によって、指定の *vartype* にサーバ・データが変換されるためです。

次の表は、使用可能な文字データ表現を示します。これらの違いは、データがブランク埋め込みであるか、NULL 文字で終了するかによって生じます。

vartype	プログラムの型	埋め込み文字	ターミネータ
CHARBIND	DBCHAR	ブランク	なし
STRINGBIND	DBCHAR	ブランク	¥0
NTBSTRINGBIND	DBCHAR	なし	¥0
VARYCHARBIND	DBVARYCHAR	なし	なし
BOUNDARYBIND	DBCHAR	なし	¥0
SENSITIVITYBIND	DBCHAR	なし	¥0

上記の表の「¥0」は、NULL ターミネータ文字を示しています。

整数または浮動小数点のデータを文字バインド型に変換するときにオーバフローが発生すると、結果の値の最初の文字は、エラーを示すアスタリスク (「*」) となります。

バイナリ・データを保管するには、2 とおりの方法があります。

vartype	プログラムの型	埋め込み文字
BINARYBIND	DBBINARY	NULL
VARYBINBIND	DBVARBINARY	なし

整数データのカラムの合計や平均値を算出するとき、サーバは、カラムのサイズに関係なく常に 4 バイトの整数を返します。したがって、このような計算の結果を格納する変数は DBINT として宣言し、バインドの *vartype* には INTBIND を指定してください。

varlen

プログラム変数の長さ (バイト単位)。

vartype の値が MONEYBIND や FLT8BIND などの固定長の型を表すものである場合は、この長さは無視されます。

文字型とバイナリ型の場合は、*varlen* には、NULL ターミネータなどの特別な終了バイトに必要な領域を含む、使用可能なコピー先バッファ領域の全長を指定してください。*varlen* が 0 の場合は、使用可能なすべてのバイトがプログラム変数にコピーされます (char と binary のサーバ・データの場合、使用可能なバイトの総数は、ブランクの埋め込み文字を含む、データベース・カラムの定義された長さと同じです。varchar と varbinary のデータの場合、使用可能なバイトの総数は、カラムに格納されている実際のデータと同じです)。したがって、プログラム変数が、結果を処理するのに十分な大きさであることが確実な場合は、*varlen* を 0 に設定してもかまいません。

varaddr

データのコピー先であるプログラム変数のアドレスです。

typeinfo

decimal または numeric のデータの精度と位取りに関する情報が格納されている DBTYPEINFO 構造体へのポインタです。アプリケーションでは、dbaltbind_ps を呼び出してカラムを DBDECIMAL または DBNUMERIC の変数にバインドする前に、DBTYPEINFO 構造体に精度と位取りの値を設定します。

typeinfo が NULL の場合

- 結果カラムのデータ型が numeric または decimal の場合、dbaltbind_ps は結果カラムから精度と位取りの値を取得します。
- 結果カラムのデータ型が numeric または decimal でない場合、dbaltbind_ps はデフォルト値である精度 18 と位取り 0 を使用します。

vartype が DECIMALBIND または NUMERICBIND ではない場合、*typeinfo* は無視されます。

DBTYPEINFO 構造体は、次のように定義されています。

```
typedef struct typeinfo {
     DBINT    precision;
     DBINT    scale;
} DBTYPEINFO;
```

precision の有効な値は、1 から 77 までです。*scale* の有効な値は、0 から 77 までです。*scale* の値は *precision* 以下でなければなりません。

戻り値

SUCCEED または FAIL。

カラム番号が正しくない場合、*vartype* で指定されたデータ変換が正しくない場合、または *varaddr* が NULL の場合、dbaltbind_ps は FAIL を返します。

使用法

- dbaltbind_ps は、numeric データ型と decimal データ型の精度と位取りをサポートする点を除いて dbaltbind と同じです。dbaltbind はこれらをサポートしていません。dbaltbind を呼び出すことは、*typeinfo* に NULL を指定して dbaltbind_ps を呼び出すこととまったく同じです。

- dbaltbind_ps は、サーバから返された計算カラム・データをプログラム変数にコピーするように DB-Library に指示します (計算カラムは、Transact-SQL select 文の compute 句から生成されます)。計算データが含まれている新しいローが dbnextrow または dbgetrow を介して読み込まれると、その計算ロー内の指定された *column* のデータが、アドレス *varaddr* のプログラム変数にコピーされます。コピーする計算カラムごとに dbaltbind_ps を呼び出す必要があります。すべての計算カラムをプログラム変数にバインドする必要はありません。
- サーバは 2 つのタイプのローを返します。select 文の select リストで指定されたカラムのデータが含まれる通常ローと、compute 句から生成される計算ローです。dbaltbind_ps は計算ローのデータをバインドします。通常ローのデータをバインドするには、dbbind_ps を使用してください。
- dbaltbind_ps 呼び出しは、dbresults 呼び出しの後で、dbnextrow の最初の呼び出しの前に行ってください。
- dbaltbind_ps はデータをプログラム変数にコピーするため、多少のオーバヘッドが発生します。このコピー作業を避けるには、返されたデータに直接アクセスする dbadata ルーチンを使用してください。
- 1 つの結果カラムをバインドできるのは 1 つのプログラム変数だけです。1 つの結果カラムを複数のプログラム変数にバインドした場合は、最後のバインドだけが有効になります。
- サーバから NULL 値が返されることがあるので、DB-Library ではデータ型ごとに 1 つのデフォルト値が用意されており、NULL 値をバインドすると、自動的にこのデフォルト値が代入されます。dbsetnull 関数を使用することにより、ユーザは、独自の null 代入値を明示的に設定できます。(デフォルト代入値のリストについては、dbsetnull 関数のリファレンス・ページを参照してください)。

参照

dbaltbind、dbanullbind、dbbind、dbbind_ps、dbconvert、dbconvert_ps、dbdata、dbnullbind、dbsetnull、dbsetversion、dbwillconvert、「データ型」(443 ページ)

dbaltcolid

説明

計算カラムのカラム ID を返します。

構文

int dbaltcolid(dbproc, computeid, column)

```
DBPROCESS    *dbproc;
int          computeid;
int          column;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid

対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

column

対象となる計算カラムの番号です。select リストの最初のカラムは 1 です。

戻り値

計算カラムの select リスト ID です。select リストの最初のカラムは 1 です。*computeid* または *column* 値が無効の場合、dbaltcolid は -1 を返します。

使用法

- このルーチンは、計算カラムの select リスト ID を返します。たとえば、SQL 文が次のとおりであるとします。

  ```
  select dept, name from employee
  order by dept, name
  compute count(name) by dept
  ```

  この場合、dbaltcolid(*dbproc*, 1, 1) を呼び出すと 2 が返されます。「name」は select リストの 2 番目のカラムであるからです。

参照

dbadata、dbadlen、dbaltlen、dbgetrow、dbnextrow、dbnumalts、dbprtype

dbaltlen

説明

特定の計算カラムのデータの最大長を返します。

構文

DBINT dbaltlen(dbproc, computeid, column)

```
DBPROCESS      *dbproc;
int            computeid;
int            column;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid

対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

column

対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定の計算カラムのデータのバイト単位の最大長です。該当するカラムがない場合や compute 句がない場合は、dbaltlen は -1 を返します。

使用法

このルーチンは、計算ロー内のカラムの最大長を返します。可変長データの場合、これは必ずしもデータの実際の長さではなく、最大長です。実際のデータ長を調べるには、dbadlen を使用してください。

たとえば、SQL 文が次のとおりであるとします。

```
select dept, name from employee
order by dept, name
compute count(name) by dept
```

dbaltlen(*dbproc*, 1, 1) を呼び出すと 4 が返されます。count は SYBINT4 型で、長さは 4 バイトであるためです。

参照

dbadata、dbadlen、dbalttype、dbgetrow、dbnextrow、dbnumalts

dbaltop

説明

特定の計算カラムの集合演算子のタイプを返します。

構文

int dbaltop(dbproc, computeid, column)

```
DBPROCESS    *dbproc;
int          computeid;
int          column;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid

対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

column

対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

計算カラムの集合演算子のタイプを表すトークン値です。エラーの場合、dbaltop は -1 を返します。

使用法

- このルーチンは、計算ロー内の特定カラムの集合演算子のタイプを返します。たとえば、SQL 文が次のとおりであるとします。

  ```
  select dept, name from employee
  order by dept, name
  compute count(name) by dept
  ```

  この場合、dbaltop(*dbproc*, 1, 1) を呼び出すと、count のトークン値が返されます。最初の compute 句の最初の集合演算子は count であるからです。

- dbprtype を使用すると、トークン値を、読み込み可能なトークン文字列に変換できます。すべてのトークン値とそれに相当するトークン文字列のリストについては、dbprtype のリファレンス・ページを参照してください。

参照

dbadata、dbadlen、dbaltlen、dbnextrow、dbnumalts、dbprtype

dbalttype

説明

計算カラムのデータ型を返します。

構文

int dbalttype(dbproc, computeid, column)

```
DBPROCESS    *dbproc;
int          computeid;
int          column;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
　対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

column
　対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定の計算カラムのデータ型を表すトークン値です。

このルーチンから返されるトークン値は、カラムのサーバ・データ型と正確に対応しない場合もあります。

- SYBVARCHAR は、SYBCHAR として返されます。
- SYBVARBINARY は、SYBBINARY として返されます。
- SYBDATETIMN は、SYBDATETIME として返されます。
- SYBMONEYN は、SYBMONEY として返されます。
- SYBFLTN は、SYBFLT8 として返されます。
- SYBINTN は、実際の SYBINTN のデータ型に応じて、SYBINT1、SYBINT2、SYBINT4 として返されます。

computeid または *column* の値が正しくない場合、dbalttype は -1 を返します。

使用法

- このルーチンは、計算カラムのデータ型を返します。サーバ・データ型のリストについては、「データ型」(443 ページ) を参照してください。

- dbalttype は、実際にはデータ型を整数で表したトークン値 (SYBCHAR、SYBFLT8 など) を返します。トークン値を読み込み可能なトークン文字列に変換するには、dbprtype を使用してください。すべてのトークン値とそれに相当するトークン文字列のリストについては、dbprtype のリファレンス・ページを参照してください。
- たとえば、SQL 文が次のとおりであるとします。

```
select dept, name from employee
order by dept, name
compute count(name) by dept
```

dbalttype(*dbproc*, 1, 1) の呼び出しはトークン値 SYBINT4 を返します。これは、カウントが SYBINT4 型であるためです。dbprtype は SYBINT4 を読み取り可能なトークン文字列「int」に変換します。

参照 dbadata、dbadlen、dbaltlen、dbnextrow、dbnumalts、dbprtype、「データ型」(443 ページ)

dbaltutype

説明 計算カラムのユーザ定義データ型を返します。

構文

```
DBINT dbaltutype(dbproc, computeid, column)

DBPROCESS    *dbproc;
int          computeid;
int          column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は、1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

ルーチンが正常に終了すると、指定の計算カラムのユーザ定義データ型が返されます。エラーの場合には、負の整数が返されます。

使用法

- dbaltutype は、計算カラムのユーザ定義データ型を返します。
- ユーザ定義データ型をサーバ・データベースや Server-Library プログラムに追加する方法については、『ASE リファレンス・マニュアル』または『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。
- DB-Library のデータ型 DBINT とユーザ定義データ型はどちらも長さが 32 ビットなので、dbaltutype は DBINT データ型として定義されています。

参照

dbalttype、dbcolutype

dbanullbind

説明

インジケータ変数を計算ロー・カラムに関連付けます。

構文

```
RETCODE dbanullbind(dbproc, computeid, column,
indicator)

DBPROCESS    *dbproc;
int          computeid;
int          column;
DBINT        *indicator;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
　対象となる計算ローです。1 つの select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は、1 です。

column
　インジケータ変数を関連付けるカラムの番号です。

indicator
インジケータ変数を指すポインタです。

注意 *indicator* は、インジケータ変数を指すポインタです。設定されるのは、変数自体です。

戻り値

SUCCEED または FAIL。

computeid または *column* のどちらかが正しくない場合、dbanullbind は FAIL を返します。

使用法

- dbanullbind は、計算ロー・カラムにインジケータ変数を関連付けます。インジケータ変数は、特定の計算ロー・カラムが正常に変換されてプログラム変数にコピーされたかどうか、または計算ロー・カラムが NULL であるかどうかを示します。
- インジケータ変数は、計算ローが dbnextrow によって処理されるときに設定されます。設定される値は次のとおりです。
  - カラムが NULL の場合は *-1* です。
  - カラムが dbaltbind によってプログラム変数にバインドされているけれども、そのバインドでデータ変換が指定されていないときに、カラムのデータを格納するにはプログラム変数が小さすぎてデータがトランケートされた場合は、カラムのデータのバイト単位の全長です。
  - カラムがバインドされて、プログラム変数に正常にコピーされた場合は *0* です。

注意 文字列がトランケートされたことの検出は、CHARBIND と VARYCHARBIND のみを対象として実装されています。

参照

dbadata、dbadlen、dbaltbind、dbnextrow、dbnullbind

dbbind

説明

通常の結果カラムをプログラム変数にバインドします。

構文

```
RETCODE dbbind(dbproc, column, vartype, varlen,
        varaddr)
```

```
DBPROCESS    *dbproc;
int          column;
int          vartype;
DBINT        varlen;
BYTE         *varaddr;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column

プログラム変数にコピーされるロー・データのカラム番号です。最初のカラムの番号は 1 です。

vartype

バインドのデータ型を表します。このデータ型は、DBPROCESS からのデータのコピーを受け取るプログラム変数のデータ型と対応していなければなりません。次の表に、*vartype* の値とプログラム変数の型の対応を示します。

dbbind は、さまざまな型変換をサポートするため、*vartype* は SQL クエリから返される型と異なっている可能性があります。たとえば、SYBMONEY 型の結果を、FLT8BIND を介して DBFLT8 型のプログラム変数にバインドできます。このとき、適切なデータ変換が自動的に行われます。DB-Library によって行われるデータ変換のリストについては、dbwillconvert のリファレンス・ページを参照してください。

注意　dbbind ルーチンには、numeric データ型と decimal データ型に対して精度と位取りを明示的に指定する機能はありません。dbbind で numeric または decimal のデータを扱うとき、numeric または decimal カラムへのバインドの場合はバインド元データの精度と位取りが使用され、それ以外の場合はデフォルトの精度 18 と位取り 0 が使用されます。精度と位取りの値を明示的に指定する場合は、dbbind_ps を使用します。dbbind を呼び出すことは、*typeinfo* の値に NULL を指定して dbbind_ps を呼び出すのと同じことです。

DB-Library によって使用される型定義のリストについては、「データ型」(443 ページ) を参照してください。

表 2-3 は、dbbind が認識する有効な *vartype* の値と、各 vartype に対応するサーバ・データ型とプログラム変数型のリストです。

表 2-3：dbbind のバインド型

vartype	プログラム変数の型	サーバ・データ型
CHARBIND	DBCHAR	SYBCHAR または SYBTEXT
STRINGBIND	DBCHAR	SYBCHAR または SYBTEXT
NTBSTRINGBIND	DBCHAR	SYBCHAR または SYBTEXT
VARYCHARBIND	DBVARYCHAR	SYBCHAR または SYBTEXT
BINARYBIND	DBBINARY	SYBBINARY または SYBIMAGE
VARYBINBIND	DBVARYBIN	SYBBINARY または SYBIMAGE
TINYBIND	DBTINYINT	SYBINT1
SMALLBIND	DBSMALLINT	SYBINT2
INTBIND	DBINT	SYBINT4
FLT8BIND	DBFLT8	SYBFLT8
REALBIND	DBREAL	SYBREAL
NUMERICBIND	DBNUMERIC	SYBNUMERIC
DECIMALBIND	DBDECIMAL	SYBDECIMAL
BITBIND	DBBIT	SYBBIT
DATETIMEBIND	DBDATETIME	SYBDATETIME
SMALLDATETIMEBIND	DBDATETIME4	SYBDATETIME4
MONEYBIND	DBMONEY	SYBMONEY
SMALLMONEYBIND	DBMONEY4	SYBMONEY4
BOUNDARYBIND	DBCHAR	SYBBOUNDARY
SENSITIVITYBIND	DBCHAR	SYBSENSITIVITY

警告！ライブラリのバージョンが DBVERSION_100 以上に設定されていない場合は (設定するには dbsetversion を使用します)、*vartype* に BOUNDARYBIND、DECIMALBIND、NUMERICBIND、または SENSITIVITYBIND を使用するとエラーになります。

前述の表にサーバ・データ型が含まれているのは、参考のためです。指定する *vartype* は、特定のサーバ・データ型に対応しているものでなくてもかまいません。前述したように、dbbind によって、指定の *vartype* にサーバ・データが変換されるためです。

注意　サーバ・データ型の nchar と nvarchar は、内部的に char データ型と varchar データ型に変換されます。これらは、DB-Library のデータ型定数 SYBCHAR に対応しています。

次の表は、使用可能な文字データと text データの表現を示します。これらの違いは、データがブランク埋め込みであるか、NULL 文字で終了するかによって生じます。*varlen* が 0 の場合は、埋め込みは行われません。「¥0」は、NULL ターミネータ文字を示しています。

vartype	プログラムの型	埋め込み文字	ターミネータ
CHARBIND	DBCHAR	ブランク	なし
STRINGBIND	DBCHAR	ブランク	¥0
NTBSTRINGBIND	DBCHAR	なし	¥0
VARYCHARBIND	DBVARYCHAR	なし	なし
BOUNDARYBIND	DBCHAR	なし	¥0
SENSITIVITYBIND	DBCHAR	なし	¥0

整数または浮動小数点のデータを文字／テキスト・バインド型に変換するときにオーバフローが発生すると、結果の値の最初の文字は、エラーを示すアスタリスク (「*」) となります。

binary および image のデータを保管するには、2 とおりの方法があります。

vartype	プログラムの型	埋め込み文字
BINARYBIND	DBBINARY	NULL
VARYBINBIND	DBVARBINARY	なし

varlen

プログラム変数の長さ (バイト単位)。

vartype の値が MONEYBIND や FLT8BIND などの固定長の型を表すものである場合は、この長さは無視されます。

char、text、binary、image データ型の場合、*varlen* には、NULL ターミネータなどの特別な終了バイトに必要な領域を含む、使用可能なコピー先バッファ領域の全長を指定してください。*varlen* が 0 の場合は、使用可能なすべてのバイトがプログラム変数にコピーされます (char と binary のサーバ・データの場合、使用可能なバイトの総数は、ブランクの埋め込み文字を含む、データベース・カラムの定義された長さと同じです。varchar、varbinary、text、および image のデータの場合、使用可能なバイトの総数は、カラムに格納されている実際のデータと同じです)。したがって、プログラム変数が、結果を処理するのに十分な大きさであることが確実な場合は、*varlen* を 0 に設定してもかまいません。

varlen が 0 の場合、埋め込みは行われません。

DB-Library は、データの変換の結果オーバフローが発生したことを示すメッセージを発行する場合があります。これは、サーバ・データに対して指定された *varlen* が小さすぎる場合に発生します。

varaddr
データのコピー先であるプログラム変数のアドレスです。

戻り値

SUCCEED または FAIL。

カラム番号が正しくない場合、*vartype* で指定されたデータ変換が有効でない場合、*varaddr* が NULL の場合、dbbind は FAIL を返します。

使用法

- サーバからのデータは、一度に 1 ローずつ返されます。このルーチンは、通常カラム (select 文の select リストに指定) のデータをプログラム変数にコピーするように DB-Library に指示します。通常データ (計算データではない) が含まれる新しいローが dbnextrow または dbgetrow を介して読み込まれると、そのロー内の指定された *column* のデータが、アドレス *varaddr* のプログラム変数にコピーされます。コピーする通常カラムごとに dbbind を呼び出す必要があります。すべてのカラムをプログラム変数にバインドする必要はありません。
- サーバは 2 つのタイプのローを返します。通常ローと、select 文の compute 句から生成される計算ローです。dbbind は、通常ローのデータをバインドします。計算ローのデータをバインドするには、dbaltbind を使用してください。
- dbbind 呼び出しは、dbresults 呼び出しの後で、dbnextrow の最初の呼び出しの前に行ってください。
- 標準的な呼び出しのシーケンスを次に示します。

```
DBINT     xvariable;
DBCHAR    yvariable[10];

/* read the query into the command buffer */
dbcmd(dbproc, "select x = 100, y = 'hello');

/* send the query to Adaptive Server Enterprise */
dbsqlexec(dbproc);

/* get ready to process the query results */
dbresults(dbproc);

/* bind column data to program variables */
dbbind(dbproc, 1, INTBIND, (DBINT) 0,
    (BYTE *) &xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
    yvariable);
```

```
/* now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    C-code to print or process row data
}
```

- dbbind はデータをプログラム変数にコピーするため、多少のオーバヘッドが発生します。このコピー作業を避けるには、返されたデータに直接アクセスする dbdata ルーチンを使用してください。
- 1 つの結果カラムをバインドできるのは 1 つのプログラム変数だけです。1 つの結果カラムを複数のプログラム変数にバインドした場合は、最後のバインドだけが有効になります。
- サーバから NULL 値が返されることがあるので、DB-Library ではデータ型ごとに 1 つのデフォルト値が用意されており、NULL 値をバインドすると、自動的にこのデフォルト値が代入されます。dbsetnull 関数を使用することにより、ユーザは、独自の null 代入値を明示的に設定できます。(デフォルト代入値のリストについては、dbsetnull 関数のリファレンス・ページを参照してください)。

参照　dbaltbind、dbaltbind_ps、dbnullbind、dbbind_ps、dbconvert、dbconvert_ps、dbdata、dbnullbind、dbsetnull、dbsetversion、dbwillconvert、「データ型」(443 ページ)

dbbind_ps

説明　通常の結果カラムをプログラム変数にバインドします。numeric データ型と decimal データ型の精度と位取りをサポートします。

構文

```
RETCODE dbbind_ps(dbproc, column, vartype, varlen,
        varaddr, typeinfo)

DBPROCESS    *dbproc;
int          column;
int          vartype;
DBINT        varlen;
BYTE         *varaddr;
DBTYPEINFO   *typeinfo;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column

プログラム変数にコピーされるロー・データのカラム番号です。最初のカラムの番号は 1 です。

vartype

バインドのデータ型を表します。このデータ型は、DBPROCESS からのデータのコピーを受け取るプログラム変数のデータ型と対応していなければなりません。次の表に、*vartype* の値とプログラム変数の型の対応を示します。

dbbind_ps は、さまざまな型変換をサポートするため、*vartype* は SQL クエリから返される型と異なっている可能性があります。たとえば、SYBMONEY 型の結果を、FLT8BIND を介して DBFLT8 型のプログラム変数にバインドできます。このとき、適切なデータ変換が自動的に行われます。DB-Library によって行われるデータ変換のリストについては、dbwillconvert のリファレンス・ページを参照してください。

DB-Library によって使用される型定義のリストについては、「データ型」(443 ページ) を参照してください。

表 2-4 は、dbbind_ps が認識する有効な *vartype* の値と、各 vartype に対応するサーバ・データ型とプログラム変数型のリストです。

表 2-4 : dbbind_ps のバインド型

vartype	プログラム変数の型	サーバ・タイプ
CHARBIND	DBCHAR	SYBCHAR または SYBTEXT
STRINGBIND	DBCHAR	SYBCHAR または SYBTEXT
NTBSTRINGBIND	DBCHAR	SYBCHAR または SYBTEXT
VARYCHARBIND	DBVARYCHAR	SYBCHAR または SYBTEXT
BINARYBIND	DBBINARY	SYBBINARY または SYBIMAGE
VARYBINBIND	DBVARYBIN	SYBBINARY または SYBIMAGE
TINYBIND	DBTINYINT	SYBINT1
SMALLBIND	DBSMALLINT	SYBINT2
INTBIND	DBINT	SYBINT4
FLT8BIND	DBFLT8	SYBFLT8
REALBIND	DBREAL	SYBREAL
NUMERICBIND	DBNUMERIC	SYBNUMERIC
DECIMALBIND	DBDECIMAL	SYBDECIMAL
BITBIND	DBBIT	SYBBIT
DATETIMEBIND	DBDATETIME	SYBDATETIME
SMALLDATETIMEBIND	DBDATETIME4	SYBDATETIME4
MONEYBIND	DBMONEY	SYBMONEY
SMALLMONEYBIND	DBMONEY4	SYBMONEY4
BOUNDARYBIND	DBCHAR	SYBBOUNDARY
SENSITIVITYBIND	DBCHAR	SYBSENSITIVITY

警告！ライブラリのバージョンが DBVERSION_100 以上に設定されていない場合は (設定するには dbsetversion を使用します)、*vartype* に BOUNDARYBIND、DECIMALBIND、NUMERICBIND、または SENSITIVITYBIND を使用するとエラーになります。*

前述の表にサーバ・データ型が含まれているのは、参考のためです。指定する *vartype* は、特定のサーバ型に対応しているものでなくてもかまいません。前述したように、dbbind_ps によって、指定の *vartype* にサーバ・データが変換されるためです。

注意　サーバ・データ型の nchar と nvarchar は、内部的に char データ型と varchar データ型に変換されます。これらは、DB-Library のデータ型定数 SYBCHAR に対応しています。

次の表は、使用可能な文字データと text データの表現を示します。これらの違いは、データがブランク埋め込みであるか、NULL 文字で終了するかによって生じます。*varlen* が 0 の場合は、埋め込みは行われません。「¥0」は、NULL ターミネータ文字を示しています。

vartype	プログラムの型	埋め込み文字	ターミネータ
CHARBIND	DBCHAR	ブランク	なし
STRINGBIND	DBCHAR	ブランク	¥0
NTBSTRINGBIND	DBCHAR	なし	¥0
VARYCHARBIND	DBVARYCHAR	なし	なし
BOUNDARYBIND	DBCHAR	なし	¥0
SENSITIVITYBIND	DBCHAR	なし	¥0

整数または浮動小数点のデータを文字／テキスト・バインド型に変換するときにオーバフローが発生すると、結果の値の最初の文字は、エラーを示すアスタリスク (「*」) となります。

binary データと image データを保管するには、2 とおりの方法があります。

vartype	プログラム変数タイプ	埋め込み文字
BINARYBIND	DBBINARY	NULL
VARYBINBIND	DBVARBINARY	なし

varlen

プログラム変数の長さ (バイト単位)。

vartype の値が MONEYBIND や FLT8BIND などの固定長の型を表すものである場合は、この長さは無視されます。

char、text、binary、image データ型の場合、*varlen* には、NULL ターミネータなどの特別な終了バイトに必要な領域を含む、使用可能なコピー先バッファ領域の全長を指定してください。*varlen* が 0 の場合は、使用可能なすべてのバイトがプログラム変数にコピーされます (char と binary のサーバ・データの場合、使用可能なバイトの総数は、ブランクの埋め込み文字を含む、データベース・カラムの定義された長さと同じです。varchar、varbinary、text、および image のデータの場合、使用可能なバイトの総数は、カラムに格納されている実際のデータと同じです)。したがって、プログラム変数が、結果を処理するのに十分な大きさであることが確実な場合は、*varlen* を 0 に設定してもかまいません。

注意 *varlen* が 0 の場合、埋め込みは行われません。

varaddr
データのコピー先であるプログラム変数のアドレスです。

typeinfo
decimal または numeric のデータの精度と位取りに関する情報が格納されている DBTYPEINFO 構造体へのポインタです。アプリケーションでは、dbbind_ps を呼び出してカラムを DBDECIMAL または DBNUMERIC の変数にバインドする前に、DBTYPEINFO 構造体に精度と位取りの値を設定します。

typeinfo が NULL の場合

- 結果カラムのデータ型が numeric または decimal の場合、dbbind_ps は結果カラムから精度と位取りの値を取得します。
- 結果カラムのデータ型が numeric または decimal でない場合、dbbind_ps はデフォルト値である精度 18 と位取り 0 を使用します。

vartype が DECIMALBIND または NUMERICBIND ではない場合、*typeinfo* は無視されます。

DBTYPEINFO 構造体は、次のように定義されています。

```
typedef struct typeinfo {
     DBINTprecision;
     DBINTscale;
} DBTYPEINFO;
```

precision の有効な値は、1 から 77 までです。*scale* の有効な値は、0 から 77 までです。*scale* の値は *precision* 以下でなければなりません。

戻り値

SUCCEED または FAIL。

カラム番号が正しくない場合、*vartype* で指定されたデータ変換が有効でない場合、*varaddr* が NULL の場合、dbbind_ps は FAIL を返します。

使用法

- dbbind_ps のパラメータは、DBNUMERIC または DBDECIMAL の変数の精度と位取りに関する情報を指定するパラメータ *typeinfo* が追加されている点を除いて、dbbind のパラメータとまったく同じです。
- dbbind_ps は、numeric データ型と decimal データ型の精度と位取りをサポートする点を除いて、dbbind と同じです。dbbind はこれらをサポートしていません。dbbind を呼び出すことは、*typeinfo* に NULL を指定して dbbind_ps を呼び出すこととまったく同じです。

- サーバからのデータは、一度に 1 ローずつ返されます。このルーチンは、通常カラム (select 文の select リストに指定) のデータをプログラム変数にコピーするように DB-Library に指示します。通常データ (計算データではない) が含まれる新しいローが dbnextrow または dbgetrow を介して読み込まれると、そのロー内の指定された *column* のデータが、アドレス *varaddr* のプログラム変数にコピーされます。コピーする通常カラムごとに dbbind または dbbind_ps を呼び出す必要があります。すべてのカラムをプログラム変数にバインドする必要はありません。
- サーバは 2 つのタイプのローを返します。通常ローと、select 文の compute 句から生成される計算ローです。通常ローのデータをバインドするには dbbind_ps を使用し、計算ローのデータをバインドするには dbaltbind_ps を使用します。
- dbbind_ps 呼び出しは、dbresults 呼び出しの後で、dbnextrow の最初の呼び出しの前に行ってください。
- dbbind_ps はデータをプログラム変数にコピーするため、多少のオーバヘッドが発生します。このコピー作業を避けるには、返されたデータに直接アクセスする dbdata ルーチンを使用してください。
- 1 つの結果カラムをバインドできるのは 1 つのプログラム変数だけです。1 つの結果カラムを複数のプログラム変数にバインドした場合は、最後のバインドだけが有効になります。
- サーバから NULL 値が返されることがあるので、DB-Library ではデータ型ごとに 1 つのデフォルト値が用意されており、NULL 値をバインドすると、自動的にこのデフォルト値が代入されます。dbsetnull 関数を使用することにより、ユーザは、独自の null 代入値を明示的に設定できます。デフォルト代入値のリストについては、dbsetnull 関数のリファレンス・ページを参照してください。

参照

dbaltbind、dbaltbind_ps、dbanullbind、dbbind、dbconvert、dbconvert_ps、dbdata、dbnullbind、dbsetnull、dbsetversion、dbwillconvert、「データ型」(443 ページ)

dbbufsize

説明

DBPROCESS ロー・バッファのサイズを返します。

構文

```
int dbbufsize(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

DBPROCESS ロー・バッファのサイズをロー単位で表す整数です。

dbproc が NULL の場合、またはロー・バッファリングが使用可能でない場合、dbbufsize は 0 を返します。

使用法

- dbbufsize は、DBPROCESS ロー・バッファのサイズを返します。
- ロー・バッファリングを利用すると、アプリケーションで、指定した数のサーバ結果ローをプログラム・メモリ内に保持することができます。ロー・バッファリングが行われるようにするには、dbsetopt (*dbproc*、DBBUFFER、*n*) を呼び出します。*n* は、バッファ内に保持するローの数です。結果ローをバッファリングしている場合は、dbgetrow を使用することによって、ランダムな順序でローにアクセスできます。ロー・バッファリングの利点と影響については、dbgetrow のリファレンス・ページを参照してください。

参照

dbclrbuf、dbgetrow、dbsetopt、「オプション」(436 ページ)

dbbylist

説明

計算ローの bylist を返します。

構文

```
BYTE *dbbylist(dbproc, computeid, size)

DBPROCESS    *dbproc;
int          computeid;
int          *size;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

size
整数へのポインタです。この整数には、dbbylist によって bylist 内の要素の数が設定されます。

戻り値

指定の計算の bylist を構成するカラムの番号が格納されているバイト配列を指すポインタです。BYTEs の配列は DBPROCESS の一部なので、解放しないでください。*computeid* が範囲外の場合は、NULL が返されます。

カラム番号からカラム名を導出するには、dbcolname を呼び出してください。

配列のサイズは、*size* パラメータに返されます。*size* 0 は、この特定の計算に bylist がないか、*computeid* が範囲外であることを示しています。

使用法

- dbbylist は、計算ローの bylist を返します (select 文の compute 句には、キーワード by に続けてカラムのリストを指定できます。このリストは、「bylist」として知られ、指定されたカラムの値の変更に基づいて、結果をサブグループに分類します。compute 句によるローの集計が各サブグループに対して行われ、サブグループごとに計算ローが生成されます)。
- アプリケーションでこのルーチンを呼び出すには、dbresults の戻り値が SUCCEED でなければなりません。
- 次のコマンドが実行されたとします。

```
select dept, name, year, sales from employee
order by dept, name, year
compute count(name) by dept, name
```

この場合、bylist に 2 つの項目があるので、dbbylist (*dbproc*, 1, &*size*) を呼び出すと、*size* は 2 に設定されます。このルーチンは、2 つの BYTE から成る配列を指すポインタを返します。これらの BYTE の値は 1 と 2 で、この bylist が select リストのカラム 1 と 2 で構成されていることを示します。

参照

dbadata、dbadlen、dbaltlen、dbalttype、dbcolname、dbgetrow、dbnextrow

dbcancel

説明

現在のコマンド・バッチをキャンセルします。

構文

```
RETCODE dbcancel(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

最も一般的なエラー理由は dead DBPROCESS またはネットワーク・エラーです。サーバが dead の場合、dbcancel も FAIL を返します。

使用法

- このルーチンは、サーバ上での現在のコマンド・バッチの実行をキャンセルし、保留中の結果をすべてフラッシュします。アプリケーションでは、dbsqlexec、dbsqlsend、dbsqlok、dbresults、または dbnextrow を呼び出した後に dbcancel ルーチンを呼び出すことができます。このルーチンは、サーバにアテンション・パケットを送信して、コマンド・バッチの実行を中止させます。保留中の結果はすべて読み込まれ、廃棄されます。
- dbcancel は、現在のコマンド・バッチの「すべてのコマンド」をキャンセルします。現在のコマンドの結果だけをキャンセルするには、このルーチンではなく、dbcanquery を呼び出してください。
- 一部のアプリケーションでは、DB-Library がネットワークからの読み込みを行っているときに、長時間実行しているクエリをキャンセルする機能が必要になることがあります。この場合は、次のいずれかの方法を使用します。
  - dbsettime を使用してサーバの読み込み時間の制限を設定します。また、SYBETIME エラーに応答するために、エラー・ハンドラ関数に特別な場合の処理を追加します。詳細については、dberrhandle と dbsettime のリファレンス・ページを参照してください。
  - dbsetinterrupt を使用して、ユーザ独自の割り込み処理をインストールします。詳細については、dbsetinterrupt のリファレンス・ページを参照してください。

- dbsetinterrupt を使用して独自の割り込みハンドラを設定した場合に、その割り込みハンドラ内で dbcancel を呼び出すことはできません。これを行うと、サーバから DB-Library への出力は同期が取れなくなります。割り込みハンドラからキャンセルする方法については、dbsetinterrupt のリファレンス・ページを参照してください。

参照

dbcanquery、dbnextrow、dbresults、dbsetinterrupt、dbsqlexec、dbsqlok、dbsqlsend

dbcanquery

説明

最後に実行されたクエリからの保留中のローをキャンセルします。

構文

```
RETCODE dbcanquery(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

失敗の主な原因は、DBPROCESS のステータスが dead になっていること、またはネットワーク・エラーです。

使用法

- このルーチンは、最後に実行された SQL クエリによって生成されたローのうち、まだ読み込まれていないローをすべて廃棄する効率的な方法です。dbcanquery の呼び出しは、NO_MORE_ROWS が返されるまで dbnextrow を呼び出すことと同じですが、dbcanquery は、メモリ割り付けを行わず、ユーザ・データへのバインドも実行しないため、より高速です。
- dbsetinterrupt を使用してユーザ独自の割り込みハンドラを設定している場合に、その割り込みハンドラ内で dbcanquery を呼び出すことはできません。これを行うと、サーバから DB-Library への出力は同期が取れなくなります。現在のクエリからの読み込まれていないローをすべて無視するには、割り込みハンドラでフラグを設定し、このフラグを、次回の dbnextrow の呼び出しの前に確認するようにします。

- アプリケーションで dbcanquery を呼び出すには、dbresults で SUCCEED が返されている必要があります。
- 現在のコマンド・バッチのすべてのコマンドのすべての結果を無視するには、代わりに dbcancel を使用してください。

参照　dbcancel、dbnextrow、dbresults、dbsetinterrupt、dbsqlexec

dbchange

説明　コマンド・バッチが現在のデータベースを変更したかどうかを調べます。

構文

```
char *dbchange(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　新しいデータベースがある場合は、そのデータベースの NULL で終了する名前へのポインタです。データベースが変更されなかった場合は、NULL が返されます。

使用法

- dbchange は、現在のデータベースの変更をプログラムに知らせます。このルーチンは、Transact-SQL の use コマンドの実行を検出して処理を行います。
- use コマンドはコマンド・バッチ内の任意の場所で使用できますが、データベースが実際に変更されるのはバッチの終了時です。したがって、dbchange によって調べることができるのは、現在のコマンド・バッチが後続のコマンド・バッチのためにデータベースを変更したかどうかだけです。
- データベースが変更されたかどうかを調べるために dbchange がモニタする内部的な DBPROCESS のフラグは、プログラムが dbsqlexec または dbsqlsend を呼び出して新しいコマンド・バッチを実行するときにクリアされます。したがって、データベースの変更を追跡する最も簡単な方法は、各コマンド・バッチの最後に dbresults から NO_MORE_RESULTS が返された時点で dbchange を呼び出すという方法です。

- もう 1 つの方法として、dbname を呼び出して、現在のデータベースの名前を取得できます。

参照 dbname、dbresults、dbsqlexec、dbsqlsend、dbuse

dbcharsetconv

説明 サーバが文字セット変換を実行しているかどうかを示します。

構文

```
DBBOOL dbcharsetconv(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 「サーバが文字セット変換を実行している場合は「TRUE」、実行していない場合は「FALSE」を返します。

使用法

- クライアントとサーバが同じ文字セットを使用している場合は、サーバは変換を実行していません。この場合、dbcharsetconv は「FALSE」を返します。
- クライアントが、自分が使用している文字セットの名前を取得するには、dbgetcharset を呼び出します。
- クライアントがサーバの文字セットの名前を取得するには、dbservcharset を呼び出します。

参照 dbservcharset、dbgetcharset、DBSETLCHARSET

dbclose

説明 1 つの DBPROCESS 構造体をクローズし、割り付けを解除します。

構文

```
void dbclose(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

なし。

使用法

- dbclose は dbopen の逆の処理を行います。このルーチンは、1 つの DBPROCESS 構造体に関連付けられているすべてのアクティビティを終了し、その領域割り付けを解除します。また、対応するネットワーク接続もクローズします。
- オープンしている DBPROCESS 構造体をすべてクローズするには、dbexit を使用してください。
- dbclose は、LOGINREC に関連付けられている領域の割り付けは解除しません。LOGINREC の領域割り付けを解除するには、アプリケーションで dbloginfree を呼び出します。
- dbclose を呼び出すときに指定した引数が、dbopen で返されたものでない場合は、エラーが発生します。

参照

dbexit、dbopen

dbclrbuf

説明

ロー・バッファからローを削除します。

構文

```
void dbclrbuf(dbproc, n)

DBPROCESS    *dbproc;
DBINT        n;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

n
ロー・バッファから削除するローの数です。*n* がバッファ内のローの数以上の場合は、最も新しいローを除いたすべてのローが削除されます。*n* が 1 より小さい場合、この関数呼び出しは無視されます。

戻り値

なし。

使用法

- DB-Library には、アプリケーション・プログラムがロー・バッファリングを行うための機能があります。ロー・バッファリングが行われるようにするには、dbsetopt(*dbproc*, DBBUFFER, *n*) を呼び出します。*n* は、DB-Library がバッファに保持するローの数です。バッファリングがオンになっていれば、dbgetrow を使用して、サーバから読み込んだローをランダムに参照できます。ロー・バッファリングの利点と影響については、dbgetrow のリファレンス・ページを参照してください。
- ロー・バッファが満杯になる原因は 2 つあります。バッファに保持する数として指定された *n* を超える数のローがサーバから返された場合と、要求したローを保存するのに十分な領域を割り付けることができなかった場合です。ロー・バッファが満杯になると、dbnextrow は BUF_FULL を返し、サーバからの次のローの読み込みを拒否します。ロー・バッファがいったん満杯になると、dbclrbuf を呼び出すことによって少なくとも 1 つのローが解放されるまで、後続の dbnextrow 呼び出しは BUF_FULL を返します。dbclrbuf は、バッファ内の最も古いローを最初に解放します。
- 結果ローをバッファからクリアすると、プログラムでそのローを使用することはできなくなります。
- ロー・バッファリングの例については、オンラインのサンプル・プログラムの *example4.c* を参照してください。

参照

dbgetrow、dbnextrow、dbsetopt、「オプション」(436 ページ)

dbclropt

説明

dbsetopt によって設定されたオプションをクリアします。

構文

RETCODE dbclropt(dbproc, option, param)

```
DBPROCESS    *dbproc;
int          option;
char*        param;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。*dbproc* が NULL の場合は、すべてのアクティブな DBPROCESS 構造体について、オプションがクリアされます。

option

オフにするオプションです。オプションのリストについては、「オプション」(436 ページ) を参照してください。

param

オプションにはパラメータを指定できるものがあります。たとえば、DBOFFSET オプションは SQL 構成体をパラメータとして受け取り、そのオフセットを返します。パラメータを指定できるオプションのリストについては、「オプション」(436 ページ) を参照してください。パラメータを指定しないオプションの場合は、*param* を NULL にしてください。

クリアするオプションが、パラメータを受け取るけれども、インスタンスが 1 つしか存在しないものである場合は、dbclropt は *param* 引数を無視します。たとえば、DBBUFFER オプションをクリアするときに、dbclropt は *param* の値を無視します。これは、ロー・バッファリングの設定が一度に 1 つだけであるためです。一方、DBOFFSET オプションには複数の設定値が可能で、それぞれに異なるパラメータがあります。select 文に対するオフセットと order by 句に対するオフセットを調べる場合のように、DBOFFSET が 2 回設定されていることもあります。その場合は、dbclropt で select オフセットと order by オフセットのどちらをクリアするのかを判断するために、*param* 引数が必要です。

サーバ・オプションに不正なパラメータが指定されている場合に、このことが検出されるのは、次にコマンド・バッファがサーバに送信されるときです。その結果 dbsqlexec または dbsqlsend の呼び出しは失敗し、DB-Library は、ユーザがインストールしたメッセージ・ハンドラを呼び出します。DB-Library オプション (DBBUFFER または DBTEXTLIMIT) のいずれかに正しくないパラメータが指定されている場合には、dbclropt 呼び出し自体が失敗します。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、dbsetopt で設定されたサーバ・オプションと DB-Library オプションをクリアします。サーバ・オプションは SQL を介して直接設定したりクリアしたりできますが、アプリケーションでは、オプションの設定とクリアには dbsetopt と dbclropt を使用してください。このようにすれば、サーバ・オプションと DB-Library オプションを設定するインタフェースを統一できます。また、アプリケーションで dbisopt 関数を使用してオプションのステータスを調べることもできます。
- dbclropt を実行しても、オプションはすぐにはクリアされません。オプションは、次の dbsqlexec または dbsqlsend の呼び出しによってコマンド・バッファがサーバに送信されるときにクリアされます。
- オプションのリストについては、「オプション」(436 ページ) を参照してください。

参照

dbisopt、dbsetopt、「オプション」(436 ページ)

dbcmd

説明

テキストを DBPROCESS コマンド・バッファに追加する。

構文

```
RETCODE dbcmd(dbproc, cmdstring)

DBPROCESS     *dbproc;
char          *cmdstring;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

cmdstring

dbcmd がコマンド・バッファにコピーする、NULL で終了する文字列です。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、DBPROCESS 構造体内の Transact-SQL コマンド・バッファにテキストを追加します。テキストは既存のコマンド・バッファに追加されます。バッファがサーバに送信された後を除いて、バッファの現在の内容が削除または上書きされることはありません (「コマンド・バッファのクリア」(98 ページ) を参照)。1 つのコマンド・バッファに複数のコマンドが含まれることがありますが、このようにするのがコマンド・バッファの効率的な使い方です。
- dbfcmd は、関連関数です。dbfcmd は、*cmdstring* を、追加の引数と一緒に sprintf に渡されるフォーマット文字列と解釈します。1 つのアプリケーション内に、dbcmd と dbfcmd の呼び出しが混在していてもかまいません。

dbcmd の連続呼び出し

- アプリケーションで、dbcmd を繰り返し呼び出すことができます。連続して呼び出すと、コマンド文字列はそのまま連結されます。文字列の終わりと次の文字列の始まりとの間にブランクが必要な場合に、そのブランクを挿入するのは、アプリケーション側で行います。
- 次に、dbcmd を使用して複数行の SQL コマンドを構築する例を示します。

```
DBPROCESS    *dbproc;

dbcmd(dbproc, "select name from sysobjects");
dbcmd(dbproc, " where id < 5");
dbcmd(dbproc, " and type='S'");
```

  2 番目と 3 番目のコマンド文字列の前に空白が必要な点に注意してください。
- アプリケーションは、いつでも dbgetchar、dbstrlen、dbstrcpy を呼び出してコマンド・バッファの内容にアクセスできます。
- dbcmd と dbfcmd の呼び出しで作成される DBPROCESS コマンド・バッファのサイズは、使用可能なメモリ容量以外の制約は受けません。

コマンド・バッファのクリア

- dbsqlexec または dbsqlsend を呼び出した後の dbcmd または dbfcmd の最初の呼び出し時に、新しいテキストが入力される前にコマンド・バッファが自動的にクリアされます。これを行わない場合は、DBNOAUTOFREE オプションを設定してください。
  DBNOAUTOFREE が設定されているときは、dbfreebuf を明示的に呼び出した場合にのみコマンド・バッファがクリアされます。

参照

dbfcmd、dbfreebuf、dbgetchar、dbstrcpy、dbstrlen、「オプション」(436 ページ)

DBCMDROW

説明

現在のコマンドがローを返せるかどうかを調べます。

構文

```
RETCODE DBCMDROW(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL で、コマンドがローを返すことができるかどうかを示します。

使用法

- DBCMDROW は、dbresults によって現在処理されているコマンドがローを返せるコマンド (つまり、Transact-SQL select 文または select が含まれているストアド・プロシージャを実行する execute) であるかどうかを調べます。アプリケーションは、dbresults が SUCCEED を返した後に、このルーチンを呼び出すことができます。
- DBCMDROW マクロが SUCCEED を返しても、条件を満たすローがなければ、コマンドはローを返しません。実際にローが返されるかどうかを調べるには、DBROWS を使用してください。

参照

dbnextrow、dbresults、DBROWS、DBROWTYPE

dbcolbrowse

説明

通常の結果カラムのソースが、DB-Library のブラウズ・モード機能を介して更新可能かどうかを調べます。

構文

DBBOOL dbcolbrowse(dbproc, colnum)

```
DBPROCESS    *dbproc;
int          colnum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

colnum

対象となる結果カラムの番号です。カラム番号は 1 から始まります。

戻り値

「TRUE」または「FALSE」。

使用法

- dbcolbrowse は、DB-Library のブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「第 1 章 DB-Library の概要」を参照してください。
- dbcolbrowse を使用すると、select リスト内の通常 (非計算) の結果カラムのソースであるデータベース・カラムが、DB-Library ブラウズ・モード機能を使用して更新可能かどうかを調べることができます。このルーチンは、アドホック・クエリを調べるときに便利です。クエリがプログラム内にハードコードされている場合は、dbcolbrowse は不要です。
- 更新可能なカラムとは、ブラウズ可能なテーブルから導出されたカラムです (テーブルには、ユニーク・インデックスとタイムスタンプ・カラムが必要です)。カラムは、SQL 式の結果として作成されるものであってはなりません。次の select リストを例に示します。

  ```
  select title, category=type,
  wholesale=(price * 0.6) ... for browse
  ```

  この場合、結果カラム 1 と 2 (「title」と「category」) は更新可能ですが、カラム 3 (「wholesale」) は、式の結果として作成されるので更新は不可能です。
- アプリケーションは、dbresults の後ならいつでも dbcolbrowse を呼び出すことができます。

- 結果カラムの名前からソース・カラムの名前を調べるには、dbcolsource を使用してください。
- オンラインのサンプル・プログラムの *example7.c* に、dbcolbrowse の呼び出しが含まれています。

参照

dbcolsource、dbqual、dbtabbrowse、dbtabcount、dbtabname、dbtabsource、dbtsnewlen、dbtsnewval、dbtsput

dbcollen

説明

通常の結果カラム内のデータの最大長を返します。

構文

DBINT dbcollen(dbproc, column)

```
DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定のカラムのデータのバイト単位の最大長です。カラム番号が範囲外の場合、dbcollen は -1 を返します。

使用法

- このルーチンは、通常 (非計算) 結果カラムのデータの最大長を返します。可変長データの場合、これは必ずしもデータの実際の長さではなく、データが取り得る最大長です。実際のデータ長を調べるには、dbdatlen を使用してください。
- dbcollen が返す値は、rtrim や ltrim などの Transact-SQL 文字列関数による影響を受けません。たとえば、カラム *au_lname* の最大長が 20 文字であり、*au_lname* の最初のロー・インスタンスが「Goodman」(13 個のスペースが埋め込まれている値) である場合は、Transact-SQL コマンドの select rtrim(au_lname) from authors によって返される文字列の長さが 5 文字でも、dbcollen は、*au_lname* の長さとして 20 を返します。
- 次のコードは、dbcollen の例です。

```
DBPROCESS     *dbproc;
 int           colnum;
 DBINT         column_length;

 /* Put the command into the command buffer */
 dbcmd(dbproc, "select name, id, type from
       sysobjects");

 /*
 ** Send the command to Adaptive Server Enterprise
and begin
 ** execution
 */
 dbsqlexec(dbproc);

 /* process the command results */
 dbresults(dbproc);

 /* examine the column lengths */
 for (colnum = 1; colnum < 4; colnum++)
 {
     column_length = dbcollen(dbproc, colnum);
     printf("column %d, length is %ld.¥n", colnum,
           column_length);
 }
```

参照　dbcolname、dbcoltype、dbdata、dbdatlen、dbnumcols

dbcolname

説明　通常の結果カラムの名前を返します。

構文　char *dbcolname(dbproc, column)

```
DBPROCESS    *dbproc;
int          column;
```

パラメータ　dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定のカラムの NULL で終了する名前を指す CHAR ポインタです。カラム番号が範囲外の場合、dbcolname は NULL を返します。

使用法

- このルーチンは、通常 (非計算) 結果カラムの NULL で終了する名前を指すポインタを返します。
- 次のコードは、dbcolname の例です。

```
DBPROCESS    *dbproc;

/* Put the command into the command buffer */
dbcmd(dbproc, "select name, id, type from
       sysobjects");

/*
** Send the command to Adaptive Server Enterprise
and begin
** execution
*/
dbsqlexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/* Examine the column names */
printf("first column name is %s¥n",
                dbcolname(dbproc, 1));
printf("second column name is %s¥n",
                dbcolname(dbproc, 2));
printf("third column name is %s¥n",
                dbcolname(dbproc, 3));
```

参照

dbcollen、dbcoltype、dbdata、dbdatlen、dbnumcols

dbcolsource

説明

指定された通常の結果カラムの導出元であるデータベース・カラムの名前を指すポインタを返します。

構文

char *dbcolsource(dbproc, colnum)

```
DBPROCESS    *dbproc;
int          colnum;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

colnum
　対象となる結果カラムの番号です。カラム番号は 1 から始まります。

戻り値

NULL で終了するカラム名を指すポインタです。このポインタが NULL となるのは、カラム番号が範囲外の場合、またはカラムが max(colname) などの SQL 式の結果である場合です。

使用法

- dbcolsource は、DB-Library のブラウズ・モード・ルーチンの 1 つです。これは、ブラウズ・モードの select (つまり、for browse キーワードを含む select) の結果に対してのみ使用可能です。ブラウズ・モードの詳細については、「第 1 章 DB-Library の概要」を参照してください。
- dbcolsource を利用すると、アドホック・クエリに基づいてデータベース・カラムを更新するのに必要な情報を得ることができます。select 文では、オプションとして、通常 (非計算) 結果カラムのヘッダ名を指定できます。

  ```
  select author = au_lname from authors for browse
  ```

  テーブルを更新するときは、ヘッダ名ではなくデータベース・カラム名 (この例では「author」ではなく「au_lname」) を使用する必要があります。dbcolsource ルーチンを使用すれば、元のデータベース・カラム名を取得できます。

  ```
  dbcolsource(dbproc, 1)
  ```

  この呼び出しは、文字列「au_lname」を指すポインタを返します。
- dbcolsource は、アドホック・クエリに便利です。クエリがプログラムにハードコードされている場合は、このルーチンは必要ありません。
- アプリケーションは、dbresults の後ならいつでも dbcolsource を呼び出すことができます。
- オンラインのサンプル・プログラムの *example7.c* に、dbcolsource の呼び出しが含まれています。

参照

dbcolbrowse、dbqual、dbtabbrowse、dbtabcount、dbtabname、dbtabsource、dbtsnewlen、dbtsnewval、dbtsput

dbcoltype

説明

通常の結果カラムのデータ型を返します。

構文

int dbcoltype(dbproc, column)

```
DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column

対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定のカラムのデータ型を表すトークン値です。

このルーチンから返されるトークン値は、カラムのサーバ・データ型と正確に対応しない場合もあります。

- SYBVARCHAR は、SYBCHAR として返されます。
- SYBVARBINARY は、SYBBINARY として返されます。
- SYBDATETIMN は、SYBDATETIME として返されます。
- SYBMONEYN は、SYBMONEY として返されます。
- SYBFLTN は、SYBFLT8 として返されます。
- SYBINTN は、実際の SYBINTN のデータ型に応じて、SYBINT1、SYBINT2、SYBINT4 として返されます。

カラム番号が範囲外の場合、dbcoltype は -1 を返します。

使用法

- このルーチンは、通常 (非計算) 結果カラムのデータ型を返します。サーバ・データ型のリストについては、「データ型」(443 ページ) を参照してください。
- dbcoltype は、実際にはデータ型を整数で表したトークン値 (SYBCHAR、SYBFLT8 など) を返します。トークン値を読み込み可能なトークン文字列に変換するには、dbprtype を使用してください。すべてのトークン値とそれに相当するトークン文字列のリストについては、dbprtype のリファレンス・ページを参照してください。
- カラムのデータ型が可変長であるかどうかを調べるには、dbvarylen を使用します。

- 次のコードは、dbcoltype の例です。

```
DBPROCESS    *dbproc;
int          colnum;
int          coltype;

/* Put the command into the command buffer */
dbcmd(dbproc, "select name, id, type from
     sysobjects");

/* Send the command to Adaptive Server Enterprise
and begin
** execution.
*/
dbsqlexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/* Examine the column types */
for (colnum = 1; colnum < 4; colnum++)
{
coltype = dbcoltype(dbproc, colnum);
printf("column %d, type is %s.¥n", colnum,
dbprtype(coltype));
}
```

参照　dbcollen、dbcolname、dbdata、dbdatlen、dbnumcols、dbprtype、dbvarylen、「データ型」(443 ページ)

dbcoltypeinfo

説明　numeric 型または decimal 型の通常の結果カラムの精度と位取りの情報を返します。

構文　DBTYPEINFO * dbcoltypeinfo(dbproc, column)

```
DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定の numeric または decimal のカラムの精度と位取りの値が格納されている DBTYPEINFO 構造体を指すポインタです。指定されたカラム番号が結果セットにない場合は NULL を返します。

DBTYPEINFO 構造体は、次のように定義されています。

```
typedef struct typeinfo {
     DBINT    precision;
     DBINT    scale;
} DBTYPEINFO;
```

カラムのデータ型が numeric または decimal でない場合は、返される構造体に意味のない値が含まれることがあります。この関数を呼び出す前に、dbcoltype が SYBNUMERIC または SYBDECIMAL を返すことを確認してください。

使用法

- このルーチンは、データ型が numeric または decimal である通常 (非計算) 結果カラムの精度と位取りの値が格納されている DBTYPEINFO 構造体へのポインタを返します。
- 他のデータ型のカラムの場合は、返される精度と位取りの値は意味を持ちません。dbcoltypeinfo を呼び出す前に、dbcoltype が SYBNUMERIC または SYBDECIMAL を返すことを確認してください。

参照

dbcollen、dbcolname、dbcoltype、dbdata、dbdatlen、dbnumcols、dbprtype、dbvarylen、「データ型」(443 ページ)

dbcolutype

説明

通常の結果カラムのユーザ定義データ型を返します。

構文

DBINT dbcolutype(dbproc, column)

```
DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
　対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

column のユーザ定義データ型です。*column* が範囲外の場合は負の整数です。

使用法

- dbcolutype は、通常の結果カラムのユーザ定義データ型を返します。ユーザ定義データ型を Adaptive Server Enterprise データベースに追加する方法については、『ASE リファレンス・マニュアル』の sp_addtype を参照してください。
- dbcolutype は、ユーザ定義データ型のサイズを取り込めるよう、データ型 DBINT として定義されます。DBINT とユーザ定義データ型は、どちらも長さが 32 ビットです。
- 次のコードは、dbcolutype の例です。

```
DBPROCESS     *dbproc;
 int          colnum;
 int          numcols;

 /* Put the command into the command buffer */
 dbcmd(dbproc, "select * from mytable");

 /*
 ** Send the command to the Adaptive Server
Enterprise and begin
 ** execution.
 */
 dbsqlexec(dbproc);

 /* Process the command results */
 dbresults(dbproc);

 /* Examine the user-defined column types */
 numcols =  dbnumcols(dbproc);
 for (colnum = 1; colnum < numcols; colnum++)
 {
      printf ("column %d, user-defined type is ¥
           %ld.¥n", colnum, dbcolutype(dbproc,
           colnum));
 }
```

参照 dbaltutype、dbcoltype

dbconvert

説明 あるデータ型から別のデータ型にデータを変換します。

構文

```
DBINT dbconvert(dbproc, srctype, src, srclen,
        desttype, dest, destlen)

DBPROCESS    *dbproc;
int          srctype;
BYTE         *src;
DBINT        srclen;
int          desttype;
BYTE         *dest;
DBINT        destlen;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。dbconvert で DBPROCESS を使用する目的は、dbsetnull によって指定された独自の NULL 値を代入することだけです。*dbproc* が NULL の場合、dbconvert は NULL 値のデータ変換にデフォルト値を使用します。

srctype

変換されるデータのデータ型です。このパラメータは、表 2-7 (116 ページ) に示すサーバ・データ型のいずれかです。

src

変換されるデータを指すポインタです。このポインタが NULL の場合、dbconvert は、該当する NULL 値を変換先変数に格納します。サーバ・データを取得するには、dbdata を使用してください。

srclen

変換されるデータのバイト単位の長さです。*srclen* が 0 の場合、変換元データは NULL であるとみなされ、dbconvert は該当する NULL 値を変換先変数に格納します。それ以外のとき、データ型が char、text、binary、または image でなければ、この長さは無視されます。SYBCHAR、SYBBOUNDARY、SYBSENSITIVITY データの場合、長さを -1 とすることは、文字列が NULL で終了することを示します。サーバ・データの長さを取得するには、dbdatlen を使用してください。

desttype
　データの変換先のデータ型です。このパラメータは、表 2-7 (116 ページ) に示すサーバ・データ型のいずれかです。

dest
　変換されたデータを受け取る変換先変数を指すポインタです。このポインタが NULL の場合、dbconvert は、ユーザ提供のエラー・ハンドラがあればそれを呼び出し、-1 を返します。

destlen
　変換先変数のバイト単位の長さです。固定長データ型の場合、*destlen* は無視されます。変換先が SYBCHAR、SYBBOUNDARY、または SYBSENSITIVITY の場合、*destlen* の値は、変換先バッファ領域の全長としてください。

　表 2-5 に *destlen* の特別な値を示します。

表 2-5 : destlen の特別な値 (dbconvert)

destlen の値	適用	意味
-1	SYBCHAR, SYBBOUNDARY, SBYSENSITIVITY	使用可能な領域が十分にある。 文字列の後続ブランクがトリムされ、終了の NULL が追加される。
-2	SYBCHAR	使用可能な領域が十分にある。 文字列の後続ブランクはトリムされず、終了の NULL が追加される。

戻り値

データ型の変換が正常に終了した場合は、変換されたデータのバイト単位の長さです。

変換が失敗した場合、dbconvert は、原因に応じて -1 または FAIL を返します。dbconvert は、変換先ポインタが NULL であるか、またはデータ型が正しくない場合に -1 を返します。dbconvert は、それ以外の原因の場合に FAIL を返します。

dbconvert は、失敗すると、ユーザ提供のエラー・ハンドラがある場合は最初にそれを呼び出して、DB-Library のグローバル・エラー値を設定します。

このルーチンが失敗する原因はいくつかあります。要求した変換が使用できない場合、変換によってトランケーション、オーバフロー、または変換先変数の精度の損失などが起きた場合、または文字列を数値型に変換したときに構文エラーが起きた場合などが挙げられます。

使用法

- このルーチンをプログラムで使用すると、データの表現を別の表現に変換できます。特定の変換が可能かどうかを調べるには、変換を行う前に dbwillconvert を呼び出します。
- dbconvert は、どのサーバ・データ型で保管されているデータでも変換できます (すべての変換が可能というわけではありません)。型定数と対応するプログラム変数型については、表 2-7 (116 ページ) を参照してください。
- ライブラリのバージョンが dbsetversion を使用して DBVERSION_100 以上に設定されていない場合は、dbconvert で SYBNUMERIC、SYBDECIMAL、SYBBOUNDARY、または SYBSENSITIVITY を使用するとエラーになります。
- 表 2-8 (117 ページ) に、dbconvert がサポートするデータ型変換を示します。表の左端の列が変換元のデータ型のリストで、上端の行が変換先のデータ型のリストです (簡潔にするために、各データ型のプレフィクスの「SYB」は省略されています)。T (「True」) は変換がサポートされることを示し、F (「False」) は変換がサポートされないことを示します。
- SYBBINARY データ型と SYBIMAGE データ型との間の変換は、SYBCHAR または SYBTEXT が関係する場合を除いて、ストレート・ビット・コピーです。SYBCHAR または SYBTEXT のデータを SYBBINARY または SYBIMAGE に変換するとき、DBCONVERT は SYBCHAR または SYBTEXT の文字列に先行「0x」が含まれているかどうかに関係なく、この文字列を 16 進数と解釈します。SYBBINARY または SYBIMAGE のデータを SYBCHAR または SYBTEXT に変換するときは、dbconvert は先行「0x」を付けずに 16 進数文字列を作成します。
- SYBINT2 と SYBINT4 は、符号付きデータ型です。これらのデータ型を文字に変換するとき、変換される数が符号無しであり、しかも上位ビットを使用している場合は、変換エラーが発生することがあります。
- SYBMONEY、SYBCHAR、または SYBTEXT の値を SYBFLT8 に変換するときは、精度が落ちることがあります。SYBFLT8 の値を SYBCHAR または SYBTEXT に変換するときにも、精度が落ちることがあります。
- SYBMONEY の最大値は $922,337,203,685,477.58 なので、SYBFLT8 値を SYBMONEY に変換すると、オーバフローが発生することがあります。

- 整数または浮動小数点のデータを SYBCHAR または SYBTEXT に変換するときにオーバフローが発生すると、結果の値の最初の文字は、エラーを示すアスタリスク (*) となります。
- 変換先が SYBBIT のとき、変換される値が 0 でない場合は、SYBBIT 値は 1 に設定され、変換される値が 0 の場合は、SYBBIT 値は 0 に設定されます。
- dbconvert は、numeric データ型と decimal データ型に対する精度と位取りをサポートしていません。SYBNUMERIC または SYBDECIMAL に変換するとき、dbconvert は、デフォルト値である精度 18 と位取り 0 を使用します。アプリケーションでデフォルト以外の精度と位取りを指定するには、dbconvert_ps を使用します。
- SYBBOUNDARY と SYBSENSITIVITY に変換する場合は、必ず NULL ターミネータ付きになります。
- データ型を同じデータ型へ変換することが役立つ場合もあります。たとえば、以下の例に示すように *destlen* に -1 を指定して SYBCHAR を SYBCHAR に変換することで、文字列に NULL ターミネータを簡単に追加できます。
- 次の例は、dbdata で取得したサーバ・データを変換する方法を示します。

```
DBCHAR      title[81];
 DBCHAR      price[9];

 /* Read the query into the command buffer */
 dbcmd(dbproc, "select title, price, royalty from ¥
     pubs2..titles");

 /* Send the query to Adaptive Server Enterprise */
 dbsqlexec(dbproc);

 /* Get ready to process the query results */
 dbresults(dbproc);

 /* Process each row */
 while (dbnextrow(dbproc) != NO_MORE_ROWS)
 {
     /*
     ** The first dbconvert() adds a null
     ** terminator to the string.
     */
     dbconvert(dbproc, SYBCHAR, (dbdata(dbproc,1)),
         (dbdatlen(dbproc,1)), SYBCHAR, title,
         (DBINT)-1);
```

```
        /*
        ** The second dbconvert() converts money to
        ** string.
        */
        dbconvert(dbproc, SYBMONEY,
           (dbdata(dbproc,2)), (DBINT)-1, SYBCHAR,
            price, (DBINT)-1);

        if (dbdatlen(dbproc,3) != 0)
            printf ("%s¥n $%s  %ld¥n", title, price,
            *((DBINT *)dbdata(dbproc,3)));
    }
```

dbconvert 呼び出しでは、dbdata からの戻り値をキャストする必要はありません。dbdata は BYTE ポインタを返しますが、これは dbconvert が 3 番目のパラメータとして受け取るデータ型と同じであるためです。

- dbdata を使用して直接データにアクセスするのではなく、dbbind を使用してデータを変数にバインドする場合は、dbbind 自体が変換を行うので、dbconvert は不要です。
- オンラインのサンプル・プログラムの *example5.c* には、dbconvert を使用する変換例が含まれています。
- DB-Library のデータ型と対応する Adaptive Server Enterprise データ型のリストについては、「データ型」(443 ページ) を参照してください。『ASE リファレンス・マニュアル』を参照してください。

参照

dbaltbind、dbaltbind_ps、dbbind、dbbind_ps、dbconvert_ps、dberrhandle、dbsetnull、dbsetversion、dbwillconvert、「エラー」(418 ページ)、「データ型」(443 ページ)

dbconvert_ps

説明

データを別のデータ型に変換します。numeric データ型と decimal データ型の精度と位取りをサポートします。

構文

```
DBINT dbconvert_ps(dbproc, srctype, src, srclen,
            desttype, dest, destlen, typeinfo)

DBPROCESS    *dbproc;
int          srctype;
BYTE         *src;
DBINT        srclen;
```

```
int             desttype;
BYTE            *dest;
DBINT           destlen;
DBTYPEINFO      *typeinfo;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。dbconvert_ps で DBPROCESS を使用する目的は、dbsetnull によって指定された独自の NULL 値を代入することだけです。*dbproc* が NULL の場合、dbconvert_ps は NULL 値のデータ変換にデフォルト値を使用します。

srctype
変換されるデータのデータ型です。このパラメータは、表 2-8 (117 ページ) に示すサーバ・データ型のいずれかです。

src
変換されるデータを指すポインタです。このポインタが NULL の場合、dbconvert_ps は該当する NULL 値を変換先変数に格納します。サーバ・データを取得するには、dbdata を使用してください。

srclen
変換されるデータのバイト単位の長さです。*srclen* が 0 の場合、変換元データは NULL であるとみなされ、dbconvert_ps は該当する NULL 値を変換先変数に格納します。それ以外のとき、データ型が char、text、binary、または image でなければ、この長さは無視されます。SYBCHAR データの場合、長さを -1 とすることは、文字列が NULL で終了することを示します。サーバ・データの長さを取得するには、dbdatlen を使用してください。

desttype
データの変換先のデータ型です。このパラメータは、表 2-8 (117 ページ) に示すサーバ・データ型のいずれかです。

dest
変換されたデータを受け取る変換先変数を指すポインタです。このポインタが NULL の場合、dbconvert_ps は、ユーザ提供のエラー・ハンドラがあればそれを呼び出し、-1 を返します。

destlen

変換先変数のバイト単位の長さです。固定長データ型の場合、*destlen* は無視されます。変換先が SYBCHAR、SYBBOUNDARY、または SYBSENSITIVITY の場合、*destlen* の値は、変換先バッファ領域の全長としてください。

表 2-6 に *destlen* の特別な値を示します。

表 2-6：destlen の特別な値 (dbconvert_ps)

destlen の値	適用	意味
-1	SYBCHAR, SYBBOUNDARY, SBYSENSITIVITY	使用可能な領域が十分にある。 文字列の後続ブランクがトリムされ、終了の NULL が追加される。
-2	SYBCHAR	使用可能な領域が十分にある。 文字列の後続ブランクはトリムされず、終了の NULL が追加される。

typeinfo

decimal または numeric の値の精度と位取りに関する情報が格納されている DBTYPEINFO 構造体へのポインタです。アプリケーションでは、dbconvert_ps を呼び出してデータを DBDECIMAL または DBNUMERIC の変数に変換する前に、DBTYPEINFO 構造体に精度と位取りの値を設定します。

typeinfo が NULL の場合

- 変換元の値のデータ型が SYBNUMERIC または SYBDECIMAL の場合、dbconvert_ps は、変換元の値から精度と位取りの値を取得します。これにより、変換元データが変換先領域にコピーされます。
- 変換元の値のデータ型が SYBNUMERIC または SYBDECIMAL ではない場合、dbconvert_ps は、デフォルト値である精度 18 と位取り 0 を使用します。

srctype が SYBDECIMAL または SYBNUMERIC ではない場合、*typeinfo* は無視されます。

DBTYPEINFO 構造体は、次のように定義されています。

```
typedef struct typeinfo {
     DBINT    precision;
     DBINT    scale;
} DBTYPEINFO;
```

precision の有効な値は、1 から 77 までです。*scale* の有効な値は、0 から 77 までです。*scale* の値は *precision* 以下でなければなりません。

戻り値

データ型の変換が正常に終了した場合は、変換されたデータのバイト単位の長さです。

変換が失敗した場合、dbconvert_ps は、原因に応じて -1 または FAIL を返します。dbconvert_ps は、変換先ポインタが NULL であるか、またはデータ型が正しくない場合に -1 を返します。dbconvert_ps は、それ以外の原因の場合に FAIL を返します。

dbconvert_ps は、失敗すると、ユーザ提供のエラー・ハンドラがある場合は最初にそれを呼び出して、DB-Library のグローバル・エラー値を設定します。

このルーチンが失敗する原因はいくつかあります。要求した変換が使用できない場合、変換によってトランケーション、オーバフロー、または変換先変数の精度の損失などが起きた場合、または文字列を数値型に変換したときに構文エラーが起きた場合などが挙げられます。

使用法

- dbconvert_ps は、numeric データ型と decimal データ型の精度と位取りをサポートする点を除いて dbconvert と同じです。dbconvert はこれらをサポートしていません。dbconvert を呼び出すことは、*typeinfo* に NULL を指定して dbconvert_ps を呼び出すこととまったく同じです。
- dbconvert_ps を使用すると、プログラムでデータの表現を別の表現に変換できます。特定の変換が可能かどうかを調べるには、変換を行う前に dbwillconvert を呼び出します。
- dbconvert_ps は、どのサーバ・データ型で保管されているデータでも変換できます (すべての変換が可能というわけではありません。表 2-8 (117 ページ) を参照してください)。

  表 2-7 は、サーバ・データ型の型定数と対応するプログラム変数型を示します。

表 2-7：型定数とプログラム変数の型

サーバ・データ型定数	プログラム変数タイプ
SYBCHAR	DBCHAR
SYBTEXT	DBCHAR
SYBBINARY	DBBINARY
SYBIMAGE	DBBINARY
SYBINT1	DBTINYINT
SYBINT2	DBSMALLINT
SYBINT4	DBINT
SYBFLT8	DBFLT8
SYBREAL	DBREAL
SYBNUMERIC	DBNUMERIC
SYBDECIMAL	DBDECIMAL
SYBBIT	DBBIT
SYBMONEY	DBMONEY
SYBMONEY4	DBMONEY4
SYBDATETIME	DBDATETIME
SYBDATETIME4	DBDATETIME4
SYBBOUNDARY	DBCHAR
SYBSENSITIVITY	DBCHAR

警告！ライブラリのバージョンが DBVERSION_100 以上に設定されていない場合は (設定するには dbsetversion を使用します)、dbconvert_ps で SYBNUMERIC、SYBDECIMAL、SYBBOUNDARY、SYBSENSITIVITY を使用するとエラーになります。

- 表 2-8 に、dbconvert_ps がサポートするデータ型変換を示します。左側に変換元のデータ型が縦に並べられ、上端に変換先のデータ型が横に並べられています (簡潔にするために、データ型のプレフィクスの「SYB」は省略されています)。

表 2-8：サポートされているデータ型変換

	次のように変更します。																	
変換元：	**CHAR**	**TEXT**	**BINARY**	**IMAGE**	**INT1**	**INT2**	**INT4**	**FLT8**	**REAL**	**NUMERIC**	**DECIMAL**	**BIT**	**MONEY**	**MONEY4**	**DATETIME**	**DATETIME4**	**BOUNDARY**	**SENSITIVITY**
CHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
TEXT	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
BINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
IMAGE	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
INT1	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
INT2	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
INT4	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
FLT8	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
REAL	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
NUMERIC	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
DECIMAL	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
BIT	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
MONEY	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
MONEY4	•	•	•	•	•	•	•	•	•	•	•	•	•	•				
DATETIME	•	•	•	•											•	•		
DATETIME4	•	•	•	•											•	•		
BOUNDARY	•	•															•	
SENSITIVITY	•	•																•

- SYBBINARY データ型と SYBIMAGE データ型との間の変換は、SYBCHAR または SYBTEXT が関係する場合を除いて、ストレート・ビット・コピーです。SYBCHAR または SYBTEXT のデータを SYBBINARY または SYBIMAGE に変換するとき、dbconvert_ps は SYBCHAR または SYBTEXT の文字列に先行「0x」が含まれているかどうかに関係なく、この文字列を 16 進数と解釈します。SYBBINARY または SYBIMAGE のデータを SYBCHAR または SYBTEXT に変換するときは、dbconvert_ps は先行「0x」を付けずに 16 進数文字列を作成します。
- SYBINT2 と SYBINT4 は、符号付きデータ型です。これらのデータ型を文字に変換するとき、変換される数が符号無しであり、しかも上位ビットを使用している場合は、変換エラーが発生することがあります。

- SYBMONEY、SYBCHAR、または SYBTEXT の値を SYBFLT8 に変換するときは、精度が落ちることがあります。SYBFLT8 の値を SYBCHAR または SYBTEXT に変換するときにも、精度が落ちることがあります。
- SYBMONEY の最大値は $922,337,203,685,477.58 なので、SYBFLT8 値を SYBMONEY に変換すると、オーバフローが発生することがあります。
- 整数または浮動小数点のデータを SYBCHAR または SYBTEXT に変換するときにオーバフローが発生すると、結果の値の最初の文字は、エラーを示すアスタリスク (*) となります。
- 変換先が SYBBIT のとき、変換される値が 0 でない場合は、SYBBIT 値は 1 に設定され、変換される値が 0 の場合は、SYBBIT 値は 0 に設定されます。
- SYBBOUNDARY と SYBSENSITIVITY に変換する場合は、必ず NULL ターミネータ付きになります。
- データ型を同じデータ型へ変換することが役立つ場合もあります。たとえば、*destlen* に -1 を指定して SYBCHAR を SYBCHAR に変換することで、文字列に NULL ターミネータを簡単に追加できます。
- dbdata を使用して直接データにアクセスするのではなく、dbbind または dbbind_ps を使用してデータを変数にバインドする場合は、dbbind 自体が変換を行うので、dbconvert_ps は不要です。
- オンラインのサンプル・プログラムの *example5.c* には、dbconvert_ps を使用する変換例が含まれています。
- DB-Library のデータ型と対応する Adaptive Server Enterprise データ型のリストについては、「データ型」(443 ページ) を参照してください。『ASE リファレンス・マニュアル』を参照してください。

参照

dbaltbind、dbaltbind_ps、dbbind、dbbind_ps、dbconvert、dberrhandle、dbsetnull、dbsetversion、dbwillconvert、「エラー」(418 ページ)、「データ型」(443 ページ)

DBCOUNT

説明

Transact-SQL コマンドによる影響を受けたローの数を返す。

構文

```
DBINT DBCOUNT(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

コマンドの影響を受けたローの数または -1 です。次のいずれかが true である場合、DBCOUNT は -1 を返します。

- Transact-SQL コマンドが、構文エラーなど、何らかの理由で失敗する場合
- コマンドが、print コマンドなどの、ローに影響を及ぼさないコマンドである。
- コマンドが実行するストアド・プロシージャの中で select 文がまったく実行されない。
- DBNOCOUNT オプションが on になっている。

使用法

- コマンドの結果が処理された後で、DBCOUNT を呼び出すと、そのコマンドの影響を受けたローの数を調べることができます。たとえば、select コマンドをサーバに送信してから、NO_MORE_ROWS が返されるまで dbnextrow を呼び出してすべてのローを読み込んだ場合に、このマクロを呼び出すと、取得されたローの数がわかります。
- 現在のコマンドがローを返さないコマンド (delete など) の場合は、dbresults のすぐ後に DBCOUNT を呼び出すことができます。
- 現在のコマンドがストアド・プロシージャを実行する、つまり exec またはリモート・プロシージャ・コールを実行するコマンドの場合は、DBCOUNT はそのストアド・プロシージャによって最後に実行された select 文から返されたローの数を返します。そのストアド・プロシージャが select 文をまったく実行していない場合は、-1 を返します。ストアド・プロシージャに select 文が含まれていなくても、select 文を含む別のストアド・プロシージャを呼び出して select を実行する場合もあることに注意してください。

参照

dbnextrow、dbresults、「オプション」(436 ページ)

DBCURCMD

説明

現在のコマンドの番号を返します。

構文

```
int DBCURCMD(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

現在のコマンドの番号です。

使用法

- このマクロは、現在結果を処理中のコマンドの番号を返します。
- バッチ内の最初のコマンドの番号は 1 です。コマンド番号は、dbresults が SUCCEED または FAIL を返すたびに増加します (失敗したコマンドはカウントされません)。コマンド番号は、dbsqlexec または dbsqlsend を呼び出すたびにリセットされます。

参照

DBCMDROW、DBMORECMDS、DBROWS

DBCURROW

説明

現在読み込み中のローの番号を返します。

構文

```
DBINT DBCURROW(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

現在のローの番号です。まだローが 1 つも処理されていない場合は、0 を返します。

使用法

- このマクロは、現在読み込み中のローの番号を返します。サーバから最初に返されたローの番号が 1 となり、順にカウントされます。DBCURROW は通常ローと計算ローの両方をカウントします。

- dbresults を新しく呼び出すたびに、ロー番号は 0 にリセットされます。
- dbnextrow が REG_ROW または *computeid* を返すたびに、ロー番号が 1 ずつ増加します。
- ロー・バッファリングを使用するときも、ロー番号はロー・バッファ内の位置を表すのではありません。サーバから返されたローの中での現在のローの位置を表します。dbgetrow と dbsetrow のリファレンス・ページを参照してください。

参照　dbclrbuf、DBFIRSTROW、dbgetrow、DBLASTROW、dbnextrow、dbsetopt、「オプション」(436 ページ)

dbcursor

説明　フェッチ・バッファ内の特定のローを挿入、更新、削除、ロック、またはリフレッシュします。

構文

```
RETCODE dbcursor(hc, optype, bufno, table, values)

DBCURSOR    *hc;
DBINT       optype;
DBINT       bufno;
BYTE        *table;
BYTE        *values
```

パラメータ

hc

これより前に呼び出した dbcursoropen から返されたカーソル・ハンドルです。

optype

表 2-9 : optype の値 (dbcursor)

記号値	オペレーション
CRS_UPDATE	データを更新する。
CRS_DELETE	データを削除する。
CRS_INSERT	データを挿入する。
CRS_REFRESH	バッファ内の別のローをフェッチする。
CRS_LOCKCC	別のローをフェッチしてロックする。ローが実際にブロックされるのは、トランザクション・ブロック内の場合だけである。アプリケーションがトランザクションをコミットするか終了すると、ロックは解放される。

bufno

オペレーションの対象となる、フェッチ・バッファ内のローの番号です。指定したバッファには、正しいローが入っていなければなりません。*bufno* の値が 0 の場合、CRS_REFRESH オペレーションは、バッファ内のすべてのローに対して行われます。insert オペレーションまたは update オペレーションに values パラメータが指定されていない場合は、値は対応する *bufno* 値のバインドされた変数配列から読み込まれます。バッファ内の最初のローの番号は 1 です。

table

カーソル宣言に複数のテーブルが含まれている場合に、挿入、更新、削除、またはロックするテーブルを指定します。テーブルが 1 つだけの場合、このパラメータは必要ありません。

values

更新または挿入される文字列値です。このパラメータは、update と insert で新しいカラム値を指定する場合にだけ使用してください (たとえば、*Quantity*=*Quantity*+1)。ほとんどの場合はこのパラメータを NULL に設定でき、各カラムの新しい値は、フェッチ・バッファ (dbcursorbind で指定されたプログラム変数) から取得されます。select 文に計算 (たとえば、select 5*5...) と関数 (たとえば、select getdate()、convert() など) が含まれている場合は、バッファ配列からの更新は正しく行われません。

このパラメータには、更新の場合に 2 種類、挿入の場合に 2 種類の、合計 4 つのフォーマットがあります。*optype* (update または insert) に合ったフォーマットを選択してください。どちらにも完全フォーマットと省略フォーマットがあります。完全フォーマットは、where 句を除いた完全な SQL 文 (update または insert) です。省略フォーマットは、set 句だけ (update)、または values 句だけ (insert) で構成されます。完全フォーマットを使用する場合は、*tablename* に指定した値が dbcursor の *table* パラメータよりも優先されます。where 句は自動的に追加されるので、指定しないでください。

戻り値

SUCCEED または FAIL。

この関数が失敗する原因には、次のものがあります。

- カーソルが読み込み専用としてオープンされていて、更新が許可されていない。
- サーバまたは接続の障害、またはタイムアウトが発生した。
- データベースの更新または変更のパーミッションがない。
- データベース内のトリガが原因で、lock または update/insert オペレーションが失敗する。
- オプティミスティック同時実行制御。

使用法

- ユニーク・インデックス・カラムとして使用されるカラムを更新または変更すると、対応するローは、次にフェッチされたときに見つからないとみなされます。
- 「付録 A カーソル」を参照してください。

参照

dbcursorbind、dbcursorclose、dbcursorcolinfo、dbcursorfetch、dbcursorinfo、dbcursoropen

dbcursorbind

説明

カーソル・カラムのバインド情報を登録します。

構文

```
RETCODE dbcursorbind(hc, col, vartype, varlen,
                     poutlen, pvaraddr, typeinfo)

DBCURSOR     *hc;
int          col;
int          vartype;
DBINT        varlen;
DBINT        *poutlen;
BYTE         *pvaraddr;
DBTYPEINFO   *typeinfo;
```

パラメータ

hc

dbcursoropen によって作成されたカーソル・ハンドルです。

col

プログラム変数にバインドされるカラムの番号です。

vartype

バインド・データ型です。このデータ型は dbbind の *vartype* パラメータと同じデータ型を使用し、同じ変換ルールによってバインドされます。カラムに対してこの値を NOBIND に設定すると、データはバインドされません。その代わりに、データを指すポインタは、各ローの対応する *pvaraddr* エントリ内のアドレスに返され、データの長さは対応する *varlen* 配列エントリに返されます。この機能により、アプリケーションは、dbdata と dbdatalen で行うのと同じようにカーソル・データにアクセスできます。

varlen

CHARBIND、VARYCHARBIND、BINARYBIND、STRINGBIND、NTBSTRINGBIND、VARYBINBIND などの可変長データ型の最大長。このパラメータは、INTBIND、FLT8BIND、MONEYBIND、BITBIND、SMALLBIND などの固定長データ型の場合は無視されます。vartype が NOBIND の場合も、このパラメータは無視されます。

poutlen

各ローのカラム・データの実際の長さが返される DBINT 整数の配列を指すポインタです。*poutlen* が NULL に設定されている場合は、長さは返されません。配列のサイズは、一度にフェッチされるロー (dbcursoropen の *nrows* パラメータで指定) のそれぞれについて 1 つずつ DBINT 変数を保管するのに十分な大きさにしてください。

dbcursor を使用して、バインドされたプログラム変数からの値で更新または挿入を行うとき、NULL 値を指定するには、dbcursor を呼び出す前に、対応する *poutlen* を 0 に設定します。NOBIND、または VARYCHARBIND や VARYBINBIND などの可変長データ型が指定されている場合を除いて、0 以外の値は無視されます。可変長データ型の場合、*poutlen* には、実際の項目の長さを指定してください。STRINGBIND または NTBSTRINGBIND が指定されている場合、*poutlen* の 0 以外の値は無視され、文字列の長さは NULL ターミネータをスキャンして判断されます。

pvaraddr

データのコピー先であるプログラム変数を指すポインタです。*vartype* が NOBIND の場合、*pvaraddr* はポインタ (dbcursorfetch によってフェッチされた実際のデータのアドレスへのポインタ) の配列を指すとみなされます。この配列の長さは、dbcursoropen の *nrows* の値と等しくなければなりません。カーソルがオープンされたときの *nrows* が 1 より大きい場合は、*pvaraddr* は *nrows* 個の要素を持つ配列を指すとみなされます。*pvaraddr* を NULL に設定して dbcursorbind を呼び出すと、既存のバインドは破棄されます。

typeinfo

decimal または numeric の値の精度と位取りに関する情報が格納されている DBTYPEINFO 構造体へのポインタです。*vartype* が DECIMALBIND または NUMERICBIND ではない場合、*typeinfo* は無視されます。

DBNUMERIC または DBDECIMAL の変数にバインドする場合は、アプリケーションは DBTYPEINFO 構造体を初期化して精度と位取りの値を設定してから、*vartype* に DECIMALBIND または NUMERICBIND を指定して dbcursorbind を呼び出します。

typeinfo が NULL で、*vartype* が DECIMALBIND または NUMERICBIND である場合

- 結果カラムのデータ型が numeric または decimal の場合は、dbcursorbind は制度と位取りの値を結果カラムから取得します。
- 結果カラムのデータ型が numeric または decimal でない場合は、dbcursorbind はデフォルト値である精度 18 と位取り 0 を使用します。

DBTYPEINFO 構造体は、次のように定義されています。

```
typedef struct typeinfo {
      DBINTprecision;
      DBINTscale;
 } DBTYPEINFO;
```

precision の有効な値は、1 から 77 までです。*scale* の有効な値は、0 から 77 までです。*scale* の値は *precision* 以下でなければなりません。

戻り値　SUCCEED または FAIL。

使用法

- 同じカラムに対して dbcursorbind を 2 回以上呼び出した場合は、最後の呼び出しだけが有効になります。
- この関数の機能は、カーソルなしの dbbind とほぼ同じです。
- 「付録 A カーソル」を参照してください。

参照　dbcursor、dbcursorclose、dbcursorcolinfo、dbcursorfetch、dbcursorinfo、dbcursoropen

dbcursorclose

説明　指定のハンドルに対応しているカーソルをクローズして、そのカーソルに属するすべてのデータを解放します。

構文

```
void dbcursorclose(hc)

DBCURSOR     *hc;
```

パラメータ

hc

dbcursoropen によって作成されたカーソル・ハンドルです。

戻り値　なし。

使用法

- dbcursorclose を使用して DBPROCESS 接続をクローズすると、この接続に関連付けられているすべてのカーソルが自動的にクローズされます。dbcursorclose を発行した後は、カーソル・ハンドルを使用しないでください。
- 「付録 A カーソル」を参照してください。

参照

dbcursor、dbcursorbind、dbcursorcolinfo、dbcursorfetch、dbcursorinfo、dbcursoropen

dbcursorcolinfo

説明

オープン・カーソル内の指定カラム番号のカラム情報を返します。

構文

```
RETCODE dbcursorcolinfo(hcursor, column, colname,
                        coltype, collen, usertype)

DBCURSOR        *hcursor
DBINT           column;
DBCHAR          *colname;
DBINT           *coltype;
DBINT           *collen;
DBINT           *usertype;
```

パラメータ

hcursor
dbcursoropen によって作成されたカーソル・ハンドルです。

column
どのカラムの情報を返すかを番号で指定します。

colname
カラム名を返すロケーションです。カラム名を格納するのに十分な大きさの領域を割り付けておく必要があります。

coltype
カラムのデータ型を返すロケーションです。

collen
カラムの最大長を返すロケーションです。

usertype
カラムのユーザ定義データ型を返すロケーションです。

戻り値

SUCCEED または FAIL。

使用法

- パラメータ

  colname、*coltype*、*collen*、*usertype* はいずれも NULL に設定できます。NULL に設定した場合は、その変数の情報は返されません。

- 参照先「付録 A カーソル」

参照

dbcursor、dbcursorbind、dbcursorclose、dbcursorfetch、dbcursorinfo、dbcursoropen

dbcursorfetch

説明

ローのブロックをフェッチして、dbcursorbind でユーザが宣言したプログラム変数に出力します。

構文

RETCODE dbcursorfetch(hc, fetchtype, rownum)

```
DBCURSOR    *hc;
DBINT       fetchtype;
DBINT       rownum;
```

パラメータ

hc

dbcursoropen によって作成されたカーソル・ハンドルです。

fetchtype

選択するフェッチのタイプです。選択できる値は、dbcursoropen のスクロール・オプションによって決まります。表 2-10 に、さまざまなフェッチ・タイプをリストします。

表 2-10：fetchtype の値 (dbcursorfetch)

記号値	意味	コメント
FETCH_FIRST	ローの最初のブロックをフェッチする。	このオプションは、すべてのカーソル・タイプに使用できるが、特に前方スクロールのみのカーソルを選択しているときにキーセットの先頭に戻る場合に便利。
FETCH_NEXT	ローの次のブロックをフェッチする。	結果セットが指定のキーセット・サイズを超えており、FETCH_RANDOM か FETCH_RELATIVE、またはその両方がすでに発行されている場合に、FETCH_NEXT を発行するとキーセット境界までがフェッチされる。この場合、キーセット境界までのフェッチでは部分バッファが返され、次のフェッチではキーセットが下にシフトして次のフル・セットのローが返される。
FETCH_PREV	前のローのブロックをフェッチする。	このオプションは、前方スクロールのみのカーソルには使用できない。*rownum* がキーセットの範囲内である場合、キーセット内のローだけが返されるので、ローの範囲はキーセットの範囲内でなければならない。このオプションで、キーセットが結果セット内の前の *rownum* ローに変更されることはない。
FETCH_RANDOM	キーセット内の指定のロー番号から始まるローのブロックをフェッチする。	このオプションは、キーセット内だけで有効。範囲がキーセット境界に及ぶ場合は、バッファの一部だけに値が入る。
FETCH_RELATIVE	最後のフェッチで指示されたロー数との相対位置で、ローのブロックをフェッチする。	このオプションは、最後のフェッチの最初のローから *rownum* だけ離れたローからフェッチを始める。フェッチするローは、キーセットの範囲内でなければならない。範囲がキーセット境界に及ぶ場合は、バッファの一部だけに値が入る。
FETCH_LAST	最後のローのブロックをフェッチする。	この値は、完全キーセット駆動型カーソルだけに使用可能。

rownum
ここで指定したローからバッファへの挿入が開始します。このパラメータは、*fetchtype* が FETCH_RANDOM または FETCH_RELATIVE の場合にだけ使用してください。

戻り値

SUCCEED または FAIL。

ステータス配列に、フェッチされたすべてのローに対応するステータス・ローが格納されていれば、SUCCEED が返されます。次のいずれかに該当する場合は FAIL が返されます。

- FETCH_RANDOM と FETCH_RELATIVE にはキーセット駆動型カーソルが必要。
- 前方スクロールのみの場合は FETCH_FIRST と FETCH_NEXT しか使用できない。
- サーバまたは接続に障害が発生しているか、タイムアウトになっている。
- クライアントのメモリ不足。
- FETCH_LAST オプションには完全キーセット駆動型カーソルが必要。

使用法

- dbcursoropen でフェッチ・バッファのサイズを指定します。dbcursorfetch を実行すると、dbcursoropen の *pstatus* パラメータとして渡された配列に、フェッチしたローのステータス・コードが順に格納されます。これらのコードについては、dbcursoropen のリファレンス・ページを参照してください。
- 初めに dbcursorbind を使用してプログラム変数を登録する必要があります。登録すると、データを DB-Library バッファに転送できるようになります。したがって、バインドされる変数は、指定した数のローを入れるのに十分な大きさの配列でなければなりません。ステータス配列には、各ローのステータス・コードと見つからないローのフラグが格納されます。
- FETCH_NEXT、FETCH_RANDOM、または FETCH_RELATIVE で指定されたローの範囲がキーセット境界に及ぶ場合は、キーセット内の残りのローだけが返されます。この場合はバッファの一部だけに値が入り、FETCH_ENDOFKEYSET フラグは最後のローのステータスに設定されます。次の FETCH_NEXT では、キーセットは下にシフトします。
- 参照先「付録 A カーソル」

参照

dbcursor、dbcursorbind、dbcursorclose、dbcursorcolinfo、dbcursorinfo、dbcursoropen

dbcursorinfo

説明

キーセットが結果セットの最後に到達している場合に、キーセット内のカラム数とロー数を返します。

構文

```
RETCODE dbcursorinfo(hcursor, ncols, nrows);

DBCURSOR    *hcursor;
DBINT       *ncols
DBINT       *nrows;
```

パラメータ

hcursor
dbcursoropen によって作成されたカーソル・ハンドルです。

ncols
カーソル内のカラムの数を返すロケーションです。

nrows
キーセット内のローの数を返すロケーションです。

戻り値

SUCCEED または FAIL。

使用法

- 完全キーセット駆動型カーソルの場合は、*nrows* パラメータで指定されたロケーションにキーセット内のロー数が格納されます。混合カーソルまたは動的カーソルの場合、結果セットの最後のキーセットでなければ、*nrows* は -1 に設定されます。最後のキーセットの場合は、キーセット内のロー数が返されます。これによって、プログラマはキーセットが結果セットの終わりに達したことを認識できます。
- 参照先「付録 A カーソル」

参照

dbcursor、dbcursorbind、dbcursorclose、dbcursorcolinfo、dbcursorfetch、dbcursoropen

dbcursoropen

説明

カーソルをオープンして、スクロール・オプション、同時実行オプション、およびフェッチ・バッファのサイズ (1 回のフェッチで取り出すローの数) を指定します。

構文

```
DBCURSOR *dbcursoropen(dbproc, stmt, scrollopt,
                concuropt, nrows, pstatus)

DBPROCESS   *dbproc;
BYTE        *stmt;
```

```
SHORT       scrollopt;
SHORT       concuropt;
USHORT      nrows;
DBINT       *pstatus
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

stmt
カーソルを定義する select 文です。

scrollopt
使用するスクロール方法のインジケータです。

キーセット駆動型 ― 結果セットのメンバシップと順序を、カーソル・オープン時に固定します。

動的 ― 結果セットのメンバシップと順序を、フェッチ時に決定します。

表 2-11 に、*scrollopt* に対して可能な値をリストします。

表 2-11：scrollopt の値 (dbcursoropen)

記号値	意味
CUR_FORWARD	前方スクロールのみ。
CUR_KEYSET	キーセット駆動型。結果テーブルのキーセットのコピーがローカルに保持される。結果テーブルのローの数は 1,000 以下でなければならない。
CUR_DYNAMIC	完全に動的。
int *n*	(*n***nrows*) ブロック内ではキーセット駆動型であるが、キーセット外では完全に動的である。

concuropt
同時実行制御の定義です。表 2-12 に、*concuropt* に可能値をリストします。

表 2-12：concuropt の値 (dbcursoropen)

記号値	意味	説明
CUR_READONLY	読み込み専用	カーソル・データは変更できない。
CUR_LOCKCC	更新ロックに使用	トランザクション・ブロック内では、すべてのデータは、dbcursorfetch でフェッチされるときにロックされる。
CUR_OPTCC	タイムスタンプ値に基づくオプティミスティック同時実行制御	ローのデータの変更に成功するのは、そのローが最後のフェッチ以降に更新されていない場合だけである。変更が行われたことを検出するには、タイムスタンプを使用するか、選択されたテーブル・ロー内の text と image を除くすべての値を比較する。
CUR_OPTCCVAL	値に基づくオプティミスティック同時実行制御	選択されたすべてのカラムの値を比較して変更が検出される点を除いて、CUR_OPTCC と同じ。

nrows

フェッチ・バッファ内のローの数 (カーソルの幅) です。混合カーソルの場合は、この数に *scrollopt* パラメータの値を乗じたロー数がキーセットの容量となります。

pstatus

ロー・ステータス・インジケータの配列を指すポインタです。この配列には、フェッチ・バッファ内にコピーされた各ローのステータスが返されます。この配列は、バッファにフェッチされるローごとに 1 つの DBINT 整数を保管できる大きさにしてください。

dbcursorfetch を呼び出したときに、ローがバインドされた変数に格納されると、対応するステータスにはステータス情報が格納されます。このステータス情報を格納するために、dbcursorfetch はステータス値のビットを設定します。アプリケーションは 表 2-13 に示すビットマスク値を使用してステータス値を調べます。

表 2-13 : pstatus のビットマスク値 (dbcursoropen)

記号値	意味
FTC_SUCCEED	ローは正常にコピーされた。このフラグが設定されていない場合、ローはフェッチされていない。
FTC_MISSING	ローが見つからない。
FTC_ENDOFKEYSET	キーセットの終わり。バインド配列の残りのローは使用されない。
FTC_ENDOFRESULTS	結果セットの終わり。残りのローは使用されない。

戻り値

dbcursoropen が正常に終了すると、カーソルへのハンドルが返されます。後続のカーソル関数への呼び出しには、カーソル・ハンドルが必要です。

dbcursoropen が異常終了すると、NULL が返されます。次のようなエラーが発生すると、カーソルをオープンできません。

- システム内に十分なメモリがない。キーセット内のロー数を減らすか、動的スクロールを使用するか、または一度にフェッチするロー数を少なくします。
- *scrollopt* パラメータに CUR_KEYSET オプションが使用されているが、結果セットのロー数が 1,000 を超える。select 文で返されるロー数が 1,000 を超える可能性がある場合は、動的スクロールを使用します。
- ユニーク・ロー識別子を見つけることができない。

使用法

- この関数は、select 文の内容と *scrollopt*、*concuropt*、*nrows* の値に基づいて DB-Library の内部的なデータ構造を作成します。dbcursoropen は、カーソル結果セット内のローのユニーク識別子 (ロー・キー) に関する情報をサーバに問い合わせます。カーソルがキーセット駆動型の場合、dbcursoropen はサーバに問い合わせ、ロー・キーをフェッチしてカーソルのローのキーセットを構築します。
- カーソル定義にストアド・プロシージャまたは複数の Transact-SQL 文を含めることはできません。
- dbcursor が正常に機能するには、select 文内の各テーブルにユニーク・インデックスがなければなりません。Transact-SQL の for browse、select into、compute、union、または compute by は、カーソル文では使用できません。完全キーセット駆動型カーソルだけが order 句、having 句、または group by 句を使用できます。

- *stmt* として指定された select 文がテンポラリ・テーブルを参照する場合は、現在のデータベースが tempdb でなければなりません。この制限は、テンポラリ・テーブルが別のデータベース内に作成されている場合にも適用されます。
- 同一の *dbproc* 接続内で、複数のカーソルをオープンできます。オープンできるカーソルの数の上限は、システムのメモリによって決まります。カーソル関数を呼び出すときは、DBPROCESS 接続内に実行待ちのコマンドや保留中の結果がないようにしてください。
- 参照先「付録 A カーソル」

参照

dbcursor、dbcursorbind、dbcursorclose、dbcursorcolinfo、dbcursorfetch、dbcursorinfo、dbcursoropen

dbdata

説明

通常の結果カラム内のデータを指すポインタを返します。

構文

```
BYTE *dbdata(dbproc, column)

DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

対象となる特定のカラムのデータを指す BYTE ポインタです。このポインタを適切なデータ型にキャストするようにしてください。該当するカラムがない場合、またはデータが NULL 値の場合には、NULL BYTE ポインタが返されます。データが NULL 値であることを確認するために、dbdatlen からの戻り値が 0 であるかどうかを常に調べてください。

使用法

- このルーチンは、通常 (非計算) 結果カラムのデータを指すポインタを返します。このデータは、NULL で終了するデータではありません。データ長を調べるには、dbdatlen を使用してください。

- 次のコードは、dbdata の例です。

```
DBPROCESS    *dbproc;
DBINT        row_number = 0;
DBINT        object_id;

/* Put the command into the command buffer */
 dbcmd(dbproc, "select id from sysobjects");
 /*
 ** Send the command to Adaptive Server Enterprise
and begin
 ** execution
 */
 dbsqlexec(dbproc);
 /* Process the command results */
 dbresults(dbproc);
 /* Examine the data in each row */
 while (dbnextrow(dbproc) != NO_MORE_ROWS)
 {
      row_number++;
      object_id = *((DBINT *)dbdata(dbproc, 1));
      printf("row %ld, object id is %ld.¥n",
           row_number, object_id);
 }
```

- 文字列データに NULL ターミネータを追加する場合は、strncpy などのルーチンを使用して DBPROCESS からデータをコピーしてから行ってください。次に例を示します。

```
char    objname[40];
      ...
strncpy(objname, (char *)dbdata(dbproc,2),
      (int)dbdatlen(dbproc,2));
objname[dbdatlen(dbproc,2)] = '¥0';
```

- 関数 dbbind は、結果データを自動的にプログラム変数にバインドします。このルーチンはデータをコピーしますが、一般に dbdata よりも簡単に使用できます。さらに、便利なデータ型変換機能があります。アプリケーションは、この機能を使用して、結果文字列に NULL ターミネータを追加したり、通貨や日時のデータを印刷可能文字列に変換したりすることができます。

参照　dbbind、dbcollen、dbcolname、dbcoltype、dbdatlen、dbnumcols

dbdate4cmp

説明

2 つの DBDATETIME4 値を比較します。

構文

int dbdate4cmp(dbproc, d1, d2)

```
DBPROCESS     *dbproc;
DBDATETIME4   *d1;
DBDATETIME4   *d2;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。

d1

DBDATETIME4 値を指すポインタです。

d2

DBDATETIME4 値を指すポインタです。

戻り値

d1 = *d2* の場合、dbdate4cmp は 0 を返します。

d1 < *d2* の場合、dbdate4cmp は -1 を返します。

d1 > *d2* の場合、dbdate4cmp は 1 を返します。

使用法

- dbdate4cmp は、2 つの DBDATETIME4 値を比較します。
- 有効な DBDATETIME4 値の範囲は、1900 年 1 月 1 日から 2079 年 6 月 6 日までです。DBDATETIME4 値の精度は 1 分です。

参照

dbdatecmp、dbmnycmp、dbmny4cmp

dbdate4zero

説明

DBDATETIME4 変数を 1900 年 1 月 1 日午前 0 時 00 分に初期化します。

構文

RETCODE dbdate4zero(dbproc, dateptr)

```
DBPROCESS     *dbproc;
DBDATETIME4   *dateptr;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。

dateptr
初期化する DBDATETIME4 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dateptr が NULL の場合、dbdate4zero は FAIL を返します。

使用法

- dbdate4zero は、DBDATETIME4 変数を 1900 年 1 月 1 日午前 0 時 00 分に初期化します。
- 有効な DBDATETIME4 値の範囲は、1900 年 1 月 1 日から 2079 年 6 月 6 日までです。DBDATETIME4 値の精度は 1 分です。

参照

dbdatezero

dbdatechar

説明

DBDATETIME 値の要素を表す整数を文字フォーマットに変換します。

構文

RETCODE dbdatechar(dbproc, charbuf, datepart, value)

```
DBPROCESS    *dbproc;
char         *charbuf;
int          datepart;
int          value;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

charbuf
value の文字表現 (NULL ターミネータ付き) が格納される文字バッファを指すポインタです。

datepart

value のタイプを表す記号定数です。表 2-14 に、日付を構成する各部分、DB-Library が認識する日付構成部分の記号値と、取り得る値の範囲を示します。この表の月と曜日の名前は英語名です。

表 2-14：日付の各部分とその文字表現 (dbdatechar)

日付要素	記号	値の文字表現
年	DBDATE_YY	1753 － 9999
期	DBDATE_QQ	1 － 4
月	DBDATE_MM	January － December
年間通算日	DBDATE_DY	1 － 366
日	DBDATE_DD	1 － 31
週	DBDATE_WK	1 － 54 (うるう年の場合)
曜日	DBDATE_DW	Monday － Sunday
時刻	DBDATE_HH	0 － 23
分	DBDATE_MI	0 － 59
秒	DBDATE_SS	0 － 59
ミリ秒	DBDATE_MS	0 － 999

value

変換される数値です。

戻り値

SUCCEED または FAIL。

使用法

- dbdatechar は、日時の構成要素を表す整数を文字フォーマットに変換します。たとえば、dbdatechar は月の要素の「3」を該当する文字列に対応させます。英語が使用されている場合は「March」、フランス語が使用されている場合は「mars」となります。
- 対応する文字列の言語は、*dbproc* によって決まります。
- dbdatechar は、一般に dbdatecrack と一緒に使用すると便利です。

参照

dbconvert、dbdata、dbdatename、dbdatecrack

dbdatecmp

説明

2 つの DBDATETIME 値を比較します。

構文

int dbdatecmp(dbproc, d1, d2)

```
DBPROCESS    *dbproc;
DBDATETIME   *d1;
DBDATETIME   *d2;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。

d1
DBDATETIME 値を指すポインタです。

d2
DBDATETIME 値を指すポインタです。

戻り値

d1 = *d2* の場合、dbdatecmp は 0 を返します。

d1 < *d2* の場合、dbdatecmp は -1 を返します。

d1 > *d2* の場合、dbdatecmp は 1 を返します。

使用法

- dbdatecmp は、2 つの DBDATETIME 値を比較します。
- 有効な DBDATETIME 値の範囲は、1753 年 1 月 1 日から 9999 年 12 月 31 日までです。DBDATETIME 値の精度は、300 分の 1 秒 (3.33 ミリ秒) です。

参照

dbdate4cmp、dbmnycmp、dbmny4cmp

dbdatecrack

説明

マシン読み込み可能な DBDATETIME 値を、ユーザがアクセスできるフォーマットに変換します。

構文

```
RETCODE dbdatecrack(dbproc, dateinfo, datetime)

DBPROCESS     *dbproc;
DBDATEREC     *dateinfo;
DBDATETIME    *datetime;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

dateinfo

datetime の各部分が格納される DBDATEREC 構造体を指すポインタです。DBDATEREC は次のように定義されています。

```
typedef struct dbdaterec
{
    long    dateyear;    /* 1900 to the future */
    long    datemonth;   /* 0 - 11 */
    long    datedmonth;  /* 1 - 31 */
    long    datedyear;   /* 1 - 366 */
    long    datedweek;   /* 0 - 6  */
    long    datehour;    /* 0 - 23 */
    long    dateminute;  /* 0 - 59 */
    long    datesecond;  /* 0 - 59 */
    long    datemsecond; /* 0 - 997 */
    long    datetzone;   /* 0 - 127 */
} DBDATEREC;
```

月と曜日の名前は DBPROCESS の各国言語によって決まります。これらの名前を取得するには、dbdatename または dbdayname と dbmonthname を使用します。

注意 *dateinfo->datetzone* フィールドは dbdatecrack では設定されません。

datetime

対象となる DBDATETIME 値を指すポインタです。

戻り値

SUCCEED または FAIL。

使用法

- dbdatecrack は、DBDATETIME 値の各部分を整数に変換したものを DBDATEREC 構造体に格納します。
- DBDATETIME 構造体には、日時の値が内部フォーマットで保管されます。たとえば、時刻値は午前 0 時からの 300 分の 1 秒単位の数で保管され、日付値は 1900 年 1 月 1 日からの通算日数で保管されます。dbdatecrack は、この内部値をアプリケーション・プログラムで使用しやすいように変換します。
- DBDATEREC 構造体に格納される整数の日付部分を文字列に変換するには、dbdatechar を使用します。
- dbdatecrack を呼び出して内部フォーマットの日時値を一度に変換するのは、dbdatepart を何回も呼び出すのと同じことです。
- 次のコードは、dbdatecrack の例です。

```
dbcmd(dbproc, "select name, crdate from ¥
     master..sysdatabases");
```

```
dbsqlexec(dbproc);
dbresults(dbproc);
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
     /*
     ** Print the database name and its date info
     */
     dbconvert(dbproc, dbcoltype(dbproc, 2),
          dbdata(dbproc, 2), dbdatlen(dbproc, 2),
          SYBCHAR, datestring, -1);
     printf("%s:%s¥n", (char *)
          (dbdata(dbproc, 1)), datestring);
     /*
     ** Break up the creation date into its
     ** constituent parts.
     */
     dbdatecrack(dbproc, &dateinfo,
          (DBDATETIME *) (dbdata(dbproc, 2)));

   /* Print the parts of the creation date */
     printf("¥tYear = &d.¥n", dateinfo.dateyear);
    printf("¥tMonth = &d.¥n", dateinfo.datemonth);
     printf("¥tDay of month = &d.¥n",
          dateinfo.datedmonth);
     printf("¥tDay of year = &d.¥n",
          dateinfo.datedyear);
     printf("¥tDay of week = &d.¥n",
          dateinfo.datedweek);
     printf("¥tHour = &d.¥n", dateinfo.datehour);
     printf("¥tMinute = &d.¥n",
          dateinfo.dateminute);
     printf("¥tSecond = &d.¥n",
          dateinfo.datesecond);
     printf("¥tMillisecond = &d.¥n",
          dateinfo.datemsecond);
}
```

参照　dbconvert、dbdata、dbdatechar、dbdatename、dbdatepart

dbdatename

説明

DBDATETIME 構造体の指定の構成要素を、対応する文字列に変換します。

構文

int dbdatename(dbproc, charbuf, datepart, datetime)

```
DBPROCESS    *dbproc;
char         *charbuf;
int          datepart;
DBDATETIME   *datetime;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

charbuf

datetime を構成する要素の、null ターミネータ付きの文字表現が格納される文字バッファを指すポインタです。*datetime* が NULL の場合は、*charbuf* には長さゼロの文字列が格納されます。

datepart

対象となる日付の要素です。表 2-15 に、日付を構成する各部分、DB-Library が認識する日付構成部分の記号値、取り得る値の範囲を示します。この表の月と曜日の名前は英語名です。

表 2-15：日付の各部分とその文字表現 (dbdatename)

日付要素	記号	値の文字表現
年	DBDATE_YY	1753 － 9999
期	DBDATE_QQ	1 － 4
月	DBDATE_MM	January － December
年間通算日	DBDATE_DY	1 － 366
日	DBDATE_DD	1 － 31
週	DBDATE_WK	1 － 54 (うるう年の場合)
曜日	DBDATE_DW	Monday － Sunday
時刻	DBDATE_HH	0 － 23
蛻	DBDATE_MI	0 － 59
分	DBDATE_SS	0 － 59
ミリ秒	DBDATE_MS	0 － 999

datetime

対象となる DBDATETIME 値を指すポインタです。

戻り値

**charbuf* に格納されたバイト数です。

エラーの場合、dbdatename は -1 を返します。

使用法

- dbdatename は、DBDATETIME 構造体の指定の要素を文字列に変換します。
- 月と曜日の名前は、指定された DBPROCESS の言語での名前です。*dbproc* が NULL の場合、これらの名前は DB-Library のデフォルト言語での名前となります。
- この関数は、Transact-SQL の datename 関数によく似ています。
- 次のコードは、dbdatetime の例です。

```
dbcmd(dbproc, "select name, crdate from ¥
     master..sysdatabases");
dbsqlexec(dbproc);
dbresults(dbproc);

while (dbnextrow(dbproc) != NO_MORE_ROWS)
 {
      /*
      ** Print the database name and its date info
      */
      dbconvert(dbproc, dbcoltype(dbproc, 2),
           dbdata(dbproc, 2), dbdatlen(dbproc, 2),
           SYBCHAR, datestring, -1);
      printf("%s:%s¥n", (char *) (dbdata
           (dbproc, 1)), datestring);

      /* Print the parts of the creation date */
      dbdatename(dbproc, datestring, DBDATE_YY,
           (DBDATETIME *) (dbdata(dbproc, 2)));
      printf("¥tYear = %s.¥n", datestring);

      dbdatename(dbproc, datestring, DBDATE_QQ,
           (DBDATETIME *) (dbdata(dbproc, 2)));
      printf("¥tQuarter = %s.¥n", datestring);

      dbdatename(dbproc, datestring, DBDATE_MM,
           (DBDATETIME *) (dbdata(dbproc, 2)));
      printf("¥tMonth = %s.¥n", datestring);

      dbdatename(dbproc, datestring, DBDATE_DW,
           (DBDATETIME *) (dbdata(dbproc, 2)));
      printf("¥tDay of week = %s.¥n", datestring);
```

```
dbdatename(dbproc, datestring, DBDATE_DD,
     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tDay of month = %s.¥n", datestring);

dbdatename(dbproc, datestring, DBDATE_DY,

     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tDay of year = %s.¥n", datestring);

dbdatename(dbproc, datestring, DBDATE_HH,
     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tHour = %s.¥n", datestring);

dbdatename(dbproc, datestring, DBDATE_MI,
     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tMinute = %s.¥n", datestring);

dbdatename(dbproc, datestring, DBDATE_SS,
     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tSecond = %s.¥n", datestring);

dbdatename(dbproc, datestring, DBDATE_MS,
     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tMillisecond = %s.¥n", datestring);

dbdatename(dbproc, datestring, DBDATE_WK,
     (DBDATETIME *) (dbdata(dbproc, 2)));
printf("¥tWeek = %s.¥n", datestring);
```

参照 dbconvert、dbdata、dbdatechar、dbdatecrack

dbdateorder

説明 指定の言語の日付要素の順序を返します。

構文 char *dbdateorder(dbproc, language)

```
DBPROCESS    *dbproc;
char         *language;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

language
　言語の名前です。

戻り値

月、日、年を表す「m」、「d」、「y」という 3 つの文字で構成される、null で終了する文字列を指すポインタです。dbdateorder 文字列の文字の順序は、*language* のデフォルト日時フォーマットでの順序に対応しています。

dbdateorder は、失敗すると NULL ポインタを返します。

使用法

- dbdateorder は、指定された言語での月、日、年の日付要素の順序を表す文字列を返します。*language* が NULL の場合は、指定した DBPROCESS の現在の言語が使用されます。*language* と *dbproc* の両方が NULL の場合は、DB-Library のデフォルト言語が使用されます。

  警告！ dbdateorder から返される日付順序の文字列は、DB-Library の内部データ構造体を指すポインタです。アプリケーション・プログラムでこの文字列を変更したり、解放したりしないでください。

- 次のコードは、dbdateorder の例です。

```
/* Retrieve the date order from Adaptive Server
Enterprise */
 printf("date-order:%s¥n",
     (dbdateorder(DBPROCESS *)NULL, (char *)NULL));
```

参照

dbconvert、dbdata、dbdatechar、dbdatecrack

dbdatepart

説明

DBDATETIME 値の指定された部分を数値で返します。

構文

DBINT dbdatepart(dbproc, datepart, datetime)

DBPROCESS　*dbproc;

```
int             datepart;
DBDATETIME      *datetime;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

datepart

対象となる日付の要素です。表 2-16 に、日付を構成する各部分、DB-Library が認識する日付構成部分の記号値、取り得る値の範囲を示します。この表の月と曜日の名前は英語名です。

表 2-16：日付の各部分とその文字表現 (dbdatepart)

日付要素	記号	値の文字表現
年	DBDATE_YY	1753 － 9999
期	DBDATE_QQ	1 － 4
月	DBDATE_MM	January ～ December
年間通算日	DBDATE_DY	1 － 366
日	DBDATE_DD	1 － 31
週	DBDATE_WK	1 － 54 (うるう年の場合)
曜日	DBDATE_DW	Monday － Sunday
時刻	DBDATE_HH	0 － 23
蛻	DBDATE_MI	0 － 59
分	DBDATE_SS	0 － 59
ミリ秒	DBDATE_MS	0 － 999

datetime

対象となる DBDATETIME 値を指すポインタです。

戻り値

指定の日付部分の値です。

使用法

- dbdatepart は、DBDATETIME 値の指定された部分を数値で返します。
- dbdatepart は、Transact-SQL の datepart 関数に似ています。

参照

dbconvert、dbdata、dbdatechar、dbdatecrack、dbdatename

dbdatezero

説明

DBDATETIME 値を 1900 年 1 月 1 日午前 0 時 00 分に初期化します。

構文

RETCODE dbdatezero(dbproc, dateptr)

```
DBPROCESS     *dbproc;
DBDATETIME    *dateptr;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。

dateptr
初期化する DBDATETIME 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dateptr が NULL の場合、dbdatezero は FAIL を返します。

使用法

- dbdatezero は、DBDATETIME 値を 1900 年 1 月 1 日午前 0 時 00 分に初期化します。
- 有効な DBDATETIME 値の範囲は、1753 年 1 月 1 日から 9999 年 12 月 31 日までです。DBDATETIME 値の精度は、300 分の 1 秒 (3.33 ミリ秒) です。

参照

dbdate4zero

dbdatlen

説明

通常の結果カラム内のデータの長さを返します。

構文

DBINT dbdatlen(dbproc, column)

```
DBPROCESS     *dbproc;
int           column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
対象となるカラムの番号です。最初のカラムの番号は 1 です。

戻り値

特定のカラムに返されるデータのバイト単位の長さです。データの値が NULL の場合、dbdatlen は 0 を返します。カラム番号が範囲外の場合、dbdatlen は -1 を返します。

使用法

- このルーチンは、select によって返される通常 (非計算) 結果カラムのデータの長さをバイト単位で返します。ほとんどの場合、これはそのカラムの実際のデータの長さです。ただし、text カラムと image カラムの場合は、dbdatlen が返す整数がそのカラムのデータの実際の長さよりも小さいことがあります。これは、select から返される text または image のデータの大きさが、サーバ・グローバル変数 *@@textsize* によって制限されるためです。
- データが取り得る最大長を調べるには、dbcollen ルーチンを使用してください。データ自体を指すポインタを取得するには、dbdata を使用してください。
- 次のコードは、dbdatlen の例です。

```
DBPROCESS    *dbproc;
DBINT        row_number = 0;
DBINT        data_length;

* Put the command into the command buffer */
dbcmd(dbproc, "select name from sysobjects");
/*
** Send the command to Adaptive Server Enterprise
and begin
** execution
*/
dbsqlexec(dbproc);

/* Process the command results */
dbresults(dbproc);

/* Examine the data lengths of each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
     row_number++;
     data_length = dbdatlen(dbproc, 1);
     printf("row %ld, data length is %ld.¥n",
          row_number, data_length);
}
```

参照

dbcollen、dbcolname、dbcoltype、dbdata、dbnumcols

dbdayname

説明

指定された曜日の、指定された言語での名前を調べます。

構文

char *dbdayname(dbproc, language, daynum)

```
DBPROCESS    *dbproc;
char         *language;
int          daynum;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

language
　要求する言語の名前です。

daynum
　要求する曜日の番号です。曜日の番号の範囲は 1 (月曜日) から 7 (日曜日) までです。

戻り値

正常に終了した場合は、指定した曜日の名前です。エラーの場合は、NULL ポインタです。

使用法

- dbdayname は、指定の曜日の、指定の言語での名前を返します。*language* が NULL の場合は、*dbproc* の現在の言語が使用されます。*language* と *dbproc* の両方が NULL の場合は、アメリカ英語が使用されます。
- 次のコードは、dbdayname の例です。

```
/*
** Retrieve the name of each day of the week in
** U.S. English.
*/
for (daynum = 1; daynum <= 7; daynum++)
     printf("Day %d:%s¥n", daynum,
     dbdayname((DBPROCESS *)NULL, (char *)NULL,
     daynum));
```

参照

db12hour、dbdateorder、dbmonthname、DBSETLNATLANG

DBDEAD

説明　特定の DBPROCESS が dead であるかどうかを判断します。

構文　DBBOOL DBDEAD(dbproc)

```
DBPROCESS    *dbproc;
```

パラメータ　dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　「TRUE」または「FALSE」。

使用法

- このマクロは、指定された DBPROCESS のステータスが dead であるかどうかを示します。これは、ユーザ提供のエラー・ハンドラで特に便利です。
- DBPROCESS のステータスが dead のとき、DBPROCESS をパラメータとして受け取る DB-Library ルーチンのほとんどはただちに異常終了し、ユーザ提供のエラー・ハンドラが呼び出されます。

  注意　ユーザ提供のエラー・ハンドラがない場合、DBPROCESS が dead になると、影響を受ける DB-Library ルーチンが異常終了するのではなく、プログラム自体がアボートします。

- DBDEAD はサーバとは通信せず、DBPROCESS の現在のステータスだけを調べます。それまでに呼び出された DB-Library ルーチンによって DBPROCESS のステータスが dead と設定されていない場合は、DBDEAD を実行すると、DBPROCESS は正常であるとして報告されます。

参照　dberrhandle、「エラー」(418 ページ)

dberrhandle

説明　DB-Library エラーを処理するユーザ関数をインストールします。

構文　int (*dberrhandle(handler))()

```
int         (*handler)();
```

パラメータ

ハンドラ

DB-Library がエラーの発生を認識したときに呼び出されるユーザ関数を指すポインタです。DB-Library は、表 2-17 に示す 6 つのパラメータを指定してこの関数を呼び出します。

表 2-17：エラー・ハンドラ・パラメータ

パラメータ	意味
dbproc	影響を受ける DBPROCESS。このエラーに関係する DBPROCESS がない場合、このパラメータは NULL になる。
severity	エラーの重大度 (データ型 *int*)。エラーの重大度は *syberror.h* で定義されている。
dberr	エラーの識別番号 (データ型 *int*)。エラー番号は *sybdb.h* で定義されている。
oserr	エラーの原因を表す、オペレーティング・システム固有のエラー番号 (データ型 int)。該当するオペレーティング・システム・エラーがない場合、このパラメータの値は DBNOERR になる。
dberrstr	*dberr* の印刷可能な説明 (データ型 char *)。
oserrstr	*oserr* の印刷可能な説明 (データ型 char *)。

エラー・ハンドラは、表 2-18 でリストする 4 つの値のいずれかを返し、DB-Library に特定のアクションを実行するよう指示します。

表 2-18：エラー・ハンドラ戻り値

戻り値	対処法
INT_EXIT	エラー・メッセージを出力して、プログラムをアボートする。また、オペレーティング・システムにエラー表示を返す (UNIX プログラマの方へ：DB-Library はコア・ファイルを残しません)。
INT_CANCEL	エラーを起こした DB-Library ルーチンから FAIL を返す。タイムアウト・エラーのときに INT_CANCEL を返すと、*dbproc* は強制終了する。
INT_TIMEOUT	エラーを引き起こしたオペレーションをキャンセルするが、*dbproc* は機能している状態のままにする。この戻り値が意味を持つのは、タイムアウト・エラー (SYBETIME) の場合だけである。その他の場合、この値はエラーとみなされ、INT_EXIT として扱われる。
INT_CONTINUE	再びタイムアウト期間が経過するまで待機する。その期間が経過した時点で、再度エラー・ハンドラを呼び出す。この戻り値が意味を持つのは、タイムアウト・エラー (SYBETIME) の場合だけである。その他の場合、この値はエラーとみなされ、INT_EXIT として扱われる。

エラー・ハンドラが上記の 4 つの値以外の値を返すと、プログラムはアボートします。

Windows プラットフォームのエラー・ハンドラは、次の例で示すように CS_PUBLIC で宣言してください。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。

次の例は、標準的なエラー・ハンドラ・ルーチンを示しています。

```
#include   <sybfront.h>
#include   <sybdb.h>
#include   <syberror.h>

int CS_PUBLIC err_handler(dbproc, severity, dberr,
oserr, dberrstr, oserrstr)
DBPROCESS     *dbproc;
 int          severity;
 int          dberr;
 int          oserr;
 char         *dberrstr;
 char         *oserrstr;
{
      if ((dbproc == NULL) || (DBDEAD(dbproc)))
           return (INT_EXIT);
      else
```

```
        {
            printf("DB-Library error:¥n¥t%s¥n",
                dberrstr);
            if (oserr != DBNOERR)
                printf("Operating-system ¥
                    error:¥n¥t%s¥n", oserrstr);
            return(INT_CANCEL);
        }
    }
```

戻り値

インストール済みのエラー・ハンドラを指すポインタです。エラー・ハンドラがインストールされていない場合、このポインタは NULL です。

使用法

- dberrhandle は、ユーザ提供のエラー・ハンドラ関数をインストールします。
  DB-Library エラーが発生すると、DB-Library はただちにこのエラー・ハンドラを呼び出します。DB-Library エラーを適切に処理するには、ユーザ提供のエラー・ハンドラをインストールしてください。
- アプリケーションで dberrhandle を呼び出してメッセージ・ハンドラ関数をインストールしていない場合は、DB-Library はエラー・メッセージを無視します。メッセージは出力されません。
- 発生したエラーに対する DB-Library の応答は、ユーザ提供のエラー・ハンドラによって決まります。ユーザ提供のエラー・ハンドラは、次のどの処置を取るかを DB-Library に指示する必要があります。
  - プログラムをアボートする
  - エラー・コードを返し、DBPROCESS のステータスを「dead」(使用不可能) にする
  - エラーを引き起こしたオペレーションを取り消す
  - リトライを続ける (タイムアウト・エラーの場合)
- ユーザ提供のエラー・ハンドラがない (または dberrhandle に NULL ポインタが渡された) 場合は、DB-Library はデフォルトのエラー処理を行います。エラーによって、影響を受けた DBPROCESS が使用不可能になった場合は、プログラムをアボートします (DBPROCESS が使用不可能かどうかを調べるには DBDEAD を呼び出します)。エラーが発生しても DBPROCESS は使用不可能になっていない場合は、DB-Library は呼び出し元のプログラムにエラー・コードを返します。

- NULL パラメータを指定して dberrhandle を呼び出すことによって、既存のエラー・ハンドラのインストールを「解除」できます。また、いつでも新しいエラー・ハンドラをインストールできます。新しいメッセージ・ハンドラは、自動的に既存のメッセージ・ハンドラと置き換わります。
- プログラムでエラー重大度の値を参照する場合は、そのソース・ファイルでヘッダ・ファイル *syberror.h* をインクルードする必要があります。
- DB-Library エラーのリストについては、「エラー」(418 ページ) を参照してください。
- もう 1 つのルーチンである dbmsghandle は、DB-Library がサーバのエラー・メッセージに応答して呼び出すメッセージ・ハンドラをインストールします。
- アプリケーションが DB-Library からのメッセージとサーバからのメッセージを同時に発生させた場合、DB-Library は、DB-Library エラー・ハンドラを呼び出す前にサーバ・メッセージ・ハンドラを呼び出します。
- DB-Library/C のエラー値 SYBESMSG は、サーバ・エラー・メッセージに応答して生成されますが、サーバ情報メッセージに応答して生成されることはありません。したがって、サーバ・エラーが発生すると、サーバ・メッセージ・ハンドラと DB-Library/C エラー・ハンドラの両方が呼び出されますが、サーバが情報メッセージを生成したときには、サーバ・メッセージ・ハンドラだけが呼び出されます。

  サーバ・メッセージ・ハンドラをインストールした場合は、同じエラーが二度ユーザに通知されないようにするために、DB-Library エラー・ハンドラでの SYBESMSG エラー出力の抑制もできます。

  表 2-19 は、DB-Library/C がいつアプリケーションのメッセージ・ハンドラとエラー・ハンドラを呼び出すかをまとめたものです。

表 2-19：一般的なエラー

エラーまたはメッセージ	メッセージ・ハンドラを呼び出す	エラー・ハンドラを呼び出す
SQL 構文エラー。	はい。	はい (SYBESMSG)。 (メッセージを無視するようにハンドラをコーディングする。)
SQL print 文。	はい。	いいえ。
SQL raiserror。	はい。	いいえ。

エラーまたはメッセージ	メッセージ・ハンドラを呼び出す	エラー・ハンドラを呼び出す
サーバが停止している。	いいえ。	はい (SYBESEOF)。 (アプリケーションを終了するようにハンドラをコーディングする。)
サーバからのタイムアウト。 **注意**　デフォルトのタイムアウト期間は無限に設定されています。dbsettime でタイムアウト期間が設定されていなければ、エラー・ハンドラはタイムアウトの通知を受け取りません。	いいえ。	はい (SYBETIME)。 (再びタイムアウト期間が経過するまで待つために、INT_CONTINUE を返すようにハンドラをコーディングする。)
クエリ時のデッドロック。	はい。 (デッドロックの有無を調べるようにハンドラをコーディングする。「dbsetuserdata」(361 ページ) の例を参照。)	はい (SYBESMSG)。 (メッセージを無視するようにハンドラをコーディングする。)
ログイン時のタイムアウト。	いいえ。	はい (SYBEFCON、SYBECONN)。
ログインの失敗 (dbopen)。	はい。	はい (SYBEPWD)。 (アプリケーションを終了するようにハンドラをコーディングする。)
データベース・メッセージの使用。	はい。 (メッセージを無視するようにハンドラをコーディングする。)	いいえ。
必要なときに dbresults を呼び出さないなど、DB-Library/C 呼び出しの使用法の誤り。	いいえ。	はい (SYBERPND、...)。 はい (SYBERPND)。
致命的なサーバ・エラー (重大度 17 以上)。	はい (アプリケーションを終了するようにハンドラをコーディングする。)	はい (SYBESMSG)。

参照　DBDEAD、dbmsghandle、「エラー」(418 ページ)

dbexit

説明

すべての DBPROCESS 構造体をクローズして割り付けを解除し、dbinit で初期化した構造体の終了処理を行います。

構文

void dbexit()

戻り値

なし。

使用法

- dbexit は、割り付けられているすべての DBPROCESS 構造体に対して dbclose を繰り返し呼び出します。dbclose は、単一の DBPROCESS 構造体に関連付けられているすべてのアクティビティをクリーンアップし、領域の割り付けを解除します。
- 1 つの DBPROCESS 構造体だけをクローズする場合には、直接 dbclose を使用できます。
- dbexit は、dbinit によって初期化されたすべての構造体のクリーンアップも行い、構造体に関連付けられていたメモリを解放します。このルーチンは、dbinit を呼び出すアプリケーションの最後の DB-Library 呼び出しにしてください。
- 今後の互換性と移植性を確実にするために、オペレーティング環境に関わらず、すべてのアプリケーションで dbinit と dbexit を呼び出すことを強くおすすめします。

  dbinit を必要とする環境では、dbexit を呼び出した後は、他の DB-Library 呼び出しを行わないでください。

参照

dbclose、dbinit、dbopen

dbfcmd

説明

C ランタイム・ライブラリ sprintf 型のフォーマットを使用して、テキストを DBPROCESS コマンド・バッファに追加します。

構文

RETCODE dbfcmd(dbproc, cmdstring, args...)

```
DBPROCESS    *dbproc;
char         *cmdstring;
???          args...;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

cmdstring

sprintf ルーチンによって使用される形式のフォーマット文字列です。

dbfcmd には、オプションで不定数の引数があります。必要な引数の数と型は、*cmdstring* 引数に含まれるフォーマット指定子によって決まります。引数は C-library sprintf 関数に直接渡されます。dbfcmd と C コンパイラのどちらも、この引数の型を検査することはできません。sprintf を使用する場合と同様に、引数の型が対応するフォーマット指定子と一致することをプログラマが確認する必要があります。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、DBPROCESS 構造体内の Transact-SQL コマンド・バッファにテキストを追加します。dbfcmd は C 言語の標準 I/O ライブラリの sprintf 関数と同様に機能し、% 変換指定子を使用できます。sprintf のフォーマット機能が必要ない場合は、代わりに dbcmd を使用できます。
- 表 2-20 は、dbfcmd がサポートする変換のリストです。

表 2-20：dbfcmd 変換

変換	プログラム変数タイプ
%s	char*、NULL で終了する
%d	int、10 進表現
%f	double
%g	double
%e	double
%%	なし。「%」という文字がコマンド・バッファに書き込まれる

データ型 SYBDATETIME は、%s を使用して、文字列に変換して渡す必要があります。データ型 SYBMONEY は、%s を使用して文字列に変換して渡すことも、%f を使用して浮動小数点数に変換して渡すこともできます。

注意 現時点では、dbfcmd の 1 回の呼び出しで処理できる引数は 8 つまでです。9 つ以上の引数を必要とするコマンドをフォーマットするには、dbfcmd を繰り返し呼び出してください。

- dbfcmd は、コマンド・バッファの領域割り付けを管理します。テキストは既存のコマンド・バッファに追加されます。バッファがサーバに送信された後を除いて、バッファの現在の内容が削除または上書きされることはありません (「コマンド・バッファのクリア」(159 ページ) を参照)。1 つのコマンド・バッファに複数のコマンドが含まれることもありますが、このようにするのがコマンド・バッファの効率的な使い方です。
- アプリケーションで、dbfcmd を繰り返し呼び出すことができます。連続して呼び出すと、コマンド文字列はそのまま連結されます。文字列の終わりと次の文字列の始まりとの間にブランクが必要な場合に、そのブランクを挿入するのは、アプリケーション側で行います。
- 次に、dbfcmd を使用して複数行の SQL コマンドを構築する例を示します。

```
char         *column_name;
DBPROCESS    *dbproc;
int          low_id;
char         *object_type;
char         *tablename;
dbfcmd(dbproc, "select %s from %s", column_name,
     tablename);
dbfcmd(dbproc, " where id > %d", low_id);
dbfcmd(dbproc, " and type='%s", object_type);
```

2 番目と 3 番目のコマンド文字列の前に空白が必要な点に注意してください。

- 文字変数または文字列変数を dbfcmd に渡すときは、引用符 (一重または二重) または null 文字 (ASCII 0) を含む変数に注意してください。
  - SQL コマンド内の引用符の使用が不適切なときは、SQL 構文エラーが発生したり、予期したものとは異なる結果が返されたりすることがあります。

    - NULL 文字 (ASCII 0) をコマンド・バッファに挿入しないでください。NULL 文字は、DB-Library とサーバの処理の混乱の原因となり、SQL 構文エラーが発生したり、予期したものとは異なる結果が返されたりすることがあります。

- dbfcmd は sprintf を呼び出すので、% (パーセント記号) は、フォーマット・コマンドの始まりとしての特別な意味を持つことに注意してください。コマンド文字列に % が含まれる場合は、その前にもう 1 つ % を付ける必要があります。
- dbfcmd の文字列パラメータとして NULL ポインタを渡さないように注意してください。NULL 値の可能性がある場合は、その変数を dbfcmd 呼び出しで使用する前に、NULL 値であるかどうかを確認する必要があります。
- 1 つのアプリケーション内に、dbcmd と dbfcmd の呼び出しが混在していてもかまいません。
- アプリケーションは、いつでも dbgetchar、dbstrlen、dbstrcpy を呼び出してコマンド・バッファの内容にアクセスできます。
- dbcmd と dbfcmd の呼び出しで作成される DBPROCESS コマンド・バッファのサイズは、使用可能なメモリ容量以外の制約は受けません。

コマンド・バッファのクリア

dbsqlexec または dbsqlsend を呼び出した後の dbcmd または dbfcmd の最初の呼び出し時に、新しいテキストが入力される前にコマンド・バッファが自動的にクリアされます。これを行わない場合は、DBNOAUTOFREE オプションを設定してください。DBNOAUTOFREE が設定されているときは、dbfreebuf を明示的に呼び出した場合にのみコマンド・バッファがクリアされます。

制限事項

現時点では、dbfcmd の 1 回の呼び出しで処理できる *args* は 8 つまでです。9 つ以上の *args* を必要とするコマンドをフォーマットするには、dbfcmd を繰り返し呼び出してください。プラットフォームによっては、1 回の呼び出しで 9 つ以上の *args* を処理できる dbfcmd もあります。コードを移植可能にするには、渡す引数は 8 つ以内としてください。

dbfcmd はテキスト代入を行うので、DBPROCESS コマンド・バッファの他に作業バッファを使用します。dbfcmd は、この作業バッファを動的に割り付けます。割り付けられる領域のサイズは、定義されている定数 (1024) と *cmdstring* の文字列長の 2 倍のいずれか大きい方です。たとえば、*cmdstring* の長さが 600 バイトの場合、dbfcmd は 1200 バイトの作業バッファを割り当てます。*cmdstring* の長さが 34 バイトの場合、dbfcmd は 1024 バイトの作業バッファを割り当てます。この制限に対処するには、次のようにします。

```
sprintf (buffer, "%s", SQL commmand");
dbcmd (dbproc, buffer)
```

args が、*cmdstring* のサイズに比べて非常に大きい場合は、作業バッファが小さすぎて代入後の文字列を保持できないことがあります。このような場合は、*cmdstring* を分割して dbfcmd を複数回呼び出してください。

作業バッファは、DBPROCESS コマンド・バッファとは別のものであることに注意してください。作業バッファは、テキスト代入を行うときに dbfcmd だけが使用できるテンポラリのバッファです。代入が行われた後は、テキストは DBPROCESS コマンド・バッファに保持されます。DBPROCESS コマンド・バッファのサイズには、使用可能なメモリ以外の制約はありません。

参照

dbcmd、dbfreebuf、dbgetchar、dbstrcpy、dbstrlen、「オプション」(436 ページ)

DBFIRSTROW

説明

ロー・バッファ内の最初のローの番号を返します。

構文

```
DBINT DBFIRSTROW(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

ロー・バッファ内の最初のローの番号です。ローの番号は、サーバから最初に返されたローを 1 として、順にカウントされます。エラーの場合は、0 を返します。

使用法

- このマクロは、ロー・バッファ内の最初のローの番号を返します。
- ローをバッファリングしていない場合は、DBFIRSTROW、DBCURROW、DBLASTROW の値は常に同じです。DBBUFFER オプションを設定してバッファリングを使用可能にしている場合は、DBFIRSTROW はロー・バッファの最初のローの番号を返します。
- サーバから最初に返されたロー (値は 1) が、ロー・バッファの最初のローとはかぎりません。ロー・バッファ内のローは、アプリケーション・プログラムの操作によって変動します。詳細については、dbclrbuf のリファレンス・ページを参照してください。

参照

dbclrbuf、DBCURROW、dbgetrow、DBLASTROW、dbnextrow、dbsetopt、「オプション」(436 ページ)

dbfree_xlate

説明

1 組の文字セット変換テーブルを解放します。

構文

```
RETCODE *dbfree_xlate(dbproc, xlt_tosrv, xlt_todisp)

DBPROCESS    *dbproc;
DBXLATE      *xlt_tosrv;
DBXLATE      *xlt_todisp;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信およびデータの管理に使用するすべての情報が含まれます。

xlt_tosrv
ディスプレイ固有の文字列をサーバ文字列に変換するために使用する変換テーブルを指すポインタです。この変換テーブルは、dbload_xlate を使用して割り付けます。

xlt_todisp
サーバ文字列をディスプレイ固有の文字列に変換するために使用する変換テーブルを指すポインタです。この変換テーブルは、dbload_xlate を使用して割り付けます。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、dbload_xlate によって割り付けられた 1 組の文字セット変換テーブルを解放します。

- 文字セット変換テーブルは、サーバの標準文字セットとディスプレイ・デバイスの文字セットの間で文字を変換します。
- 次のコードは、dbfree_xlate の例です。

```
char      destbuf[128];
int       srcbytes_used;
DBXLATE   *xlt_todisp;  DBXLATE     *xlt_tosrv;
dbload_xlate((DBPROCESS *)NULL, "iso_1",
"trans.xlt", &xlt_tosrv, &xlt_todisp);
printf("Original string:¥n¥t%s¥n¥n",
      TEST_STRING);
dbxlate((DBPROCESS *)NULL, TEST_STRING,
      strlen(TEST_STRING), destbuf, -1, xlt_todisp,
      &srcbytes_used);
printf("Translated to display character set:¥
      ¥n¥t%s¥n¥n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
    xlt_todisp);
```

参照 dbload_xlate、dbxlate

dbfreebuf

説明 コマンド・バッファをクリアします。

構文 void dbfreebuf(dbproc)

DBPROCESS *dbproc;

パラメータ dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 なし。

使用法

- このルーチンは、DBPROCESS コマンド・バッファに割り付けられていた領域を解放してこのバッファをクリアします。次に、コマンド・バッファを NULL に設定します。コマンドをコマンド・バッファに追加するには、dbcmd ルーチンまたは dbfcmd ルーチンを使用します。

- dbsqlexec または dbsqlsend を呼び出した後の最初の dbcmd または dbfcmd の呼び出し時に、新しいテキストが入力される前に dbfreebuf が自動的に呼び出されて、コマンド・バッファがクリアされます。これを行わない場合は、DBNOAUTOFREE オプションを設定してください。DBNOAUTOFREE が設定されているときは、dbfreebuf を明示的に呼び出した場合にのみコマンド・バッファがクリアされます。
- アプリケーションは、いつでも dbgetchar、dbstrlen、dbstrcpy を呼び出してコマンド・バッファの内容にアクセスできます。

参照　dbcmd、dbfcmd、dbgetchar、dbsqlexec、dbsqlsend、dbstrcpy、dbstrlen、「オプション」(436 ページ)

dbfreequal

説明　dbqual で割り付けられたメモリを解放します。

構文

```
void dbfreequal(qualptr)

char        *qualptr;
```

パラメータ

qualptr
dbqual によって割り付けられたメモリを指すポインタです。

戻り値　なし。

使用法

- dbfreequal は、DB-Library ブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「第 1 章 DB-Library の概要」を参照してください。
- dbqual は、ブラウズ可能なテーブル内の 1 つのローを更新するためにアプリケーションが使用できる where 句を生成します。このとき、where 句を格納するバッファを動的に割り付けます。where 句が不要になると、アプリケーションは dbfreequal を使用してバッファを解放できます。

参照　dbqual

dbfreesort

説明

dbloadsort で割り付けられたソート順構造体を解放します。

構文

RETCODE dbfreesort(dbproc, sortorder)

```
DBPROCESS       *dbproc;
DBSORTORDER     *sortorder;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

sortorder

dbloadsort によって割り付けられた DBSORTORDER 構造体を指すポインタです。

戻り値

SUCCEED または FAIL。

使用法

- dbfreesort は、dbloadsort によって割り付けられたソート順構造体を解放します。dbstrcmp や dbstrsort などの DB-Library ルーチンは、文字データのソートの方法を決めるときにソート順を使用します。
- アプリケーション・プログラムは、ソートや比較を行うときに、サーバと同じ方法で自動的に文字データをソートします。ソート順がロードされていないときは、dbstrcmp や dbstrsort などのルーチンはバイナリ値順に文字をソートします。

  警告！アプリケーション・プログラムで、オペレーティング・システムの機能を使用して **sortorder* 構造体を直接解放することは避けてください。この構造体は、malloc をサポートしないオペレーティング・システムでは malloc とは別のメカニズムを使用して割り付けられるので、複数の部分から構成される可能性があり、その場合は残りの部分を別に解放しなければならないからです。

- 次のコードは、dbfreesort の例です。

```
sortorder = dbloadsort(dbproc);

retval = dbstrcmp(dbproc, "ABC", 3, "abc", 3,
     sortorder);
printf("ABC dbstrcmp' with abc yields %d.¥n",
     retval);

retval = dbstrcmp(dbproc, "abc", 3, "ABC", 3,
```

```
        sortorder);
    printf("abc dbstrcmp' with ABC yields %d.¥n",
        retval);

    dbfreesort(dbproc, sortorder);
```

参照　dbloadsort、dbstrcmp、dbstrsort

dbgetchar

説明　コマンド・バッファ内の文字を指すポインタを返します。

構文

```
char *dbgetchar(dbproc, n)

DBPROCESS    *dbproc;
int          n;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

n
コマンド・バッファの目的の文字の位置です。最初の文字の位置は 0 になります。

戻り値　dbgetchar は、コマンド・バッファの *n* 番目の文字を指すポインタを返します。*n* が範囲外の場合、dbgetchar は NULL を返します。

使用法

- dbgetchar を使用すると、コマンド・バッファ内の特定の文字を指すポインタを取得できます。dbgetchar は、コマンド・バッファ内の *n* で指定された位置の文字を指すポインタを返します。最初の文字の位置は 0 です。
- 内部的には、コマンド・バッファは NULL で終了しないテキスト文字列を連結したリストです。dbgetchar、dbstrcpy、dbstrlen を組み合わせると、コマンド・バッファ内の位置を指定してその部分をコピーできます。

- コマンド・バッファは 1 つの単なる大きなテキスト文字列ではなく、複数のテキスト文字列を連結したリストなので、バッファ全体にインデックスを付けるには dbgetchar を使用する必要があります。dbgetchar を使用してポインタだけを取得した後、ユーザが自分でポインタ値を増加させると、文字列の終わりを過ぎてセグメンテーション・フォールトを引き起こす可能性があります。

参照 dbcmd、dbfcmd、dbfreebuf、dbstrcpy、dbstrlen

dbgetcharset

説明 クライアント文字セットの名前を DBPROCESS 構造体から取得します。

構文

```
char *dbgetcharset(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 クライアント文字セットの NULL で終了する名前を指すポインタです。エラーの場合は NULL です。

使用法

- dbgetcharset は、クライアントの文字セットの名前を返します。
- DB-Library/C クライアントは、接続しているサーバとは異なる文字セットを使用できます。クライアントとサーバが異なる文字セットを使用している場合に、クライアントの文字セットに対する文字変換をサーバがサポートしていれば、サーバはクライアントと通信するときにサーバ文字セットとの間の変換を行います。
- アプリケーションが使用している文字セットをサーバに通知するには、DBSETLCHARSET を使用します。
- サーバが文字セット変換を実行するかどうかをアプリケーションで調べるには、dbcharsetconv を呼び出します。
- サーバの文字セット名をアプリケーションで取得するには、dbservcharset を呼び出します。

参照 dbcharsetconv、dblogin、dbopen、dbservcharset、DBSETLCHARSET

dbgetloginfo

説明

Tabular Data Stream (TDS) ログイン応答情報を、DBPROCESS 構造体から、新しく割り付けられた DBLOGINFO 構造体に転送します。

構文

RETCODE dbgetloginfo(dbproc, loginfo)

```
DBPROCESS     *dbproc;
DBLOGINFO     **loginfo;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

loginfo

DBLOGINFO ポインタ変数のアドレスです。dbgetloginfo は、新しく割り付けられた DBLOGINFO 構造体のアドレスを指すように DBLOGINFO ポインタを設定します。

戻り値

SUCCEED または FAIL。

使用法

- dbgetloginfo は TDS ログイン応答情報を、DBPROCESS 構造体から、新しく割り付けられた DBLOGINFO 構造体に転送します。
- アプリケーションで dbgetloginfo を呼び出す必要があるのは、1) そのアプリケーションが Open Server ゲートウェイ・アプリケーションであり、2) TDS パススルーを使用している場合のみです。
- TDS は、クライアントとサーバの間で、要求と要求結果を転送するために使用されるアプリケーション・プロトコルです。
- クライアントがサーバと直接接続している場合、2 つのプログラムは、データの送受信に使用する TDS のフォーマットをネゴシエートします。ゲートウェイ・アプリケーションが TDS パススルーを使用する場合は、TDS パケットは検査も処理も行われず、そのままクライアントとリモート・サーバの間で転送されます。このため、リモート・サーバとクライアントは、使用する TDS フォーマットについて合意しておく必要があります。
- dbgetloginfo は、クライアントとリモート・サーバが TDS フォーマットをネゴシエートするための 4 つの呼び出しの 2 番目の呼び出しです。この 4 つのうち 2 つは Server Library 呼び出しです。SRV_CONNECT イベント・ハンドラの中でだけ実行できるこれら 4 つの呼び出しは、次のとおりです。

- srv_getloginfo － DBLOGINFO 構造体を割り付け、この構造体にクライアント SRV_PROC からの TDS 情報を格納します。
- dbsetloginfo － 手順 1 で取得した TDS 情報を、DBLOGINFO 構造体から DB-Library/C の LOGINREC 構造体に転送してから、DBLOGINFO 構造体を解放します。情報が転送されると、アプリケーションで dbopen を呼び出してリモート・サーバとの接続を確立するときに、この LOGINREC 構造体を使用できます。
- dbgetloginfo － クライアントの TDS 情報に対するリモート・サーバの応答を、DBPROCESS 構造体から、新しく割り付けられた DBLOGINFO 構造体に転送します。
- srv_setloginfo － 前の手順で取得したリモート・サーバの応答をクライアントに送信してから、DBLOGINFO 構造体を解放します。

- 次に、TDS パススルーを行うリモート接続を用意する SRV_CONNECT ハンドラの例を示します。

```
RETCODE connect_handler(srvproc)
SRVPROC       *srvproc;
 {
 DBLOGINFO    *loginfo;
 LOGINREC     *loginrec;
 DBPROCESS    *dbproc;
/*
 ** Get the TDS login information from the client
 ** SRV_PROC.
 */
 srv_getloginfo(srvproc, &loginfo);
/* Get a LOGINREC structure */
 loginrec = dblogin();
/*
 ** Initialize the LOGINREC with the login info
 ** from the SRV_PROC.
 */
 dbsetloginfo(loginrec, loginfo);
/* Connect to the remote server */
 dbproc = dbopen(loginrec, REMOTE_SERVER_NAME);
/*
 ** Get the TDS login response information from
 ** the remote connection.
 */
 dbgetloginfo(dbproc, &loginfo);
/*
 ** Return the login response information to the
```

```
        ** SRV_PROC.
        */
        srv_setloginfo(srvproc, loginfo);
       /* Accept the connection and return */
        srv_senddone(srvproc, 0, 0, 0);
        return(SRV_CONTINUE);
        }
```

参照　dbrecvpassthru、dbsendpassthru、dbsetloginfo

dbgetlusername

説明　LOGINREC 構造体からユーザ名を返します。

構文　int dbgetlusername(login, name_buffer, buffer_len)

```
LOGINREC  *login;
BYTE      *name_buffer;
int       buffer_len;
```

パラメータ

login
dbopen に引数として渡すことができる、LOGINREC 構造体を指すポインタです。LOGINREC 構造体を取得するには、dblogin を呼び出します。

name_buffer
バッファを指すポインタです。ユーザ名は、LOGINREC 構造体からこのバッファにコピーされます。

buffer_len
コピー先バッファのバイト単位の長さです。

戻り値　バッファにコピーされたバイト数です。このバイト数には、NULL ターミネータは含まれません。

ユーザ名が *buffer_len* - 1 バイトよりも長い場合、dbgetlusername は *buffer_len* - 1 バイトをバッファにコピーして、DBTRUNCATED を返します。

login が NULL の場合、*name_buffer* が NULL の場合、*buffer_len* が 0 より小さい場合は、dbgetlusername は FAIL を返します。

使用法

- dbgetlusername は、ユーザ名を LOGINREC 構造体から *name_buffer* バッファにコピーします。

- LOGINREC 構造体内のユーザ名を設定するには、DBSETLUSER を使用してください。
- dbgetlusername は、最大で *buffer_len* - 1 バイトをコピーし、ユーザ名文字列に NULL ターミネータを付けます。LOGINREC 構造体内のユーザ名の長さは最大 DBMAXNAME バイトなので、アプリケーションには DBMAXNAME + 1 バイトより長いバッファは必要ありません。
- LOGINREC 内のユーザ名が *buffer_len* - 1 バイトよりも長い場合、dbgetlusername は名前をトランケートして DBTRUNCATED を返します。

参照 dblogin、DBSETLUSER

dbgetmaxprocs

説明 同時にオープンできる DBPROCESS の現在の最大数を調べます。

構文 int dbgetmaxprocs()

パラメータ なし。

戻り値 同時にオープンできる DBPROCESS の現在の最大数を表す整数です。

使用法 1 つの DB-Library プログラムで同時にオープンできる DBPROCESS の数には上限があります。デフォルトでは、この数は 25 です。アプリケーションは、dbsetmaxprocs を呼び出してこの制限を変更できます。

参照 dbopen、dbsetmaxprocs

dbgetnatlang

説明 各国言語を DBPROCESS 構造体から取得します。

構文 char* dbgetnatlang(dbproc)

```
DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

クライアント DBPROCESS が使用している各国言語を表す文字列を指すポインタです。

使用法

- dbgetnatlang は、クライアントが使用している各国言語の名前を指すポインタを返します。
- DB-Library/C クライアントは、接続先のサーバとは異なる各国言語を使用できます。アプリケーションが使用しようとする言語をサーバに通知するには、DBSETLNATLANG を使用します。

参照

dblogin、dbopen、DBSETLNATLANG

dbgetoff

説明

コマンド・バッファ内に Transact-SQL 構成体があるかどうかを調べます。

構文

int dbgetoff(dbproc, offtype, startfrom)

```
DBPROCESS       *dbproc;
DBUSMALLINT     offtype;
int             startfrom;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

offtype
検索するオフセットのタイプです。このタイプは、ヘッダ・ファイル *sybdb.h* に定義されています。次のタイプがあります。

OFF_SELECT
OFF_FROM
OFF_ORDER
OFF_COMPUTE
OFF_TABLE
OFF_PROCEDURE
OFF_STATEMENT
OFF_PARAM
OFF_EXEC

詳細については、「オプション」(436 ページ) を参照してください。

startfrom
バッファ内の検索を始める場所です。コマンド・バッファの先頭は 0 です。

戻り値

指定したオフセットの、コマンド・バッファ内の文字オフセットです。指定のオフセットが見つからない場合は、-1 が返されます。

使用法

- DBOFFSET オプションが設定されていれば (「オプション」(436 ページ) を参照)、このルーチンを使用して、コマンド・バッファ内に特定の Transact-SQL 構成体があるかどうかを調べることができます。次の簡単な例では、プログラムがコマンド・バッファの内容を認識しておらず、SQL キーワードの select が出現する位置を知る必要があると仮定します。

```
int      select_offset[10];
int      last_offset;
int      i;
/* Set the offset option */
 dbsetopt(dbproc, DBOFFSET, "select");

/*
 ** Assume the command buffer contains the
 ** following selects.
 */
 dbcmd(dbproc, "select x = 100 select y = 5");

/* Send the query to Adaptive Server Enterprise */
 dbsqlexec(dbproc);

/* Get all the offsets to the select keyword */
 for (i = 0, last_offset = 0; last_offset != -1;
      i++)
```

```
if ((last_offset = dbgetoff(dbproc,
     OFF_SELECT, last_offset) != -1)
select_offset[i] = last_offset++;
```

この例では、select_offset[0] = 0、select_offset[1] = 15 です。

- dbgetoff はサブクエリ内の select 文を認識しません。たとえば、コマンド・バッファの内容が次のとおりであるとします。

```
select pub_name
from publishers
where pub_id not in
   (select pub_id
    from titles
    where type = "business")
```

このとき、2 番目の「select」は認識されません。

参照　dbcmd、dbgetchar、dbsetopt、dbstrcpy、dbstrlen、「オプション」(436 ページ)

dbgetpacket

説明　現在使用している TDS パケット・サイズを返します。

構文　int dbgetpacket(dbproc)

DBPROCESS　*dbproc;

パラメータ　dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　現在使用中の TDS パケット・サイズを返します。

使用法

- dbgetpacket は、現在使用中の TDS パケット・サイズを返します。
- TDS (Tabular Data Stream) は、クライアントとサーバ間の要求と要求結果の転送に使用されるアプリケーション・プロトコルです。
- TDS データは、「パケット」と呼ばれる固定サイズのまとまりに分けて送信されます。TDS パケットのデフォルト・サイズは、512 バイトです。

- アプリケーションでこの TDS パケット・サイズを変更するには、DBSETLPACKET を使用して LOGINREC 構造体内のパケット・サイズ・フィールドを設定します。アプリケーションがサーバまたは Open Server にログインすると、サーバは、作成された DBPROCESS 接続の TDS パケット・サイズを、このフィールドの値以下に設定します。サーバに領域の制約がある場合は、パケット・サイズの設定値がこのフィールドの値より小さくなります。それ以外の場合には、パケット・サイズはこのフィールドの値と等しくなります。
- アプリケーションで大量の text データまたは image データを送受信する場合は、パケットをデフォルトの 512 バイトより大きくすれば、ネットワークの読み込みと書き込みが少なくなるので、効率的です。

参照 DBSETLPACKET

dbgetrow

説明 ロー・バッファ内の指定のローを読み込みます。

構文

```
STATUS dbgetrow(dbproc, row)

DBPROCESS    *dbproc;
DBINT        row;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

row

読み込むローの番号です。サーバから最初に返されたローの番号が 1 で、番号は順にカウントされます。ロー・バッファ内の最初のローが必ずしもサーバから最初に返されたローではないことに注意してください。

戻り値 dbgetrow が返す値には、次の 4 種類があります。

- 現在のローが通常ローの場合は、REG_ROW が返されます。

- 現在のローが計算ローの場合は、ローの *computeid* が返されます (*computeid* については、dbaltbind のリファレンス・ページを参照してください)。
- 指定のローがロー・バッファにない場合は NO_MORE_ROWS が返され、現在のローはそのまま変わりません。
- ルーチンが失敗した場合、FAIL が返されます。

使用法

- dbgetrow は、ロー・バッファ内の指定のローを現在のローとして設定し、このローを読み込みます。このルーチンは、DBBUFFER オプションがオンで、ロー・バッファリングが使用可能になっている場合にだけ機能します。dbgetrow が呼び出されると、dbbind または dbaltbind で指定されたロー・データからプログラム変数へのバインドが有効になります。
- ロー・バッファリングを利用すると、指定した数のサーバ結果ローをプログラムのメモリ内に保持できます。ロー・バッファリングを行わない場合は、新しい dbnextrow 呼び出しによって生成される結果ローは、直前の結果ローの内容を上書きします。したがって、プログラムでランダムな順序で結果ローにアクセスする場合は、ロー・バッファリングが便利です。ただし、バッファ内のローごとに割り付けおよび解放を行う必要があるので、メモリおよびパフォーマンスへの影響があります。したがって、この機能は必要な場合にだけ使用してください。特に、dbgetrow または dbsetrow を呼び出す場合にだけ DBBUFFER オプションをオンにしてください。ロー・バッファリングはネットワーク・バッファリングとは無関係であり、まったく別の問題である点に注意してください。
- アプリケーションでロー・バッファリングを使用できない場合は、NO_MORE_ROWS が返されるまで dbnextrow を繰り返し呼び出すことにより、サーバからローを読み込んだ時点でそのローを処理します。ロー・バッファリングが使用可能な場合は、dbnextrow を使用してこれまでにサーバから読み込んだ任意のローに dbgetrow を使用して移動することができます。その後で dbnextrow を呼び出すと、アプリケーションはバッファ内の次のローを読み込むことになります。dbnextrow がバッファの最後のローに達したとき、さらにサーバからのローがある場合は、再度サーバからローを読み込みます。バッファが一杯のときは、dbclrbuf を使用して一部のローをバッファからクリアしなければ、dbnextrow を実行してもサーバからのローの読み込みは行われません。

- DBFIRSTROW、DBLASTROW、DBCURROW の 3 つのマクロは、dbgetrow 呼び出しと一緒に使用すると便利です。たとえば、DBFIRSTROW はバッファ内の最初のローの番号を取得します。次に例を示します。

  ```
  dbgetrow(dbproc, DBFIRSTROW(dbproc))
  ```

  この呼び出しでは、バッファ内の最初のローが現在のローとして設定されます。

- dbsetrow ルーチンは、バッファされているローの 1 つを「現在のロー」として設定しますが、ローの内容は読み込みません。
- ロー・バッファリングの例については、オンラインのサンプル・プログラムの *example4.c* を参照してください。

参照

dbaltbind、dbbind、dbclrbuf、DBCURROW、DBFIRSTROW、DBLASTROW、dbnextrow、dbsetrow、「オプション」(436 ページ)

DBGETTIME

説明

DB-Library が SQL コマンドに対するサーバの応答を待つ時間を秒数で返します。

構文

int DBGETTIME()

戻り値

タイムアウト値です。つまり、DB-Library がサーバの応答を待つ時間 (秒) がこの値に達するとタイムアウトになります。タイムアウト値 0 は、無期限のタイムアウト時間を示します。

使用法

- このルーチンは、dbsqlexec、dbsqlok、dbresults、dbnextrow を呼び出したときに DB-Library がサーバからの応答を待つ時間を、秒単位で返します。デフォルトのタイムアウト値は 0 で、無期限のタイムアウト時間を意味します。
- プログラムでタイムアウト値を変更するには、dbsettime を呼び出します。

参照

dbsettime

dbgetuserdata

説明

DBPROCESS 構造体からユーザ割り付けのデータを指すポインタを返します。

構文

```
BYTE *dbgetuserdata(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

ユーザのプライベートなデータ領域を指す、汎用の BYTE ポインタです。このポインタは、dbsetuserdata ルーチンを使用して事前に保存しておく必要があります。

使用法

- DBPROCESS 構造体からユーザ割り付けのデータを指すポインタを返します。アプリケーションは、dbsetuserdata ルーチンを使用して事前にこのポインタを保存しておく必要があります。
- アプリケーションで dbgetuserdata と dbsetuserdata を使用すると、ユーザ・データを特定の DBPROCESS に関連付けることができます。これにより、グローバル変数を使用して対応付ける必要がなくなります。これらのルーチンの用途の 1 つに、dbsetuserdata のリファレンス・ページの例に示したデッドロックの処理があります。この例では、アプリケーションのメッセージ・ハンドラがデッドロックを検出したときに、トランザクションを再実行します。
- このルーチンは、アプリケーションに複数の DBPROCESS 構造体があるときに特に役立ちます。

参照

dbsetuserdata

dbhasretstat

説明

現在のコマンドまたはリモート・プロシージャ・コールがリターン・ステータス番号を生成したかどうかを調べます。

構文

```
DBBOOL dbhasretstat(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

「TRUE」または「FALSE」。

使用法

- このルーチンは、現在の Transact-SQL コマンドまたはリモート・プロシージャ・コールがリターン・ステータス番号を生成したかどうかを調べます。ステータス番号は、Adaptive Server Enterprise で実行されるすべてのストアド・プロシージャから返されます。ステータス番号はストアド・プロシージャの機能であるため、ステータス番号を生成できるのは、リモート・プロシージャ・コールと execute コマンドだけです。
- 実際にステータス番号を取得するのは、dbretstatus ルーチンです。正常に終了したストアド・プロシージャは、ステータス番号 0 を返します。リターン・ステータス番号のリストについては、『リファレンス・マニュアル』を参照してください。
- ストアド・プロシージャを実行するとき、サーバは、その他のすべての結果を返した直後にステータス番号を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbhasretstat を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションで dbhasretstat または dbretstatus を呼び出すには、その前に dbresults と dbnextrow を必要な回数だけ呼び出してすべての結果を処理する必要があります。
- アプリケーションがステータス番号およびリターン・パラメータ値を処理する順序は重要ではありません。
- ストアド・プロシージャを RPC コマンドとして実行するときは (dbrpcinit、dbrpcparam、dbrpcsend を使用)、他の結果の処理がすべて完了してからリターン・ステータスを取り出すことができるようになります。この使用方法については、オンラインのサンプル・プログラムの *example8.c* を参照してください。
- ストアド・プロシージャを Transact-SQL コマンドのバッチから実行した場合は (dbsqlexec または dbsqlsend を使用)、ストアド・プロシージャの実行後に他のコマンドが実行される可能性があります。このような状況では、リターン・ステータスの取得が多少複雑になります。

- ストアド・プロシージャ・コマンドがバッチ内にある唯一のコマンドであることが確実な場合は、オンラインのサンプル・プログラムの *example8.c* のように、dbresults ループの後でリターン・ステータスを取り出します。
- バッチ内に複数のコマンドが存在する可能性がある場合は、リターン・ステータスの取り出しは、dbresults ループの内側で、dbnextrow を使用してすべてのローをフェッチした後に行ってください。次のコードはこの状況でリターン・ステータス値を検索するプログラム論理です。

```
while ( (result_code = dbresults(dbproc)
        != NO_MORE_RESULTS)
{
  if (result_code == SUCCEED)
  {
    ... bind rows here ...
    while ((row_code = dbnextrow(dbproc))
           != NO_MORE_ROWS)
    {
      ... process rows here ...
    }
    /* Now check for a return status */
    if (dbhasretstat(dbproc) == TRUE)
    {
      printf("(return status %d)¥n",
             dbretstatus(dbproc));
    }
    if (dbnumrets(dbproc) > 0)
    {
      ... get output parameters here ...
    }
  } /* if result_code */
  else
  {
    printf("Query failed.¥n");
  }
} /* while dbresults */
```

参照　dbnextrow、dbresults、dbretdata、dbretstatus、dbrpcinit、dbrpcparam、dbrpcsend

dbinit

説明 DB-Library を初期化する。

構文 RETCODE dbinit()

戻り値 SUCCEED または FAIL。

使用法

- このルーチンは、特定のプライベート DB-Library 構造体を初期化します。この初期化が必要な環境では、dbinit は他の DB-Library ルーチンを呼び出す前に呼び出す必要があります。ほとんどの DB-Library ルーチンは、dbinit より先に呼び出されると、アプリケーションを終了させます。
- 将来の互換性と移植性を確実にするために、オペレーティング環境にかかわらず、すべてのアプリケーションで dbinit を呼び出すことを強くおすすめします。

参照 dbexit

DBIORDESC

説明 (UNIX のみ) サーバから送られてくるデータを読み込むために DBPROCESS が使用する、UNIX ファイル記述子に対するアクセスをプログラムに提供する。

構文

```
int DBIORDESC(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 指定した DBPROCESS がサーバからのデータを読み込むのに使用する、整数のファイル記述子です。

使用法

- このルーチンを利用すると、アプリケーションが複数の入力ストリームに的確に応答することができます。アプリケーションの性質によっては、サーバに対する情報要求 (通常は dbsqlsend 呼び出しによって行われる) とサーバの応答 (dbsqlok、dbresults、または dbnextrow を呼び出して読み込まれる) の間に、かなりの時間がかかることがあります。この時間を、アプリケーションの他の部分のサービスに使用できます。DBIORDESC ルーチンを使用すると、DBPROCESS がサーバからデータ・ストリームを読み込むときに使用する I/O 記述子を取得できます。この情報をオペレーティング・システムのさまざまな機能 (UNIX の UNIX select 呼び出しなど) とともに使用すると、アプリケーションで複数の入力ストリームに効率よく応答できるようになります。
- dbpoll は、アプリケーションのサーバ接続 (DBPROCESS ポインタで示す) にサーバ応答が到着しているかどうかを調べます。一般に、dbpoll の方が DBIORDESC よりも使い方は簡単です。この理由と、DBIORDESC に移植性がないという理由から、通常は dbpoll を使用します。
- DBIORDESC から返されるファイル記述子は、これと組み合わせて使用するオペレーティング・システム機能が、受信データ・ストリームからデータを読み込むものではない場合にのみ使用できます。このストリームからのデータが、DB-Library ルーチン以外の手段で読み込まれると、フロントエンドとサーバの間の通信に混乱をきたすことになります。
- アプリケーションで、サーバから読み込むデータがまだあるかどうかを調べるとき、UNIX の select 関数のほかに DB-Library の DBRBUF ルーチンを使用できます。
- このルーチンと対になる DBIOWDESC ルーチンは、データをサーバに書き込むために使用するファイル記述子にアクセスできるようにするものです。

参照

dbcmd、DBIOWDESC、dbnextrow、dbpoll、DBRBUF、dbresults、dbsqlok、dbsqlsend

DBIOWDESC

説明

(UNIX のみ) サーバにデータを書き込むために DBPROCESS が使用する、UNIX ファイル記述子に対するアクセスをプログラムに提供する。

構文

int DBIOWDESC(dbproc)

DBPROCESS *dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

指定した DBPROCESS がサーバにデータを書き込むのに使用する、整数のファイル記述子です。

使用法

- このルーチンを使用すると、アプリケーションで複数の入力および出力ストリームを有効利用できるようになります。アプリケーションの性質によっては、サーバへの情報の書き込みの開始 (通常は dbsqlsend 呼び出しによって行われる) からその作業の完了までに、かなりの時間がかかることがあります。この時間を、アプリケーションの他の部分のサービスに使用できます。DBIOWDESC ルーチンを使用すると、DBPROCESS がサーバにデータ・ストリームを書き込むときに使用する I/O 記述子を取得できます。この情報をオペレーティング・システムのさまざまな機能 (UNIX の select 関数など) と組み合わせることにより、アプリケーションで複数の入力および出力ストリームを有効利用できるようになります。
- このルーチンから返されるファイル記述子は、これと組み合わせるオペレーティング・システム機能が、送信データ・ストリームにデータを書き込むものではない場合にのみ使用できます。このストリームへのデータが、DB-Library ルーチン以外の手段で書き込まれると、フロントエンドとサーバの間の通信に混乱をきたすことになります。
- このルーチンと対になる DBIORDESC ルーチンを使用すると、サーバからのデータの読み込みに使用されるファイル記述子にアクセスできるようになります。アプリケーションによっては、DBIORDESC よりも、別のルーチンである dbpoll の方が適している場合もあります。

参照

dbcmd、DBIORDESC、dbnextrow、dbpoll、dbresults、dbsqlok、dbsqlsend

DBISAVAIL

説明　DBPROCESS が通常使用可能かどうかを調べます。

構文

```
DBBOOL DBISAVAIL(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　「DBPROCESS が通常使用可能な場合は「TRUE」です。それ以外の場合は「FALSE」です。

使用法　このルーチンは、指定の DBPROCESS に対して通常の使用が可能であるかどうかを示します。DBPROCESS がオープンされた時点の状態は「使用可能」であり、これはなんらかの作業で使用されるまで続きます。多くの DB-Library ルーチンは、自動的に DBPROCESS を「使用不可能」に設定しますが、dbsetavail だけは DBPROCESS を「使用可能」に再設定します。この機能は、1 つのプログラム内の各部分で 1 つの DBPROCESS を共用する場合に役立ちます。

参照　dbsetavail

dbisopt

説明　サーバ・オプションまたは DB-Library オプションのステータスを調べます。

構文

```
DBBOOL dbisopt(dbproc, option, param)

DBPROCESS    *dbproc;
int          option;
char         *param;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。dbsetopt 関数と dbclropt 関数とは異なり、この関数では *dbproc* に NULL を指定することはできません。

option
対象となるオプションです。オプションのリストについては、「オプション」(436 ページ) を参照してください。

param
オプションにはパラメータを指定できるものがあります。たとえば、DBOFFSET オプションは SQL 構成体をパラメータとして受け取り、そのオフセットを返します。パラメータを指定できるオプションのリストについては、オプション を参照してください。パラメータを指定しないオプションの場合は、*param* を NULL にしてください。

対象となるオプションが、パラメータを受け取るけれども、インスタンスが 1 つしか存在しないものである場合は、dbisopt はこの *param* 引数を無視します。たとえば、ロー・バッファリングの設定は一度に 1 つしか存在しないので、DBBUFFER オプションを調べるときは、dbisopt は *param* の値を無視します。一方、DBOFFSET オプションには複数の設定値が可能で、それぞれに異なるパラメータがあります。select 文に対するオフセットと order by 句に対するオフセットを調べる場合のように、DBOFFSET が 2 回設定されていることもあります。この場合、dbisopt は、select オフセットと order by オフセットのどちらを調べるのかを判断するのに、*param* 引数を必要とします。

戻り値
「TRUE」または「FALSE」。

使用法
- このルーチンは、サーバ・オプションと DB-Library オプションのステータスを調べます。サーバ・オプションは SQL を介して直接設定したりクリアしたりできますが、アプリケーションでは、オプションの設定とクリアには dbsetopt と dbclropt を使用してください。このようにすれば、サーバ・オプションと DB-Library オプションを設定するインタフェースを統一できます。また、アプリケーションで dbisopt 関数を使用してオプションのステータスを調べることもできます。
- 各オプションとそのデフォルト・ステータスのリストについては、「オプション」(436 ページ) を参照してください。

参照
dbclropt、dbsetopt、「オプション」(436 ページ)

DBLASTROW

説明
ロー・バッファ内の最後のローの番号を返します。

構文

```
DBINT DBLASTROW(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

ロー・バッファ内の最後のローの番号です。エラーの場合、このルーチンは 0 を返します。

使用法

- このマクロは、ロー・バッファ内の最後のローの番号を返します。ローの番号は、ロー・バッファの先頭からではなく、サーバから最初に返されたローを 1 としてカウントされます。
- ローをバッファリングしていない場合、DBFIRSTROW、DBCURROW、DBLASTROW は常に同じ値です。DBBUFFER オプションを設定してバッファリングを使用可能にしている場合は、DBLASTROW はロー・バッファ内の最後のローの番号を返します。

参照

dbclrbuf、DBCURROW、DBFIRSTROW、dbgetrow、dbnextrow、dbsetopt、「オプション」(436 ページ)

dbload_xlate

説明

1 組の文字セット変換テーブルをロードします。

構文

```
RETCODE dbload_xlate(dbproc, srv_charset, xlate_name,
                     xlt_tosrv, xlt_todisp)

DBPROCESS    *dbproc;
char         *srv_charset;
char         *xlt_name;
DBXLATE      **xlt_tosrv;
DBXLATE      **xlt_todisp;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

srv_charset
サーバの文字セットの名前を指すポインタです。dbload_xlate は、メインの Sybase インストール・ディレクトリの下の *charsets* ディレクトリ内でこの名前のディレクトリを探します。たとえば、サーバが iso_1 文字セットを使用している場合、dbload_xlate は *$SYBASE/charsets/iso_1* を探します。

xlt_name
ディスプレイ固有の文字セットが格納されているファイルの名前を指すポインタです。dbload_xlate は、サーバ文字セット・ディレクトリ内で、このファイルを探します。

xlt_tosrv
ディスプレイ固有の文字列をサーバ文字列に変換するのに使用する文字セット変換テーブルを指すポインタです。この変換テーブルは、dbload_xlate を使用して割り付けます。

xlt_todisp
サーバ文字列をディスプレイ固有の文字列に変換するのに使用する文字セット変換テーブルを指すポインタです。この変換テーブルは、dbload_xlate を使用して割り付けます。

戻り値

SUCCEED または FAIL。

使用法

- dbload_xlate は、ディスプレイ固有のローカライゼーション・ファイルを読み込み、2 つの文字セット変換テーブルを割り付けます。これは、サーバの文字セットからディスプレイ固有の文字セットに変換するためのテーブルと、ディスプレイ固有の文字セットからサーバの文字セットに変換するためのテーブルです。
- 次のコードは、dbload_xlate の例です。

```
char        destbuf[128];
int         srcbytes_used;
DBXLATE*    xlt_todisp;
DBXLATE     *xlt_tosrv;

dbload_xlate((DBPROCESS *)NULL, "iso_1",
      "trans.xlt", &xlt-tosrv, &xlt-todisp);
printf("Original string:¥n¥t%s¥n¥n",
       TEST_STRING);
dbxlate((DBPROCESS *)NULL, TEST_STRING,
      strlen(TEST_STRING), destbuf, -1, xlt_todisp,
      &srcbytes_used);
printf("Translated to display character set:¥
      ¥n¥t%s¥n¥n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
      xlt_todisp);
```

参照　dbfree_xlate、dbxlate

dbloadsort

説明　サーバのソート順をロードします。

構文

```
DBSORTORDER *dbloadsort(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　正常に終了した場合は、DBSORTORDER 構造体を指すポインタです。エラーの場合は NULL です。

使用法

- dbloadsort は、サーバの文字セットのソート順に関する情報を返します。この情報は、dbstrcmp または dbstrsort で 2 つの文字列を比較するときに使用されます。
- dbloadsort は、DBSORTORDER 構造体を割り付けて、サーバ文字セットのソート順情報を格納します。この構造体は、dbfreesort によって解放されます。
- 次のコードは、dbloadsort の例です。

```
sortorder = dbloadsort(dbproc);
retval = dbstrcmp(dbproc, "ABC", 3, "abc", 3,
      sortorder);
 printf("ABC dbstrcmp'ed with abc yields %d.¥n",
      retval);
retval = dbstrcmp(dbproc, "abc", 3, "ABC", 3,
      sortorder);
 printf("abc dbstrcmp'ed with ABC yields %d.¥n",
      retval);
dbfreesort(dbproc, sortorder);
```

参照　dbfreesort、dbstrcmp、dbstrsort

dblogin

説明 dbopen で使用するログイン・レコードを割り付けます。

構文 LOGINREC *dblogin()

戻り値 LOGINREC 構造体を指すポインタです。この構造体を割り付けることができない場合、dblogin は NULL を返します。

使用法

- このルーチンは、dbopen で使用する LOGINREC 構造体を割り付けます。
- LOGINREC の各コンポーネントを指定するために使用可能なさまざまなルーチンが用意されています。プログラムでホスト名、ユーザ名、ユーザ・パスワード、アプリケーション名を指定するには、それぞれ DBSETLHOST、DBSETLUSER、DBSETLPWD、DBSETAPP を使用します。一般に、プログラムが指定する必要があるのはユーザ・パスワードだけですが、これも、パスワードが NULL 値の場合には省略できます。LOGINREC 構造体のその他の変数は、デフォルト値に設定されます。
- LOGINREC のその他のコンポーネントも変更できます。
  - LOGINREC 構造体内の各国言語名を設定するには、DBSETLNATLANG を使用します。DBSETLNATLANG は、サーバのデフォルトの各国言語を使用しない場合にのみ呼び出します。
  - LOGINREC 内の TDS パケット・サイズを設定するには、DBSETLPACKET を使用します。TDS パケット・サイズは、明示的に設定されていなければ、デフォルトの 512 バイトになります。TDS は、クライアントとサーバ間の情報の交換に使用されるアプリケーション・プロトコルです。
  - LOGINREC の文字セットを設定するには、DBSETLCHARSET を使用します。アプリケーションで DBSETLCHARSET を呼び出す必要があるのは、ISO-8859-1 (サーバでの名称は「iso_1」) を使用しない場合のみです。
- クライアントとサーバの間で接続を行うとき、接続が失敗する状況には次の 2 つがあります (システムは正しく構成されているものとします)。
  - サーバが存在するマシンが正しく動作しており、ネットワークも正しく動作している場合。

この場合に、指定のポートで受信するサーバがないときは、サーバが存在するマシンは、接続が形成できないことをネットワーク・エラーによってクライアントに通知します。この場合、dbsetlogintime に関係なく、接続は実行できません。

- サーバが存在するマシンがダウンしている場合。

  この場合は、サーバが存在するマシンが応答しません。「応答なし」はエラーとはみなされないので、エラーが発生したという通知がネットワークからクライアントに送られることはありません。ただし、dbsetlogintime の呼び出しによってタイムアウト時間が設定されている場合は、設定された時間内にクライアントが応答を受信しなければタイムアウト・エラーが発生します。

- 次のコードは、dblogin の例です。

```
DBPROCESS     *dbproc;
LOGINREC      *loginrec;
 loginrec = dblogin();
 DBSETLPWD(loginrec, "server_password");
 DBSETLAPP(loginrec, "my_program");
 dbproc = dbopen(loginrec, "my_server");
```

- アプリケーションの dbopen 呼び出しがすべて完了した後は、LOGINREC 構造体は必要なくなります。プログラムは、dbloginfree を呼び出して LOGINREC 構造体を解放できます。

参照　dbloginfree、dbopen、dbrpwclr、dbrpwset、DBSETLAPP、DBSETLCHARSET、DBSETLHOST、DBSETLNATLANG、DBSETLPACKET、DBSETLPWD、DBSETLUSER

dbloginfree

説明　ログイン・レコードを解放します。

構文　void dbloginfree(loginptr)

LOGINREC　*loginptr;

パラメータ　loginptr

LOGINREC 構造体を指すポインタです。

戻り値　なし。

使用法

dblogin は、dbopen が使用する LOGINREC 構造体を返します。アプリケーションの dbopen 呼び出しがすべて完了した後は、LOGINREC 構造体は必要なくなります。dbloginfree は、指定された LOGINREC 構造体に関連付けられているメモリを解放します。

参照

dblogin、dbopen

dbmny4add

説明

2 つの DBMONEY4 値を加算します。

構文

RETCODE dbmny4add(dbproc, m1, m2, sum)

```
DBPROCESS    *dbproc;
DBMONEY4     *m1;
DBMONEY4     *m2;
DBMONEY4     *sum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

DBMONEY4 値を指すポインタです。

m2

DBMONEY4 値を指すポインタです。

sum

加算の結果を保持する DBMONEY4 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmny4add は、オーバフローの場合、または *m1*、*m2*、*sum* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmny4add は、*m1* と *m2* の 2 つの DBMONEY4 値を加算し、その結果を **sum* に格納します。

- オーバフローの場合、dbmny4add は FAIL を返し、**sum* を $0.00 に設定します。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照　dbmny4sub、dbmny4mul、dbmny4divide、dbmny4minus、dbmny4add、dbmny4sub、dbmnymul、dbmnydivide、dbmnyminus

dbmny4cmp

説明　2 つの DBMONEY4 値を比較します。

構文　int dbmny4cmp(dbproc, m1, m2)

```
DBPROCESS    *dbproc;
DBMONEY4     *m1;
DBMONEY4     *m2;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

　このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1
　DBMONEY4 値を指すポインタです。

m2
　DBMONEY4 値を指すポインタです。

戻り値

m1 = *m2* の場合、dbmny4cmp は 0 を返します。

m1 < *m2* の場合、dbmny4cmp は -1 を返します。

m1 > *m2* の場合、dbmny4cmp は 1 を返します。

使用法

- dbmny4cmp は 2 つの DBMONEY4 値を比較します。

- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照 dbmnycmp

dbmny4copy

説明 DBMONEY4 値をコピーします。

構文 RETCODE dbmny4copy(dbproc, src, dest)

```
DBPROCESS     *dbproc;
DBMONEY4      *src;
DBMONEY4      *dest;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

src

コピー元 DBMONEY4 値を指すポインタです。

dest

コピー先 DBMONEY4 変数を指すポインタです。

戻り値 SUCCEED または FAIL。

src または *dest* が NULL の場合、dbmny4copy は FAIL を返します。

使用法

- dbmny4copy は、*src* の DBMONEY4 値を *dest* の DBMONEY4 変数にコピーします。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照 dbmnycopy、dbmnyminus、dbmny4minus

dbmny4divide

説明

DBMONEY4 値を別の DBMONEY4 値で除算します。

構文

RETCODE dbmny4divide(dbproc, m1, m2, quotient)

```
DBPROCESS     *dbproc;
DBMONEY4      *m1;
DBMONEY4      *m2;
DBMONEY4      *quotient;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

除算される数として使用する DBMONEY4 値を指すポインタです。

m2

除算する数として使用する DBMONEY4 値を指すポインタです。

quotient

除算の結果を保持する DBMONEY4 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmny4divide は、オーバフローや 0 による除算の場合、または *m1*、*m2*、*quotient* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmny4divide は、*m1* の DBMONEY4 値を *m2* の DBMONEY4 値で除算し、その結果を **quotient* に格納します。
- オーバフローまたは 0 による除算の場合、dbmny4divide は FAIL を返し、**quotient* を $0.0000 に設定します。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照

dbmny4add、dbmny4sub、dbmny4mul、dbmny4minus、dbmnyadd、dbmnysub、dbmnymul、dbmnydivide、dbmnyminus

dbmny4minus

説明

DBMONEY4 値の符号を反転する。

構文

RETCODE dbmny4minus(dbproc, src, dest)

```
DBPROCESS     *dbproc;
DBMONEY4      *src;
DBMONEY4      *dest;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

src

DBMONEY4 値を指すポインタです。

dest

符号反転の結果を保持する DBMONEY4 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmny4minus は、オーバフローの場合、または *src* と *dest* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmny4minus は、*src* の DBMONEY4 値の符号を反転し、その結果を **dest* に格納します。
- オーバフローの場合、dbmny4minus は FAIL を返します。この場合、**dest* は定義されません。DBMONEY4 の負の最大値の符号を反転しようとすると、オーバフローが発生します。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyminus、dbmnycopy、dbmny4copy

dbmny4mul

説明

2 つの DBMONEY4 値を乗算します。

構文

RETCODE dbmny4mul(dbproc, m1, m2, product)

```
DBPROCESS     *dbproc;
DBMONEY4      *m1;
DBMONEY4      *m2;
DBMONEY4      *product;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

DBMONEY4 値を指すポインタです。

m2

DBMONEY4 値を指すポインタです。

product

乗算の結果を保持する DBMONEY4 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmny4mul は、オーバフローの場合、または *m1*、*m2*、*product* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmny4mul は、*m1* の DBMONEY4 値に *m2* の DBMONEY4 値を乗算し、その結果を **product* に格納します。
- オーバフローの場合、dbmny4mul は FAIL を返し、**product* を $0.0000 に設定します。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照

dbmny4add、dbmny4sub、dbmny4divide、dbmny4minus、dbmnyadd、dbmnysub、dbmnymul、dbmnydivide、dbmnyminus

dbmny4sub

説明

DBMONEY4 値から別の DBMONEY4 値を減算します。

構文

RETCODE dbmny4sub(dbproc, m1, m2, difference)

```
DBPROCESS     *dbproc;
DBMONEY4      *m1;
DBMONEY4      *m2;
DBMONEY4      *difference;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

減算される数の DBMONEY4 値を指すポインタです。

m2

減算する数の DBMONEY4 値を指すポインタです。

difference

減算の結果を保持する DBMONEY4 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmny4sub は、オーバフローの場合、または *m1*、*m2*、*difference* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmny4sub は、*m1* の DBMONEY4 値から *m2* の DBMONEY4 値を減算し、その結果を **difference* に格納します。
- オーバフローの場合、dbmny4sub は FAIL を返し、**difference* を $0.0000 に設定します。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照

dbmny4sub、dbmny4mul、dbmny4divide、dbmny4minus、dbmny4add、dbmnysub、dbmnymul、dbmnydivide、dbmnyminus

dbmny4zero

説明

DBMONEY4 変数を $0.0000 に初期化します。

構文

RETCODE dbmny4zero(dbproc, mny4ptr)

```
DBPROCESS     *dbproc;
DBMONEY4      *mny4ptr;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mny4ptr

初期化する DBMONEY4 値を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmny4zero は、*mny4ptr* が NULL の場合に FAIL を返します。

使用法

- dbmny4zero は、DBMONEY4 値を $0.0000 に初期化します。
- 有効な DBMONEY4 値の範囲は、-$214,748.3648 から $214,748.3647 までです。DBMONEY4 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyzero

dbmnyadd

説明

2 つの DBMONEY 値を加算します。

構文

RETCODE dbmnyadd(dbproc, m1, m2, sum)

```
DBPROCESS     *dbproc;
DBMONEY       *m1;
DBMONEY       *m2;
DBMONEY       *sum;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1
DBMONEY 値を指すポインタです。

m2
DBMONEY 値を指すポインタです。

sum
加算の結果を保持する DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

使用法

- dbmnyadd は、*m1* と *m2* の 2 つの DBMONEY 値を加算し、その結果を **sum* に格納します。
- オーバフローの場合、dbmnyadd は FAIL を返し、**sum* を $0.0000 に設定します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。
- dbmnyadd は、オーバフローの場合、または *m1*、*m2*、*sum* のいずれかが NULL の場合に FAIL を返します。

参照

dbmnysub、dbmnymul、dbmnydivide、dbmnyminus、dbmny4add、dbmny4sub、dbmny4mul、dbmny4divide、dbmny4minus

dbmnycmp

説明

2 つの DBMONEY 値を比較します。

構文

```
int dbmnycmp(dbproc, m1, m2)

DBPROCESS     *dbproc;
```

```
DBMONEY     *m1;
DBMONEY     *m2;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

DBMONEY 値を指すポインタです。

m2

DBMONEY 値を指すポインタです。

戻り値

m1 = *m2* の場合、dbmnycmp は 0 を返します。

m1 < *m2* の場合、dbmnycmp は -1 を返します。

m1 > *m2* の場合、dbmnycmp は 1 を返します。

使用法

- dbmnycmp は 2 つの DBMONEY 値を比較します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmny4cmp

dbmnycopy

説明

DBMONEY 値をコピーします。

構文

RETCODE dbmnycopy(dbproc, src, dest)

```
DBPROCESS   *dbproc;
DBMONEY     *src;
DBMONEY     *dest;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

src

コピー元 DBMONEY 値を指すポインタです。

dest

コピー先 DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

src または *dest* が NULL の場合、dbmnycopy は FAIL を返します。

使用法

- dbmnycopy は、*src* の DBMONEY 値を *dest* の DBMONEY 変数にコピーします。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnycopy、dbmnyminus、dbmny4minus

dbmnydec

説明

DBMONEY 値を 10,000 分の 1 ドルだけ減らします。

構文

```
RETCODE dbmnydec(dbproc, mnyptr)

DBPROCESS     *dbproc;
DBMONEY       *mnyptr;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr

値を減らす DBMONEY 値を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnydec は、オーバフローの場合、または *mnyptr* が NULL の場合に FAIL を返します。

使用法

- dbmnydec は、DBMONEY 値を 10,000 分の 1 ドルだけ減らします。
- DBMONEY の負の最大値からさらに値を減らそうとすると、オーバフローが発生します。オーバフローの場合、dbmnydec は FAIL を返します。この場合、**mnyptr* の内容は不定です。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyinc、dbmnymaxneg

dbmnydivide

説明

DBMONEY 値を別の DBMONEY 値で除算します。

構文

RETCODE dbmnydivide(dbproc, m1, m2, quotient)

```
DBPROCESS    *dbproc;
DBMONEY      *m1;
DBMONEY      *m2;
DBMONEY      *quotient;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1
除算される数として使用する DBMONEY 値を指すポインタです。

m2
除算する数として使用する DBMONEY 値を指すポインタです。

quotient
除算の結果を保持する DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnydivide は、オーバフローや 0 による除算の場合、または *m1*、*m2*、*quotient* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmnydivide は、*m1* の DBMONEY 値を *m2* の DBMONEY 値で除算し、その結果を **quotient* に格納します。
- オーバフローまたは 0 による除算の場合、dbmnydivide は FAIL を返し、**quotient* を $0.0000 に設定します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyadd、dbmnysub、dbmnymul、dbmnyminus、dbmny4add、dbmny4sub、dbmny4mul、dbmny4divide、dbmny4minus

dbmnydown

説明

DBMONEY 値を正の整数で除算します。

構文

RETCODE dbmnydown(dbproc, mnyptr, divisor, remainder)

```
DBPROCESS    *dbproc;
DBMONEY      *mnyptr;
```

```
int             divisor;
int             *remainder;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr

除算される DBMONEY 値を指すポインタです。除算の結果も **mnyptr* に格納されます。

divisor

**mnyptr* を除算する整数です。*divisor* は、65535 以下の正の数にしてください。

remainder

除算の余りを 10,000 分の 1 ドル単位で保持する整数変数を指すポインタです。*remainder* が NULL として渡された場合は、余りは返されません。

戻り値

SUCCEED または FAIL。

dbmnydown は、*mnyptr* が NULL の場合、または *divisor* が 1 から 65535 までの範囲でない場合に FAIL を返します。

使用法

- dbmnydown は、DBMONEY 値を短整数で除算し、その結果を元の DBMONEY 変数に格納して返します。
- dbmnydown は除算の余りを **remainder* に入れます。**remainder* は、除算後に残った 10,000 分の 1 ドル単位の整数です。
- *divisor* は、1 以上 65535 以下にしてください。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyscale、dbmnydivide、dbmny4divide

dbmnyinc

説明

DBMONEY 値を 10,000 分の 1 ドルだけ増やします。

構文

RETCODE dbmnyinc(dbproc, mnyptr)

```
DBPROCESS    *dbproc;
DBMONEY      *mnyptr;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr

値を増やす DBMONEY 値を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnyinc は、オーバフローの場合、または *mnyptr* が NULL の場合に FAIL を返します。

使用法

- dbmnyinc は、DBMONEY 値を 10,000 分の 1 ドルだけ増やします。
- DBMONEY の正の最大値からさらに値を増やそうとすると、オーバフローが発生します。オーバフローの場合、dbmnyinc は FAIL を返します。この場合、**mnyptr* は不定です。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnydec、dbmnymaxpos

dbmnyinit

説明

dbmnyndigit を呼び出すために DBMONEY の値を準備します。

構文

RETCODE dbmnyinit(dbproc, mnyptr, trim, negative)

```
DBPROCESS    *dbproc;
DBMONEY      *mnyptr;
int          trim;
DBBOOL       *negative;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr

初期化する DBMONEY 値を指すポインタです。dbmnyinit は **mnyptr* の値を変更します。

trim

mnyptr* からトリムする桁数です。dbmnyinit は、mnyptr* を 10 の累乗で除算して **mnyptr* の桁数を少なくします。*trim* の値によって、10 の何乗を使用するかが決まります。*trim* は 0 未満であってはなりません。

negative

DBBOOL 変数を指すポインタです。**mnyptr* が負の場合、dbmnyinit はこれを正の数にして、**negative* を「true」に設定します。

戻り値

SUCCEED または FAIL。

mnyptr が NULL の場合、*negative* が NULL の場合、または *trim* が 0 より小さい場合、dbmnyinit は FAIL を返します。

使用法

- dbmnyinit は、文字への変換を行うために DBMONEY 値を初期化します。このとき、不要な桁を除去し、負の数値を正の数値に変換します。
- dbmnyinit は、10 の累乗で除算して DBMONEY 値の桁数を少なくします。整数 *trim* の値によって、10 の何乗を使用するかが決まります。dbmnyinit は、元の値をトリムされた値で置き換えて **mnyptr* を変更します。**mnyptr* が負の場合、dbmnyinit はこれを正の数にして、**negative* を「true」に設定します。
- dbmnyinit と dbmnyndigit は、カスタムの DBMONEY-DBCHAR 変換ルーチンを作成するときに役立ちます。このようなカスタムのルーチンは、dbconvert の DBMONEY-to-DBCHAR 変換の精度 (100 分の 1 ドル単位) では十分ではない場合に役立ちます。また、dbconvert は、カンマを含む文字列を作成しません。

- dbmnyndigit は、DBMONEY 値の右端の桁を DBCHAR として返します。DBMONEY 値のすべての桁を取得するには、dbmnyndigit を繰り返し呼び出してください。詳細については、dbmnyndigit のリファレンス・ページを参照してください。
- dbmnyinit は、ほとんどの場合、dbmnyndigit と一緒に使用されます。単独で使用するときは、dbmnyinit によって負の DBMONEY 値を強制的に正の値に変換し、DBMONEY の値を 10 の累乗で除算できますが、dbmnyinit の真の目的は、dbmnyndigit 呼び出しのために DBMONEY 値を準備することです。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。
- dbmnyinit の使用例については、dbmnyndigit のリファレンス・ページを参照してください。

参照 dbconvert、dbmnyndigit

dbmnymaxneg

説明 サポートされる DBMONEY の負の最大値を返します。

構文

```
RETCODE dbmnymaxneg(dbproc, dest)

DBPROCESS     *dbproc;
DBMONEY       *dest;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

dest

DBMONEY 変数を指すポインタです。

戻り値 SUCCEED または FAIL。

dbmnymaxneg は、*dest* が NULL の場合に FAIL を返します。

使用法

- dbmnymaxneg は、サポートされる DBMONEY の負の最大値を **dest* に格納します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnymaxpos

dbmnymaxpos

説明

サポートされる DBMONEY の正の最大値を返します。

構文

```
RETCODE dbmnymaxpos(dbproc, dest)

DBPROCESS     *dbproc;
DBMONEY       *dest;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

dest

DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnymaxpos は、*dest* が NULL の場合に FAIL を返します。

使用法

- dbmnymaxpos は、サポートされる DBMONEY の正の最大値を **dest* に格納します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnymaxneg

dbmnyminus

説明

DBMONEY 値の符号を反転します。

構文

RETCODE dbmnyminus(dbproc, src, dest)

```
DBPROCESS    *dbproc;
DBMONEY      *src;
DBMONEY      *dest;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

src

DBMONEY 値を指すポインタです。

dest

符号反転の結果を保持する DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnyminus は、オーバフローの場合、または *src* と *dest* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmnyminus は、*src* の DBMONEY 値の符号を反転し、その結果を **dest* に格納します。
- オーバフローの場合、dbmnyminus は FAIL を返します。この場合、**dest* は定義されません。DBMONEY の負の最大値の符号を反転しようとすると、オーバフローが発生します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmny4minus、dbmnycopy、dbmny4copy

dbmnymul

説明

2 つの DBMONEY 値を乗算します。

構文

RETCODE dbmnymul(dbproc, m1, m2, product)

```
DBPROCESS    *dbproc;
DBMONEY      *m1;
DBMONEY      *m2;
DBMONEY      *product;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

DBMONEY 値を指すポインタです。

m2

DBMONEY 値を指すポインタです。

product

乗算の結果を保持する DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnymul は、オーバフローの場合、または *m1*、*m2*、*product* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmnymul は、*m1* の DBMONEY 値に *m2* の DBMONEY 値を乗算し、その結果を **product* に格納します。
- オーバフローの場合、dbmnymul は FAIL を返し、**product* を $0.0000 に設定します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyadd、dbmnysub、dbmnydivide、dbmnyminus、dbmny4add、dbmny4sub、dbmny4mul、dbmny4divide、dbmny4minus

dbmnyndigit

説明

DBMONEY 値の右端の桁を DBCHAR として返します。

構文

RETCODE dbmnyndigit(dbproc, mnyptr, value, zero)

```
DBPROCESS    *dbproc;
DBMONEY      *mnyptr;
DBCHAR       *value;
DBBOOL       *zero;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr

DBMONEY 値を指すポインタです。dbmnyndigit の呼び出しのたびにこの値が 10 で除算され、その結果が ***mnyptr* に返されます。

value

DBMONEY 値の右端の桁の文字表現を格納する DBCHAR 変数を指すポインタです。

zero

DBBOOL 変数を指すポインタです。dbmnyndigit は、呼び出されるたびに ***mnyptr* を 10 で除算し、除算の余りの文字表現を ***value* に格納します。除算の結果が $0.0000 の場合、dbmnyndigit は ***zero* を「true」に設定します。それ以外の場合は、***zero* を「false」に設定します。*zero* を NULL として渡した場合、この情報は返されません。

戻り値

SUCCEED または FAIL。

dbmnyndigit は、*mnyptr* または *value* が NULL の場合に FAIL を返します。

使用法

- dbmnyndigit は、DBMONEY 値の右端の桁を DBCHAR として返します。
- dbmnyndigit は、DBMONEY 値を 10 で除算し、除算の余りの文字表現を ***value* に格納し、***mnyptr* を除算の結果で置き換えます。除算の結果が $0.0000 の場合、dbmnyndigit は ***zero* を「true」に設定します。

- DBMONEY 値のすべての桁を取得するには、**zero* が「true」になるまで dbmnydigit を繰り返し呼び出します。
- dbmnyinit と dbmnyndigit は、カスタムの DBMONEY-DBCHAR 変換ルーチンを作成するときに役立ちます。このようなカスタムのルーチンは、dbconvert の DBMONEY-to-DBCHAR 変換の精度 (100 分の 1 ドル単位) では十分ではない場合に役立ちます。また、dbconvert は、カンマを含む文字列を作成しません。
- dbmnyinit は、文字への変換を行うために DBMONEY 値を初期化します。このとき、不要な桁を除去し、負の数値を正の数値に変換します。詳細については、dbmnyinit のリファレンス・ページを参照してください。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。
- 次のコードは、dbmnyndigit と dbmnyinit の例です。

```
/*
 ** This example demonstrates dbmnyinit() and
 ** dbmnyndigit(). It is a conversion routine which
 ** converts a DBMONEY value to a character string.
 ** The conversion provided by this routine is unlike
 ** the conversion provided by dbconvert() in that the
 ** resulting character string includes commas. This
 ** conversion provides precision of two digits after
 ** the decimal point.
 **
 ** For simplicity, the example assumes that all
 ** routines succeed and all parameters passed to it
 ** are valid.
 */

 #define PRECISION       2

 RETCODE     new_mnytochar(mnyptr, buf_ptr)
 DBMONEY     *mnyptr;
 char        *buf_ptr;
 {
     DBMONEY     local_mny;
     DBBOOL      negative;
     int         bytes_written;
     DBCHAR      value;
     DBBOOL      zero;
     int         ret;
```

```
char        temp_buf[32];

/*
** Since dbmnyinit() and dbmnyndigit() modify the
** DBMONEY value passed to it, and since we do
** not want to modify the DBMONEY value passed
** to us by the user we need to make a local copy.
*/
ret = dbmnycopy((DBPROCESS *)NULL, mnyptr,
     &local_mny);
/* The value of 'ret' should be checked */

/*
** Next we need to call dbmnyinit().
**
** dbmnyinit() eliminates any unwanted precision
** from the DBMONEY value. DBMONEY values are
** stored with accuracy to four digits after the
** decimal point. For this conversion routine we
** only want accuracy to two digits after the
** decimal.
**
** Passing a value of 2 for the second parameter
** eliminates those two digits of precision we do
** not care about.
**
** dbmnyinit() also turns negative DBMONEY values
** into positive DBMONEY values. The value of
** negative is set to TRUE if dbmnyinit() turns a
** negative DBMONEY value into a positive DBMONEY
** value.
**
**       NOTE:dbmnyinit() eliminates unwanted by
** precision by dividing DBMONEY values by a
** power of ten. In this conversion routine it
** divides by 100. If we pass dbmnyinit() a
** DBMONEY value of $1534.1277 the resulting
** DBMONEY value is $15.3413.
*/
negative = FALSE;
ret = dbmnyinit((DBPROCESS *)NULL, &local_mny,
    4 - PRECISION, &negative);
/* The value of 'ret' should be checked */

/*
** dbmnyndigit() extracts the rightmost digit out
```

```
** of the DBMONEY value, converts it to a
** character, places the character into the
** variable "value", and then divides the DBMONEY
** value by 10. dbmnyndigit() sets 'zero' to TRUE
** if the result of the division is $0.0000.
**
** By calling dbmnyndigit() until 'zero' is set to
** TRUE we will be returned all the digits (from
** right to left) of the DBMONEY value.
*/
zero = FALSE;
bytes_written = 0;
while( zero == FALSE )
{
    ret = dbmnyndigit((DBPROCESS *)NULL,
    &local_mny, &value, &zero);
    /* The value of 'ret' should be checked.*/

    /*
    ** As we are getting the digits, we want to
    ** place the decimal point and commas in the
    ** proper positions ...
    */
    temp_buf[bytes_written++] = value;

    /*
    ** If zero == TRUE we got all the digits. We
    ** do not want to call
    ** check_comma_and_decimal() since we might
    ** put a comma before the leftmost digit.
    */
    if( zero == FALSE )
    {
         /*
         ** As we are getting the digits, we want
         ** to place the decimal point and commas
         ** in the proper positions ...
         */
         check_comma_and_decimal(temp_buf,
                              &bytes_written);
    }
}

/*
** If we haven't written PRECISION bytes into the
** buffer yet, pad with zeros, write the decimal
```

```
    ** point to the buffer, and write a zero after
    ** the decimal point.
    */
    pad_with_zeros(temp_buf, &bytes_written);

    /*
    ** We've written the money value into the buffer
    ** backwards. Now we have to write it the right
    ** way.
    */
    reverse_money(buf_ptr, temp_buf, bytes_written,
                  negative);

    return(SUCCEED);

}

void check_comma_and_decimal(temp_buf,
                             bytes_written)
char *temp_buf;
int  *bytes_written;
{

   static     int     comma = 0;
   static     DBBOOL  after_decimal = FALSE;

   if( after_decimal )
    {
        /*
        ** When comma is 3 it is time to write a
        ** comma. We do not care about commas until
        ** after we've written the decimal point.
        */
        comma++;
    }

    /*
    ** After we've written PRECISION bytes into the
    ** buffer, it's time to write the decimal point.
    */
    if( *bytes_written == PRECISION )
    {
      temp_buf[(*bytes_written)++] = '.';
      after_decimal = TRUE;
    }
```

```
    /*
    ** When (comma == 3) that means we've written three
    ** digits and it's time to put a comma into the
    ** buffer.
    */
    if( comma == 3 )
    {
      temp_buf[(*bytes_written)++] = ',';
      comma = 0;        /* clear comma */
    }

}

void    pad_with_zeros(temp_buf, bytes_written)
char *temp_buf;
int  *bytes_written;
{

    /* If we haven't written PRECISION bytes into the
    ** buffer yet, pad with zeros, write the decimal
    ** point to the buffer, and write a zero after the
    ** decimal point.
    */
    while( *bytes_written < PRECISION )
    {
        temp_buf[(*bytes_written)++] = '0';
    }

    if( *bytes_written == PRECISION )
    {
        temp_buf[(*bytes_written)++] = '.';
        temp_buf[(*bytes_written)++] = '0';
    }

          }

void reverse_money(char_buf, temp_buf,
               bytes_written, negative)
char     *char_buf;
char     *temp_buf;
int      bytes_written;
DBBOOL   negative;
{

    int       i;
```

```
        /*
        ** We've written the money value into the buffer
        ** backwards. Now we have to write it the right
        ** way. First check to see if we need to write a
        ** negative sign, then write the dollar sign,
        ** finally write the money value.
        */
        i = 0;
        if( negative == TRUE )
        {
            char_buf[i++] = '-';
        }

       char_buf[i++] = '$';

        while( bytes_written-- )
        {
            char_buf[i++] = temp_buf[bytes_written];
        }
        /* Append null-terminator:*/
        char_buf[i] = '¥0';

    }
```

参照 dbconvert、dbmnyinit

dbmnyscale

説明 DBMONEY 値に正の整数を乗算して、指定の金額を加算します。

構文 RETCODE dbmnyscale(dbproc, mnyptr, multiplier, addend)

```
DBPROCESS    *dbproc;
DBMONEY      *mnyptr;
int          multiplier;
int          addend;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr
乗算を行う DBMONEY 値を指すポインタです。**mnyptr* には、dbmnyscale 演算の結果も格納されます。

multiplier
**mnyptr* に乗算する整数です。*multiplier* は、1 以上 65535 以下の正の数値にしてください。

addend
乗算後に **mnyptr* に加算される、10,000 分の 1 ドル単位の数を表す整数です。

戻り値

SUCCEED または FAIL。

dbmnyscale は、*mnyptr* が NULL の場合、オーバフローが発生している場合、または *multiplier* が 1 から 65535 までの範囲でない場合に FAIL を返します。

使用法

- dbmnyscale は、DBMONEY 値に短整数を乗算し、*addend* (10,000 分の 1 ドル) を加算して、その結果を元の DBMONEY 変数に返します。
- *multiplier* は、1 以上 65535 以下でなければなりません。
- オーバフローの場合、dbmnyscale は FAIL を返します。この場合、**mnyptr* は不定です。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnydown、dbmnymul、dbmny4mul

dbmnysub

説明

DBMONEY 値から別の DBMONEY 値を減算します。

構文

RETCODE dbmnysub(dbproc, m1, m2, difference)

```
DBPROCESS    *dbproc;
DBMONEY      *m1;
DBMONEY      *m2;
DBMONEY      *difference;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

m1

減算される DBMONEY 値を指すポインタです。

m2

減算する DBMONEY 値を指すポインタです。

difference

減算の結果を保持する DBMONEY 変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnysub は、オーバフローの場合、または *m1*、*m2*、*difference* のいずれかが NULL の場合に FAIL を返します。

使用法

- dbmnysub は、*m1* の DBMONEY 値から *m2* の DBMONEY 値を減算し、その結果を **difference* に格納します。
- オーバフローの場合、dbmnysub は FAIL を返し、*difference* を $0.0000 に設定します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmnyadd、dbmnymul、dbmnydivide、dbmnyminus、dbmny4add、dbmny4sub、dbmny4mul、dbmny4divide、dbmny4minus

dbmnyzero

説明

DBMONEY 値を $0.0000 に初期化します。

構文

```
RETCODE dbmnyzero(dbproc, mnyptr)

DBPROCESS    *dbproc;
DBMONEY      *mnyptr;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

このパラメータには NULL を指定できます。DBPROCESS は、アプリケーションのエラー・ハンドラに対するパラメータとして使用されます。この構造体には、エラー・メッセージをどの言語で出力するかについての情報も含まれています。DBPROCESS を指定しない場合は、デフォルトの各国言語が使用されます。

mnyptr

初期化する DBMONEY 値を指すポインタです。

戻り値

SUCCEED または FAIL。

dbmnyzero は、*mnyptr* が NULL の場合に FAIL を返します。

使用法

- dbmnyzero は、DBMONEY 値を $0.0000 に初期化します。
- 有効な DBMONEY 値の範囲は、+/-$922,337,203,685,477.5808 の間です。DBMONEY 値の精度は、10,000 分の 1 ドルです。

参照

dbmny4zero

dbmonthname

説明

指定された月の、指定された言語での名前を返します。

構文

```
char *dbmonthname(dbproc, language, monthnum,
        shortform)

DBPROCESS    *dbproc;
char         *language;
int          monthnum;
DBBOOL       shortform;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

language
要求する言語の名前です。

monthnum
要求する月の番号です。月の番号の範囲は 1 (1 月) から 12 (12 月) までです。

shortform
省略形とフルネームのどちらの月名を使用するかを指示するブール値です。*shortform* が「true」の場合、dbmonthname は月の名前を省略形で返します。*shortform* が「false」の場合、dbmonthname は月の名前をフルネームで返します。たとえば、1 月の名前を英語の省略形で要求した場合は、「Jan」が返されます。

省略形の月名は、各ローカライゼーション・ファイル内に定義されています。

戻り値

正常に終了した場合は指定した月の名前です。エラーの場合は NULL ポインタです。

使用法

- dbmonthname は、指定された月の名前を指定された言語で返します。言語が指定されていない (*language* が NULL である) 場合は、*dbproc* の現在の言語が使用されます。*language* と *dbproc* の両方が NULL の場合は、DB-Library のデフォルト言語があれば、その言語が使用されます。
- 次のコードは、dbmonthname の例です。

```
for (monthnum = 1; monthnum <= 12; monthnum++)
printf("Month %d:%s¥n", monthnum,
dbmonthname((DBPROCESS *)NULL,
char *)NULL,  monthnum, TRUE),
dbmonthname((DBPROCESS *)NULL,
(char *)NULL, monthnum, FALSE));
```

参照

db12hour、dbdateorder、dbdayname、DBSETLNATLANG、dbsetopt

DBMORECMDS

説明　処理するコマンドがまだあるかどうかを示します。

構文

```
RETCODE DBMORECMDS(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　SUCCEED または FAIL で、コマンド・バッチからの結果がこれ以上あるかどうかを示します。

使用法

- アプリケーションは、このマクロを使用して、処理する結果がまだあるかどうかを判断できます。
- DBMORECMDS は、dbnextrow から NO_MORE_ROWS が返された後に呼び出すことができます。現在のコマンドがローを返さないことが確実な場合は、dbresults のすぐ後に DBMORECMDS を呼び出すことができます。
- アプリケーションは、NO_MORE_RESULTS が返されるまで dbresults を繰り返し呼び出すことができるため、このルーチンを必要とすることはほとんどありません。

参照　DBCMDROW、dbresults、DBROWS、DBROWTYPE

dbmoretext

説明　text 値または image 値の一部をサーバに送信します。

構文

```
RETCODE dbmoretext(dbproc, size, text)

DBPROCESS    *dbproc;
DBINT        size;
BYTE         *text;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

size
text または image の値のうち、サーバに送信する部分のバイト単位のサイズです。dbwritetext 呼び出しの指定より大きな text または image 値をサーバに送信するとエラーになります。

text
書き出す text または image の部分を指すポインタです。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、大きな SYBTEXT 値または SYBIMAGE 値を多数の小さなまとまりの形でサーバに送信するときに、dbwritetext とともに使用します。これは、非常に大きなデータ・バッファを割り付けることができないオペレーティング・システムの場合に特に有効です。
- dbmoretext と dbwritetext は更新のときにだけ使用され、Transact-SQL の update 文の代わりとなります。
- dbsqlok と dbresults を呼び出すのは、dbmoretext の最初の呼び出しの前と、dbmoretext の最後の呼び出しの後でなければなりません。
- 詳細については、dbwritetext のリファレンス・ページを参照してください。
- DB-Library/C の DBTEXTSIZE オプションは、サーバのグローバル変数 *@@textsize* の値に反映されます。この変数は、サーバが返す text 値または image 値のサイズを制限するものです。*@@textsize* のデフォルト値は 32,768 バイトです。アプリケーションで 32,768 バイトを超える text 値または image 値を取得する場合は、dbsetopt を呼び出して *@@textsize* を大きくする必要があります。

  DB-Library/C の DBTEXTLIMIT オプションは、DB-Library/C が読み込む text 値または image 値のサイズを制限します。

参照

dbtxptr、dbtxtimestamp、dbwritetext

dbmsghandle

説明

サーバ・メッセージを処理するユーザ関数をインストールします。

構文

```
int (*dbmsghandle(handler))()

int        (*handler)();
```

パラメータ

ハンドラ

DB-Library がサーバからのエラー・メッセージまたは情報メッセージを受信したときに呼び出されるユーザ関数を指すポインタです。DB-Library は、表 2-21 に示す 8 つのパラメータを指定してこの関数を呼び出します。

表 2-21：メッセージ・ハンドラ・パラメータ

パラメータ	意味
dbproc	影響を受ける DBPROCESS。
msgno	現在のメッセージ番号 (データ型 DBINT)。この番号は sysmessages テーブルに登録されている。
msgstate	現在のメッセージのエラー・ステータス番号 (データ型 *int*)。この番号は、Sybase 製品の保守契約を結んでいるサポート・センタがエラーのコンテキストに関する情報を取得するために必要となる。
severity	現在のメッセージの情報クラスまたはエラー重大度 (データ型 *int*)。この番号に関する説明は、Adaptive Server Enterprise のマニュアルに記載されている。
msgtext	現在のメッセージの NULL で終了するテキスト (データ型 *char* *)。
srvname	メッセージを生成したサーバの NULL で終了する名前 (データ型 *char* *)。サーバ名は、システム・テーブル sysservers の srvname カラム内に格納されています。このサーバ名は、サーバ間の通信に使用される。特に、サーバが他のサーバにログインしてリモート・プロシージャ・コールを実行するときに使用される。サーバに名前がない場合、*srvname* の長さは 0 になる。
procname	メッセージを生成したストアド・プロシージャの NULL で終了する名前 (データ型 char *)。メッセージがストアド・プロシージャによって生成されたものではない場合は、*procname* の長さは 0 になる。
line	メッセージを生成したコマンド・バッチまたはストアド・プロシージャの行番号 (データ型 *int*)。行番号は 1 から始まる。行番号は、メッセージが生成されたネスト・レベルごとにカウントされる。たとえば、あるコマンド・バッチがストアド・プロシージャ A を実行し、そのストアド・プロシージャがストアド・プロシージャ B を呼び出す場合に、メッセージが B の 3 行目で生成されたときは、*line* の値は 3 になる。メッセージに対応した行番号がない場合、*line* は 0 になる。行番号のないメッセージが生成される状況には、ログイン・エラーや、存在しないストアド・プロシージャに対するリモート・プロシージャ・コール (dbrpcsend で実行) などがある。

メッセージ・ハンドラは、DB-Library に 0 の値を返す必要があります。

Windows では、次の例に示すようにメッセージ・ハンドラは CS_PUBLIC で宣言してください。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。

次の例は、標準的なメッセージ・ハンドラのルーチンを示します。

```
#include    <sybfront.h>
#include    <sybdb.h>

 int CS_PUBLIC msg_handler(dbproc, msgno, msgstate,
     severity, msgtext, srvname, procname, line)

 DBPROCESS       *dbproc;
 DBINT           msgno;
 int             msgstate;
 int             severity;
 char            *msgtext;
 char            *srvname;
 char            *procname;
 int             line;

 {
      printf ("Msg %ld, Level %d, State %d¥n",
          msgno, severity, msgstate);
       if (strlen(srvname) > 0)
           printf ("Server '%s', ", srvname);
       if (strlen(procname) > 0)
           printf ("Procedure '%s', ", procname);
       if (line > 0)
           printf ("Line %d", line);

       printf("¥n¥t%s¥n", msgtext);

       return (0);
 }
```

戻り値

インストール済みのメッセージ・ハンドラを指すポインタです。メッセージ・ハンドラがまったくインストールされていない場合は、NULL です。

使用法

- dbmsghandle は、ユーザ提供のメッセージ・ハンドラ関数をインストールします。DB-Library は、サーバ・エラーまたは情報メッセージを受け取ると、ただちにこのメッセージ・ハンドラを呼び出します。サーバ・メッセージを適切に処理するには、メッセージ・ハンドラをインストールしてください。
- アプリケーションが dbmsghandle を呼び出してメッセージ・ハンドラ関数をインストールしていない場合は、DB-Library はサーバ・メッセージを無視します。メッセージは出力されません。

- コマンド・バッファにコマンドが 1 つしか含まれていない場合に、そのコマンドがサーバ・メッセージを引き起こしたときは、DB-Library は dbsqlexec の実行中にメッセージ・ハンドラを呼び出します。コマンド・バッファに複数のコマンドがあるときに、最初のコマンドに問題がなければ、実行時エラーによって dbsqlexec が失敗することはありません。代わりに、実行時エラーを引き起こしたコマンドを処理する dbresults 呼び出しが失敗します。
- NULL パラメータを指定して dbmsghandle を呼び出すことによって、既存のメッセージ・ハンドラのインストールを「解除」できます。また、いつでも新しいメッセージ・ハンドラをインストールできます。新しいメッセージ・ハンドラは、自動的に既存のメッセージ・ハンドラと置き換わります。
- サーバ・メッセージのリストについては、sysmessages テーブルを参照してください。また、Transact-SQL の print コマンドと raiserror コマンドが生成するメッセージは、dbmsghandle によって検出されます。
- dbsetuserdata ルーチンと dbgetuserdata ルーチンは、メッセージ・ハンドラとそれをトリガしたプログラム・コードとの間で情報を転送する必要があるときに、特に便利です。この方法を使用してデッドロックを処理する例については、dbsetuserdata のリファレンス・ページを参照してください。
- もう 1 つのルーチンである dberrhandle は、DB-Library が DB-Library エラーに応答して呼び出すエラー・ハンドラをインストールします。
- アプリケーションが DB-Library からのメッセージとサーバからのメッセージを同時に発生させた場合、DB-Library は、DB-Library エラー・ハンドラを呼び出す前にサーバ・メッセージ・ハンドラを呼び出します。
- DB-Library/C のエラー値 SYBESMSG は、サーバ・エラー・メッセージに応答して生成されますが、サーバ情報メッセージに応答して生成されることはありません。したがって、サーバ・エラーが発生すると、サーバ・メッセージ・ハンドラと DB-Library/C エラー・ハンドラの両方が呼び出されますが、サーバが情報メッセージを生成したときには、サーバ・メッセージ・ハンドラだけが呼び出されます。

  サーバ・メッセージ・ハンドラをインストールした場合は、同じエラーが二度ユーザに通知されないようにするために、DB-Library エラー・ハンドラでの SYBESMSG エラー出力の抑制もできます。
- 表 2-22 は、DB-Library/C がいつアプリケーションのメッセージ・ハンドラとエラー・ハンドラを呼び出すかをまとめたものです。

表 2-22 : DB-Library がメッセージ・ハンドラとエラー・ハンドラを呼び出すタイミング

エラーまたはメッセージ	メッセージ・ハンドラを呼び出す	エラー・ハンドラを呼び出す
SQL 構文エラー	あり	はい (SYBESMSG)。(メッセージを無視するようにハンドラをコーディングする。)
SQL print 文	あり	いいえ。
SQL raiserror	あり	いいえ。
サーバが停止している	なし	はい (SYBESEOF)。(アプリケーションを終了するようにハンドラをコーディングする。)
サーバからのタイムアウト	なし	はい (SYBETIME)。(再びタイムアウト期間が経過するまで待つために、INT_CONTINUE を返すようにハンドラをコーディングする。)
クエリ時のデッドロック	あり	いいえ。(デッドロックの有無を調べるようにハンドラをコーディングする。)
ログイン時のタイムアウト	なし	はい (SYBEFCON)。
ログインの失敗 (dbopen)	あり	はい (SYBEPWD)。(アプリケーションを終了するようにハンドラをコーディングする。)
データベース・メッセージの使用	あり (メッセージを無視するようにハンドラをコーディングする。)	いいえ。
必要なときに dbresults を呼び出さないなど、DB-Library/C 呼び出しの使用法の誤り	なし	はい (SYBERPND)。
致命的なサーバ・エラー (重大度 17 以上)	あり	はい (SYBESMSG)。

参照　dberrhandle、dbgetuserdata、dbsetuserdata

dbname

説明

現在のデータベースの名前を返します。

構文

```
char *dbname(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

現在のデータベースの NULL で終了する名前を指すポインタです。

使用法

- dbname は、現在のデータベース名を返します。
- データベースがいつ変更されたかを追跡する必要がある場合は、dbchange を使用してください。

参照

dbchange、dbuse

dbnextrow

説明

ロー・バッファ、およびカラム・データにバインドされるプログラム変数に、次の結果ローを読み込みます。

構文

```
STATUS dbnextrow(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

dbnextrow は、次の値を返します。

- 通常ローが読み込まれた場合は、REG_ROW が返されます。通常ローとは、クエリの where 句と一致するローです。

- 計算ローが読み込まれた場合は、*computeid* が返されます。計算ローとは、compute 句が生成するローです。*computeid* は読み込まれた計算ローの番号に一致し、最初の計算ローが 1、2 番目のローが 2 というようになります。*computeid* は、この関数の他の種類の戻り値とは一致しません。
- バッファリングがオンのときに、次のローを読み込むことによってバッファの容量を超過してしまう場合は、BUF_FULL が返されます。この場合、ローは読み込まれていません。さらにローを読み込むには、dbclrbuf を呼び出して、ロー・バッファの先頭から 1 つ以上のローを削除する必要があります。
- 結果セットの最後のローが読み込まれた場合は、NO_MORE_ROWS が返されます。クエリがローを生成しなかった場合 (たとえば、update や insert の場合、または select に一致するものがなかった場合) は、dbnextrow を初めて呼び出したときに NO_MORE_ROWS が返されます。また、クエリが失敗した場合や保留中の結果がない場合も、dbnextrow この値が返されます。
- ネットワーク・エラーやメモリ不足のエラーなどの異常事態が発生したためにルーチンが正常に完了できなかった場合は、FAIL が返されます。

使用法

- dbnextrow は、サーバから最初に返されたローから順に、結果データの次のローを読み込みます。通常、次の結果ローはサーバから直接読み込まれます。DBBUFFER オプションがオンで、dbgetrow の呼び出しによってローがランダムに読み込まれている場合は、次のローはサーバから読み込まれるのではなく、バッファリングされたローを連結したリストから読み込まれます。dbnextrow が呼び出されると、dbbind または dbaltbind で指定されたロー・データからプログラム変数へのバインドが有効になります。
- プログラム変数がカラムにバインドされている場合は、dbnextrow はバインドされた変数に新しい値を書き込んでから呼び出し元に戻ります。
- 通常ローのカラム値は、dbdata を使用して取り出すか、dbbind を使用してプログラム変数にバインドします。計算ローのカラム値は、dbadata を使用して取り出すか、dbaltbind を使用してプログラム変数にバインドします。
- アプリケーションで dbnextrow を呼び出すには、その前に dbresults から SUCCEED が返されている必要があります。コマンドがローを返すものである場合に、dbnextrow を使用して結果を処理する必要があるかどうかを判断するには、dbresults の後で DBROWS を呼び出してください。

- アプリケーションで dbresults を呼び出した後は、dbcanquery または dbcancel を呼び出して現在の結果セットをキャンセルすることも、dbnextrow をループ内で呼び出して結果のローを 1 つずつ処理することもできます。
- アプリケーションで結果を処理することを選択した場合は、次のいずれかを行うことができます。
  - NO_MORE_ROWS が返されるまで、ループで dbnextrow を呼び出して、すべての結果ローを処理する。NO_MORE_ROWS が返されたときは、dbresults をもう一度呼び出し、次の結果がある場合はその結果を処理するための設定を実行できます。
  - dbnextrow を呼び出して結果ローの一部を処理してから、dbcancel (コマンド・バッチまたは RPC 呼び出しのすべての結果をキャンセルする場合) または dbcanquery (最後の dbresults 呼び出しの結果だけをキャンセルする場合) を呼び出して残りの結果ローをキャンセルする。

  アプリケーションは、すべての結果ローを、キャンセルするか処理しなければなりません。
- 標準的な呼び出しのシーケンスを次に示します。

```
DBINT     xvariable;
DBCHAR    yvariable[10];

/* Read the query into the command buffer */
dbcmd(dbproc, "select x = 100, y = 'hello'");

/* Send the query to Adaptive Server Enterprise */
dbsqlexec(dbproc);

/* Get ready to process the query results */
dbresults(dbproc);

/* Bind column data to program variables */
dbbind(dbproc, 1, INTBIND, (DBINT) 0,
   (BYTE *) &xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
    yvariable);

/* Now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    C-code to print or process row data
}
```

- サーバは 2 つのタイプのローを返します。select 文の select リストで指定されたカラムのデータが含まれる通常ローと、compute 句から生成される計算ローです。サーバからの結果ローの処理を簡単にするために、dbnextrow は、ローのタイプに応じて異なる値を返します。詳細については、このリファレンス・ページの「戻り値」の項を参照してください。
- デフォルトの出力デバイスにサーバの結果データを表示するには、dbnextrow の代わりに dbprrow を使用します。

参照　dbaltbind、dbbind、dbcanquery、dbclrbuf、dbgetrow、dbprrow、dbsetrow、「オプション」(436 ページ)

dbnpcreate

説明　ノーティフィケーション・プロシージャを作成します。

構文

```
RETCODE dbnpcreate(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　SUCCEED または FAIL。

使用法

- dbnpcreate は、ノーティフィケーション・プロシージャを作成します。ノーティフィケーション・プロシージャは、特殊なタイプの Open Server レジスタード・プロシージャです。ノーティフィケーション・プロシージャは、通常の Open Server レジスタード・プロシージャとは異なり、実行可能な文は含まれません。ノーティフィケーション・プロシージャは、DB-Library/C アプリケーションが作成できる唯一の Open Server レジスタード・プロシージャのタイプです。
- ノーティフィケーション・プロシージャの名前とそのパラメータは、dbnpdefine と dbregparam を使用してあらかじめ定義しておく必要があります。
- DB-Library/C アプリケーションでノーティフィケーション・プロシージャを作成するには、次のことを行う必要があります。

  - dbnpdefine を使用してプロシージャを定義する
  - プロシージャのパラメータがある場合は、dbregparam を使用してそのパラメータを記述する
  - dbnpcreate を使用してプロシージャを作成する
- レジスタード・プロシージャに適用されるすべての DB-Library/C ルーチンは、ノーティフィケーション・プロシージャにも適用されます。たとえば、dbregexec はレジスタード・プロシージャを実行しますが、そのプロシージャはノーティフィケーション・プロシージャであってもかまいません。同様に、dbreglist は OpenServer 内に現在定義されているすべてのレジスタード・プロシージャのリストを返しますが、その中にノーティフィケーション・プロシージャが含まれることもあります。
- アプリケーションは、ノーティフィケーション・プロシージャが実行されたときに通知 (ノーティフィケーション) を受け取ることを要求できるので、ノーティフィケーション・プロシージャは、他のレジスタード・プロシージャと同様に、アプリケーション間の通信や同期に便利です。
- ノーティフィケーション・プロシージャを作成できるのは、Open Server 内だけです。現時点では、Adaptive Server Enterprise はノーティフィケーション・プロシージャをサポートしていません。
- DB-Library/C アプリケーションで、レジスタード・プロシージャの実行のノーティフィケーションを要求するには、dbregwatch を使用します。アプリケーションは、同期と非同期のどちらのノーティフィケーションを受け取るかを要求できます。
- 次に、ノーティフィケーション・プロシージャを作成する例を示します。

```
DBPROCESS    *dbproc;
DBINT       status;

/*
** Let's create a notification procedure called
** "message" which has two parameters:
**     msg     varchar(255)
**     user    idint
*/

/*
** Define the name of the notification procedure
** "message"
```

```
*/
dbnpdefine (dbproc, "message", DBNULLTERM);

/*
** The notification procedure has two parameters:
**      msg     varchar(255)
**      user    idint
** So, define these parameters. Note that
** neither of the parameters is defined with a
** default value.
*/
dbregparam (dbproc, "msg", SYBVARCHAR,
     DBNODEFAULT, NULL);
dbregparam (dbproc, "userid", SYBINT4,
     DBNODEFAULT, 4);

/*  Create the notification procedure:*/
status = dbnpcreate (dbproc);
if (status == FAIL)
{
    fprintf(stderr, "ERROR:Failed to create ¥
     message!¥n");
}
else
{
    fprintf(stdout, "Success in creating ¥
     message!¥n");
}
```

参照　dbreginit、dbregparam、dbregwatch、dbregnowatch

dbnpdefine

説明　ノーティフィケーション・プロシージャを定義します。

構文　RETCODE dbnpdefine(dbproc, procedure_name, namelen)

```
DBPROCESS    *dbproc;
DBCHAR       *procedure_name;
DBSMALLINT   namelen;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

procedure_name
定義するノーティフィケーション・プロシージャの名前を指すポインタです。

namelen
procedure_name のバイト単位の長さです。*procedure_name* が NULL で終了する場合は、*namelen* を DBNULLTERM として渡してください。

戻り値

SUCCEED または FAIL。

使用法

- dbnpdefine は、ノーティフィケーション・プロシージャを定義します。ノーティフィケーション・プロシージャの定義は、ノーティフィケーション・プロシージャ作成の最初のステップです。
- ノーティフィケーション・プロシージャは、特殊なタイプの Open Server レジスタード・プロシージャです。ノーティフィケーション・プロシージャは、通常の Open Server レジスタード・プロシージャとは異なり、実行可能な文は含まれません。ノーティフィケーション・プロシージャは、DB-Library/C アプリケーションが作成できる唯一の Open Server レジスタード・プロシージャのタイプです。
- DB-Library/C アプリケーションでノーティフィケーション・プロシージャを作成するには、次のことを行う必要があります。
  - dbnpdefine を使用してプロシージャを定義する
  - プロシージャのパラメータがある場合は、dbregparam を使用してそのパラメータを記述する
  - dbnpcreate を使用してプロシージャを作成する
- レジスタード・プロシージャに適用されるすべての DB-Library/C ルーチンは、ノーティフィケーション・プロシージャにも適用されます。たとえば、dbregexec はレジスタード・プロシージャを実行しますが、そのプロシージャはノーティフィケーション・プロシージャであってもかまいません。同様に、dbreglist は OpenServer 内に現在定義されているすべてのレジスタード・プロシージャのリストを返しますが、その中にノーティフィケーション・プロシージャが含まれることもあります。

- 次に、ノーティフィケーション・プロシージャを定義する例を示します。

```
DBPROCESS    *dbproc;
DBINT        status;

/*
** Let's create a notification procedure called
** "message" which has two parameters:
**      msg     varchar(255)
**       userid  int
*/

/*
** Define the name of the notification procedure
** "message"
*/
dbnpdefine (dbproc, "message", DBNULLTERM);

/* The notification procedure has two parameters:
**      msg     varchar(255)
**       userid  int
** So, define these parameters. Note that
** neither of the parameters is defined with a
** default value.
*/
dbregparam (dbproc, "msg", SYBVARCHAR,
     DBNODEFAULT, NULL);
dbregparam (dbproc, "userid", SYBINT4,
     DBNODEFAULT, 4);

/*  Create the notification procedure:*/
status = dbnpcreate (dbproc);
if (status == FAIL)
{
     fprintf(stderr, "ERROR:Failed to create ¥
       message!¥n");
}
else
{
     fprintf(stdout, "Success in creating ¥
       message!¥n");
}
```

参照　dbregparam、dbnpcreate、dbreglist

dbnullbind

説明

インジケータ変数を通常結果ローのカラムに関連付けます。

構文

RETCODE dbnullbind(dbproc, column, indicator)

```
DBPROCESS    *dbproc;
int          column;
DBINT        *indicator;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
インジケータ変数を関連付けるカラムの番号です。

indicator
インジケータ変数を指すポインタです。

戻り値

SUCCEED または FAIL。

dbnullbind は、*column* が無効の場合に FAIL を返します。

使用法

- dbnullbind は、通常の結果ローのカラムにインジケータ変数を関連付けます。インジケータ変数は、通常結果ローの特定のカラムがプログラム変数に正しく変換されてコピーされているかどうか、またはそのカラムが NULL であるかどうかを示します。
- インジケータ変数は、通常結果ローが dbnextrow によって処理されるときに設定されます。設定される値は次のとおりです。
  - カラムが NULL の場合は -1 です。
  - *column* が dbbind によってプログラム変数にバインドされているけれども、そのバインドでデータ変換が指定されていない場合に、*column* のデータを保持するにはプログラム変数が小さすぎてバインドされるデータがトランケートされたときは、カラムのデータのバイト単位の全長です。
  - *column* がプログラム変数に正常にバインドされてコピーされた場合は、0 です。

注意　文字列がトランケートされたことの検出は、CHARBIND と VARYCHARBIND のみを対象として実装されています。

参照

dbanullbind、dbbind、dbdata、dbdatlen、dbnextrow

dbnumalts

説明　計算ローのカラム数を返します。

構文　int dbnumalts(dbproc, computeid)

```
DBPROCESS    *dbproc;
int          computeid;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

computeid
　対象となる特定の計算ローを表す ID です。1 つの SQL select 文で複数の compute 句を使用でき、各句は、それぞれ別の計算ローを返します。select 文の最初の compute 句に対応する *computeid* は 1 です。*computeid* は、dbnextrow または dbgetrow から返されます。

戻り値　特定の computeid に対するカラムの数です。*computeid* が正しくない場合、dbnumalts は -1 を返します。

使用法　dbnumalts は、計算ローのカラムの数を返します。アプリケーションは、dbresults が SUCCEED を返した後に、このルーチンを呼び出すことができます。たとえば、以下の SQL 文では、dbnumalts(*dbproc*, 1) の呼び出しにより 3 が返されます。

```
select dept, year, sales from employee
order by dept, year
compute avg(sales), min(sales),
max(sales) by dept
```

参照　dbadata、dbadlen、dbaltlen、dbalttype、dbgetrow、dbnextrow、dbnumcols

dbnumcols

説明　現在の結果セットの通常カラムの数を返します。

構文　int dbnumcols(dbproc)

```
DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

現在の結果セット内のカラム数です。カラムがない場合、dbnumcols は 0 を返します。

使用法

- dbnumcols は、現在の結果セット内の通常 (非計算) カラムの数を返します。
- 次のコードは、dbnumcols の使用例です。

```
int         column_count;
 DBPROCESS      *dbproc;

 /* Put the commands into the command buffer */
 dbcmd(dbproc, "select name, id, type from ¥
    sysobjects");
 dbcmd(dbproc, "select name from sysobjects");

 /*
 ** Send the commands to Adaptive Server Enterprise
and start
 ** execution
 */
 dbsqlexec(dbproc);

 /* Process each command until there are no more */
 while (dbresults(dbproc) != NO_MORE_RESULTS)
 {
    column_count = dbnumcols(dbproc);
    printf("%d columns in this Adaptive Server
Enterprise ¥
      result.¥n", column_count);
    while (dbnextrow(dbproc) != NO_MORE_ROWS)
      printf("row received.¥n");
 }
```

参照

dbcollen、dbcolname、dbnumalts

dbnumcompute

説明　現在の結果セットの compute 句の数を返します。

構文

```
int dbnumcompute(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　現在の結果セットの compute 句の数です。

使用法　このルーチンは、現在の結果セットの compute 句の数を返します。アプリケーションは、dbresults が SUCCEED を返した後に、このルーチンを呼び出すことができます。たとえば、次の SQL 文では、select 文には compute 句が 2 つあるので、dbnumcompute(*dbproc*) を呼び出すと 2 が返されます。

```
select dept, name from employee
order by dept, name
compute count(name) by dept
compute count(name)
```

参照　dbnumalts、dbresults

DBNUMORDERS

説明　Transact-SQL select 文の order by 句で指定されたカラムの数を返します。

構文

```
int DBNUMORDERS(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　order by カラムの数です。order by 句がない場合、このルーチンは 0 を返します。エラーがある場合は -1 を返します。

使用法

select 文が実行され、その結果を処理するための dbresults が呼び出された後で、DBNUMORDERS を呼び出すと、その文の order by 句で指定されているカラムの数を調べることができます。

参照

dbordercol

dbnumrets

説明

ストアド・プロシージャによって生成されたリターン・パラメータ値の数を調べます。

構文

```
int dbnumrets(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

最後に実行されたストアド・プロシージャに対応するリターン・パラメータ値の数です。

使用法

- dbnumrets は、ストアド・プロシージャに対する最後の execute 文またはリモート・プロシージャ・コールによって返されたリターン・パラメータ値の数を返します。dbnumrets によって返された数が 0 以下である場合、使用可能なリターン・パラメータはありません。
- Transact-SQL ストアド・プロシージャは、指定された「リターン・パラメータ」の値を返すことができます。ストアド・プロシージャを呼び出したアプリケーションでは、そのプロシージャ内で変更されたリターン・パラメータの値を使用できます。これは、いくつかのプログラミング言語で使用可能な「参照渡し」の機能に似ています。

- パラメータがリターン・パラメータとして機能するには、ストアド・プロシージャ内でそのように宣言されていなければなりません。ストアド・プロシージャを呼び出す execute 文およびリモート・プロシージャ・コールも同様に、そのパラメータがリターン・パラメータとして機能することを示していなければなりません。リモート・プロシージャ・コールの場合は、dbrpcparam ルーチンを使用して、パラメータがリターン・パラメータであるかどうかを指定します。
- ストアド・プロシージャを実行するとき、サーバは、他のすべての結果を返した直後にパラメータ値を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbnumrets を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションで dbnumrets、またはリターン・パラメータを処理する他のルーチンを呼び出すには、その前に、dbresults と dbnextrow を必要な回数だけ呼び出してすべての結果を処理する必要があります。
- ストアド・プロシージャをリモート・プロシージャ・コールで呼び出した場合、リターン・パラメータ値は自動的にアプリケーションで使用できるようになります。一方、ストアド・プロシージャを execute 文で呼び出した場合は、その execute 文が含まれているコマンド・バッチが定数ではなくローカル変数をリターン・パラメータとして使用している場合にだけ、リターン・パラメータ値が使用可能となります。ストアド・プロシージャからのリターン・パラメータの詳細については、『ASE リファレンス・マニュアル』を参照してください。
- 次に、リターン・パラメータ値の検索に使用するその他のルーチンを示します。
  - dbretdata は、パラメータ値を指すポインタを返します。
  - dbretlen は、パラメータ値の長さを返します。
  - dbretname は、パラメータ値の名前を返します。
  - dbrettype は、パラメータ値のデータ型を返します。
  - dbconvert は、必要に応じて値を変換するために呼び出されます。

  これらのルーチンを dbnumrets とともに使用する方法の例については、dbretdata のリファレンス・ページを参照してください。

参照

dbnextrow、dbresults、dbretdata、dbretlen、dbretname、dbrettype、dbrpcinit、dbrpcparam

dbopen

説明

DBPROCESS 構造体を作成し、初期化します。

構文

```
DBPROCESS *dbopen(login, server)

LOGINREC  *login;
char      *server;
```

パラメータ

login

LOGINREC 構造体を指すポインタです。このポインタは、引数として dbopen へ渡されます。dblogin を呼び出すことによって、LOGINREC 構造体を取得することができます。

アプリケーションの dbopen 呼び出しがすべて完了した後は、LOGINREC 構造体は必要なくなります。プログラムは、dbloginfree を呼び出して LOGINREC 構造体を解放できます。

サーバ

接続先のサーバです。*server* は、interfaces ファイル内でそのサーバに与えられたエイリアスです。dbopen は、サーバに接続するための情報を得るために interfaces ファイル内で *server* を検索します。

server が NULL の場合、dbopen は、DSQUERY 環境変数または論理名の値に対応する interfaces エントリを検索します。DSQUERY が明示的に設定されていない場合の値は「SYBASE」になります (interfaces ファイルの指定の詳細については、dbsetifile のリファレンス・ページを参照してください。『Open Client/Server 設定ガイド』を参照してください)。

注意 UNIX 以外のプラットフォームでは、クライアント・アプリケーションは UNIX *interfaces* ファイルとは異なる方法を使用して、サーバのアドレス情報を検索します。クライアントがサーバに接続する方法については、『Open Client/Server 設定ガイド』を参照してください。

戻り値

正常に終了した場合は、DBPROCESS ポインタです。通常、dbopen は、DBPROCESS 構造体を作成または初期化できなかった場合、またはサーバへのログインができなかった場合に NULL を返します。dbopen が NULL を返すと、エラーを示す DB-Library エラー番号が生成されます。アプリケーションは、エラー・ハンドラを介してこのエラー番号にアクセスできます。ただし、サーバ・ログイン処理中に予期しない通信障害が発生したけれども、エラー・ハンドラがインストールされていない場合は、プログラムはアボートします。

使用法

- このルーチンは、DBPROCESS 構造体を割り付けて初期化します。この構造体は、DB-Library がサーバとの通信に使用する基本的なデータ構造体です。ほとんどすべての DB-Library の呼び出しで、この構造体が最初の引数になります。このルーチンは、DBPROCESS 構造体の割り付けの他に、ネットワークとの通信の設定、サーバへのログイン、すべてのデフォルトのオプションの初期化を行います。
- 次のコードは、dbopen の使用例です。

```
DBPROCESS    *dbproc;
LOGINREC     *loginrec;

loginrec = dblogin();
DBSETLPWD(loginrec, "server_password");
DBSETLAPP(loginrec, "my_program");
dbproc = dbopen(loginrec, "my_server");
```

- アプリケーションは、サーバにログインした後で、dbuse ルーチンを呼び出してデータベースを変更できます。

interfaces ファイルでの複数のクエリ・エントリ

- dbopen がサーバとの接続を確立できない場合に代替サーバとの接続を確立するように、interfaces ファイルを設定できます。
- アプリケーションで、dbopen を呼び出してサーバ MARS に接続するには、次のようにします。

```
dbopen(loginrec, MARS);
```

MARS のエントリが含まれている interfaces ファイルの内容が次のとおりであるとします。

```
#
MARS
    query tcp hp-ether violet 1025
    master tcp hp-ether violet 1025
    console tcp hp-ether violet 1026
#
VENUS
    query tcp hp-ether plum 1050
    master tcp hp-ether plum 1050
    console tcp hp-ether plum 1051
#
NEPTUNE
    query tcp hp-ether mauve 1060
    master tcp hp-ether mauve 1060
```

```
        console tcp hp-ether mauve 1061
```

- アプリケーションの接続先は、マシン「violet」上のポート番号 1025 となります。MARS が使用できない場合は、dbopen 呼び出しは失敗します。ただし、interfaces ファイル内に MARS に対する複数のクエリ・エントリがある場合は、最初の接続の試みが失敗すると、dbopen は、リストにある次のサーバへの接続を自動的に試みます。そのような interfaces ファイルの例を次に示します。

```
#
MARS
    query tcp hp-ether violet 1025
    query tcp hp-ether plum 1050
    query tcp hp-ether mauve 1060
    master tcp hp-ether violet 1025
    console tcp hp-ether violet 1026
#
VENUS
    query tcp hp-ether plum 1050
    master tcp hp-ether plum 1050
    console tcp hp-ether plum 1051
#
NEPTUNE
    query tcp hp-ether mauve 1060
    master tcp hp-ether mauve 1060
    console tcp hp-ether mauve 1061
```

- MARS の 2 番目のクエリ・エントリは VENUS のクエリ・エントリと同じであり、MARS の 3 番目のクエリ・エントリは NEPTUNE のクエリ・エントリと同じであることに注意してください。この interfaces ファイルが使用される場合に、アプリケーションが MARS との接続に失敗すると、自動的に VENUS との接続が試行されます。VENUS との接続に失敗すると、自動的に NEPTUNE との接続が試行されます。サーバの interfaces ファイル・エントリに指定する代替サーバの数に制限はありませんが、代替サーバは、同じ interfaces ファイル内に指定されていなければなりません。次のように、interfaces ファイル内のサーバ名の後ろに 2 つの数値を追加できます。

```
#
 MARS retries seconds
     query tcp hp-ether violet 1025
     query tcp hp-ether plum 1050
     query tcp hp-ether mauve 1060
     master tcp hp-ether violet 1025
```

```
console tcp hp-ether violet 1026
```

retries は、最初の一巡で接続が確立できなかった場合に、クエリ・エントリのリストを繰り返しループ処理する回数を表します。*seconds* は、dbopen がリストの先頭に戻って再びループ処理を開始する前に待機する時間 (秒) を表します。これらの数値は省略可能です。これらの数値が指定されていない場合、dbopen は各クエリ・エントリへの接続を 1 回だけ試みます。リストをループして、ループとループの間で一時停止することは、候補サーバがブート処理中の場合に有効です。複数のクエリ行は、代替サーバにプライマリ・サーバのデータベースのミラーリングされたコピーがある場合に特に便利です。

エラー

次のエラーのいずれかが発生した場合、dbopen 呼び出しは NULL を返します。これらのエラーは、アプリケーションのエラー・ハンドラ (dberrhandle によってインストールされる) でトラップできます。

DLL のエントリ関数で dbopen を呼び出すと、デッドロックが発生することがあります。dbopen は、オペレーティング・システム・スレッドを作成し、システム・ユーティリティを使用してこれらのスレッドの同期を試みます。この同期化は、オペレーティング・システムの直列化プロセスと競合します。288

注意　DB-Library アプリケーションにおける SIGALARM の使用は、dbopen が失敗する原因になることがあります。

SYBEMEM	十分なメモリの割り当てができません。
SYBEDBPS	最大数の DBPROCESS がすでに割り当てられています。 アプリケーションでは、dbsetmaxprocs と dbgetmaxprocs で、DBPROCESS 構造体の最大数を設定したり、検索したりできることに注意してください。
SYBESOCK	ソケットをオープンできません。
SYBEINTF	interfaces ファイル内にサーバ名がない。
SYBEUHST	ホスト・マシン名がわかりません。
SYBECONN	接続できない：Adaptive Server Enterprise が使用できないか、存在しません。
SYBEPWD	ログインが正しくありません。
SYBEOPIN	interfaces ファイルをオープンできない。

参照 dbclose、dbexit、dbinit、dblogin、dbloginfree、dbsetifile、dbuse

dbordercol

説明 最後に実行したクエリの order by 句にあるカラムの id を返します。

構文 int dbordercol(dbproc, order)

```
DBPROCESS    *dbproc;
int          order;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

order
対象となる特定の order by カラムを表す Id です。order by 句の中で指定される最初のカラムの番号は 1 です。

戻り値 order by 句内の指定された位置にあるカラムの id です。カラム ID は、select リストの中のカラムの位置に基づいて決定します。order が無効である場合、dbordercol は -1 を返します。

使用法 このルーチンは、SQL select コマンドの order by 句内の指定された位置にあるカラムの id を返します。

たとえば、SQL 文で dbordercol(*dbproc*, 1) を呼び出すと、3 が返されます。order by 句で指定された最初のカラムは、クエリの select リストの 3 番目のカラムを参照するためです。

```
select dept, name, salary from employee
order by salary, name
```

参照 DBNUMORDERS

dbpoll

説明 サーバの応答が DBPROCESS に到着したかどうかを調べます。

構文

```
RETCODE dbpoll(dbproc, milliseconds, ready_dbproc,
        return_reason)

DBPROCESS     *dbproc;
long          milliseconds;
DBPROCESS     **ready_dbproc;
int           *return_reason;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

dbproc は、dbpoll がチェックする DBPROCESS 接続を表します。

dbproc が NULL として渡された場合、dbpoll はオープンしているすべての DBPROCESS 接続をチェックし、応答が到着している接続があるかどうかを調べます。

milliseconds

dbpoll が呼び出し元に戻る前に応答を待つ時間 (ミリ秒) の最大値です。

milliseconds に 0 が渡された場合、dbpoll は即座に呼び出し元に戻ります。

milliseconds に -1 が渡された場合、サーバの応答が到着するか、システムの割り込みが発生するまでは、dbpoll は呼び出し元に戻りません。

ready_dbproc

DBPROCESS 構造体へのポインタを指すポインタです。dbpoll は、サーバの応答が到着している DBPROCESS を指すように **ready_dbproc* を設定します。応答が到着していない場合、dbpoll は **ready_dbproc* を NULL に設定します。

注意　*ready_dbproc* は DBPROCESS ポインタではありません。DBPROCESS ポインタを指すポインタです。

---

return_reason

dbpoll が呼び出し元に戻った理由を表す整数へのポインタです。この整数は、次の記号値のいずれかです。

DBRESULT	サーバ・コマンドに対する応答が到着している。アプリケーションは、dbsqlok を呼び出してサーバの応答を調べることができる (dbsqlsend がすでに呼び出されているものとする)。
DBNOTIFICATION	レジスタード・プロシージャ・ノーティフィケーションが到着している。このレジスタード・プロシージャに対するハンドラが dbreghandle によってインストールされている場合、dbpoll は、呼び出し元に戻る前にこのハンドラを呼び出す。レジスタード・プロシージャに対するハンドラがインストールされておらず、この DBPROCESS に対するデフォルトのハンドラもインストールされていない場合、DB-Library は、ノーティフィケーションを読み込んだときにエラーを発生させる。
DBTIMEOUT	サーバの応答が到着する前に、milliseconds パラメータで指定された時間が経過した。
DBINTERRUPT	サーバの応答は到着しておらず、タイムアウト時間も経過していないが、オペレーティング・システムの割り込みが発生した。

注意 DB-Library/C が認識するサーバの応答の種類の増加に伴い、このリストは今後拡張される可能性があります。*return_reason* に予期しない値が返されてもエラーを起こさず処理できるようにアプリケーション・プログラムをコーディングすることをおすすめします。

戻り値

SUCCEED または FAIL。

dbpoll は、チェックしているサーバ接続のいずれかが切断されている場合に FAIL を返します。dbpoll が FAIL を返すとき、*ready_dbproc* と *return_reason* の内容は定義されていません。

使用法

- dbpoll は、TDS (Tabular Data Stream) バッファに、アプリケーションがまだ読み込んでいないサーバの応答があるかどうかを調べます。
- *dbproc* は、dbpoll がチェックする DBPROCESS 接続を表します。*dbproc* が NULL として渡された場合、dbpoll は、オープンしているすべての接続を調べ、読み込まれていないサーバの応答を検出するとすぐに呼び出し元に戻ります。

- 読み込まれていない応答がある場合、dbpoll は、その応答がどの DBPROCESS 接続に対するもので、その応答が何であるのかを反映するように **ready_dbproc* と *return_reason* を設定します。
- *ready_dbproc* は DBPROCESS 構造体を指すポインタではありません。これは、DBPROCESS 構造体のアドレスを指すポインタです。dbpoll は、サーバの応答が到着している DBPROCESS を指すように **ready_dbproc* を設定します。応答が到着していない場合、dbpoll は **ready_dbproc* を NULL に設定します。
- dbpoll は、次の 2 つの目的で使用されます。
  - アプリケーションによる、サーバからの非ブロッキング読み込み (dbsqlok 呼び出し) の実装を可能にする
  - DBPROCESS に対するレジスタード・プロシージャ・ノーティフィケーションが到着したかどうかを調べる

非ブロッキング読み込みに dbpoll を使用する

- dbpoll を使用すると、dbsqlok での読み込みが可能なバイトがあるかどうかを調べることができます。
- アプリケーションの性質によっては、サーバへのコマンドの送信 (dbsqlsend または dbrpcsend によって行われる) とサーバの応答 (最初に dbsqlok で読み込まれる) の間には、かなりの時間がかかることがあります。

- この間、サーバはコマンドを処理して、結果データを構築しています。アプリケーションは、その時間を他の作業に使用することができます。準備ができたら、アプリケーションは dbpoll を呼び出して、他の作業を行っている間にサーバの応答が到着したかどうかを調べます。この使用法の例については、dbsqlok のリファレンス・ページを参照してください。

  注意 サーバ応答の最初の数バイトしか存在していなくても、dbsqlok によって読み込むデータの準備ができていると dbpoll が報告することがあります。この場合、dbsqlexec と同じように、dbsqlok は残りの応答を待つか、またはタイムアウト時間が経過するまで待ちます。しかし、実際には、ほとんどの場合に応答全体が一度に使用可能になります。

- dbpoll は、dbresults や dbnextrow と一緒には使用しないでください。dbpoll は、これらのルーチンの呼び出しがブロックするかどうかを判断することはできません。dbpoll は、DBPROCESS 接続上で読み込み可能なバイトがあるかどうかを調べることによって機能しますが、この 2 つのルーチンは常にネットワークから読み込まれるわけではないからです。
  - コマンドからの結果がすべて読み込まれると、dbresults は NO_MORE_RESULTS を返します。この場合、読み込むバイトがなくても dbresults はブロックしません。
  - 結果セットのローがすべて読み込まれると、dbnextrow は NO_MORE_ROWS を返します。この場合、読み込むバイトがなくても dbnextrow はブロックしません。
- 非ブロッキング読み込みでは、dbpoll の代わりに DBRBUF と DBIORDESC を使用できます。これらのルーチンは、UNIX 固有のプラットフォームのみで使用可能です。これらのルーチンは移植不可能なので、なるべく使用しないでください。ただし、これらのルーチンを利用すれば、DB-Library/C ソケットの処理を、アプリケーションが使用している他のソケットと統合できます。
  - DBRBUF は、UNIX のみで使用可能なルーチンです。このルーチンは、DB-Library の内部ネットワーク・バッファをチェックして、サーバの応答がすでに読み込まれているかどうかを調べます。dbpoll は、アプリケーションの DBPROCESS が使用している 1 つまたはすべての接続をチェックして、応答が読み込み可能な状態かどうかを調べます。

- DBIORDESC も UNIX のみで使用可能なルーチンで、機能は dbpoll に似ています。DBIORDESC は DBPROCESS によるネットワーク読み込みに使用されるソケット・ハンドルをアプリケーションで使用できるようにします。このソケット・ハンドルは、UNIX の select 関数とともに使用できます。

レジスタード・プロシージャ・ノーティフィケーションに dbpoll を使用する

- アプリケーションでは、1 つまたは複数の DBPROCESS 接続がレジスタード・プロシージャ・ノーティフィケーションを待っていることがあります。DBPROCESS 接続は、サーバからの結果を読み込まないかぎり、レジスタード・プロシージャ・ノーティフィケーションが到着していることを認識しません。接続が結果を読み込んでいない場合は、dbpoll を使用して、レジスタード・プロシージャ・ノーティフィケーションが到着しているかどうかを確認できます。dbpoll はレジスタード・プロシージャ・ノーティフィケーション・ストリームを読み込み、そのレジスタード・プロシージャに対するハンドラを呼び出します。
- dbpoll を使用してレジスタード・プロシージャ・ノーティフィケーションをポーリングする例を次に示します。

```
/*
 ** This code fragment illustrates the use of
 ** dbpoll() to processan event notification.
 **
 ** The code fragment will ask the Server to
 ** notify the Client when the event "shutdown"
 **   occurs. When the event notification is
 ** received from the Server, DB-Library will call
 ** the handler installed for that event. This
 ** event handler routine can then access the
 ** event's parameters, and take any appropriate
 ** action.
 */

DBINT   handlerfunc();
 DBINT   ret;

/* First install the handler for this event */
 dbreghandle(dbproc, "shutdown", handlerfunc);

/*
 ** Now make the asynchronous notification
 ** request.
```

```
*/
ret = dbregwatch(dbproc, "shutdown", DBNULLTERM,
    DBNOWAITONE);
if (ret == FAIL)
{
    fprintf(stderr, "ERROR:dbregwatch() ¥
          failed!!¥n");
}
else if (ret == DBNOPROC)
{
     fprintf(stderr, "ERROR:procedure shutdown ¥
          not defined!¥n");
}

/*
** Since we are making use of the asynchronous
** event notification mechanism, the application
** can continue doing other work. All we have to
** do is call dbpoll() once in a while, to deal
** with the event notification when it arrives.
*/
while (1)
{
     /* Have dbpoll() block for one second */
     dbpoll (NULL, 1000, NULL, &ret);

   /*
     ** If we got the event, then get out of this
     ** loop.
     */
     if (ret == DBNOTIFICATION)
     {
          break;
     }
     /* Deal with our other tasks here */
 }
```

参照 DBIORDESC、DBRBUF、dbresults、dbreghandle、dbsqlok

dbprhead

説明　サーバから返されるローのカラム見出しを出力します。

構文

```
void dbprhead(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　なし。

使用法

- このルーチンは、デフォルトの出力デバイスにデフォルト・フォーマットで、クエリ結果セットに対するカラム見出しを表示します。このフォーマットは dbprrow で使用されるフォーマットと互換性があります。
- アプリケーションは、dbresults から SUCCEED が返された後で、dbprhead を呼び出すことができます。
- 1 行の最大文字数を指定するには、DB-Library の DBPRLINELEN オプションを使用します。
- このルーチンはデバッグに役立ちます。
- ルーチン dbsprhead、dbsprline、および dbspr1row は、dbprhead と dbprrow の代わりに使用できます。これらのルーチンは、フォーマットしたロー結果を、呼び出し元が指定した文字バッファに出力します。

参照　dbbind、dbnextrow、dbprrow、dbresults、dbspr1row、dbsprhead、dbsprline

dbprrow

説明　サーバから返されたすべてのローを出力します。

構文

```
RETCODE dbprrow(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、デフォルトの出力デバイスにデフォルト・フォーマットで、クエリ結果セットに対するローを表示します。このルーチンは、すべてのローを読み込んで出力します。そのため、dbbind や dbnextrow などのルーチンを呼び出す手間を省くことができますが、あらかじめ定義された 1 つのフォーマットでしか出力できません。
- アプリケーションは、dbresults から SUCCEED が返された後で、dbprrow を呼び出すことができます。
- このルーチンを使用するときは、dbnextrow を呼び出してローをループ処理する必要はありません。
- 1 行の最大文字数を指定するには、DB-Library の DBPRLINELEN オプションを使用します。
- dbprrow は、主にデバッグに役立ちます。
- ロー・バッファリングがオンの場合、dbprrow はローを出力するだけでなく、ローをバッファに入れます。バッファが満杯になると、古いローから必要に応じて削除されます。
- ルーチン dbsprhead、dbsprline、および dbspr1row は、dbprhead と dbprrow の代わりに使用できます。これらのルーチンは、フォーマットしたロー結果を、呼び出し元が指定した文字バッファに出力します。

参照

dbbind、dbnextrow、dbprhead、dbresults、dbspr1row、dbsprhead、dbsprline

dbprtype

説明

トークン値を、読み込み可能な文字列に変換します。

構文

```
char    *dbprtype(token)
int     token;
```

パラメータ

token

dbcoltype、dbalttype、dbrettype,、dbaltop によって返されるサーバ・トークン値 (SYBCHAR、SYBFLT8 など) です。

戻り値

トークン値を読み込み可能に変換した、NULL で終了する文字列を指すポインタです。このポインタが指す領域は決して上書きされないので、このルーチンを同一文中で 2 回以上呼び出しても差し支えありません。トークン値を認識できない場合は、このルーチンは空の文字列を指すポインタを返します。

使用法

- dbcoltype、dbalttype、dbrettype、dbaltop などのルーチンは、サーバのデータ型または集合演算子を表すトークン値を返します。dbprtype は、このトークン値を読み込み可能な文字列に変換したものを返します。
- たとえば、dbprtype は、サーバのバイナリ・データ型 (SYBBINARY) を表す dbcoltype トークン値を受け取って、文字列「binary」を返します。
- dbprtype が返すトークン文字列とそれに対応するトークン値のリストを、表 2-23 に示します。

表 2-23：トークン値と対応する文字列

トークン文字列	トークン値	説明
char	SYBCHAR	char データ型
text	SYBTEXT	text データ型
binary	SYBBINARY	binary データ型
image	SYBIMAGE	image データ型
tinyint	SYBINT1	1 バイト integer データ型
smallint	SYBINT2	2 バイト integer データ型
int	SYBINT4	4 バイト integer データ型
float	SYBFLT8	8 バイト float データ型
real	SYBREAL	4 バイト float データ型
numeric	SYBNUMERIC	numeric データ型
decimal	SYBDECIMAL	decimal データ型
bit	SYBBIT	bit データ型
money	SYBMONEY	money データ型
smallmoney	SYBMONEY4	4 バイト money データ型
datetime	SYBDATETIME	datetime データ型
smalldatetime	SYBDATETIME4	4 バイト datetime データ型
boundary	SYBBOUNDARY	boundary 型
sensitivity	SYBSENSITIVITY	sensitivity 型
sum	SYBAOPSUM	合計集合演算子
avg	SYBAOPAVG	平均集合演算子
count	SYBAOPCNT	カウント集合演算子
min	SYBAOPMIN	最小集合演算子
max	SYBAOPMAX	最大集合演算子

参照 dbaltop、dbalttype、dbcoltype、dbrettype、「データ型」(443 ページ)

dbqual

説明 ブラウズ可能なテーブルの現在のローの更新に適した where 句を指すポインタを返します。

構文

```
char *dbqual(dbproc, tabnum, tabname)

DBPROCESS    *dbproc;
int          tabnum;
char         *tabname;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

tabnum

select 文の from 句で指定されている、対象となるテーブルの番号です。テーブル番号は 1 から始まります。*tabnum* が -1 の場合は、テーブルの識別には *tabname* パラメータが使用されます。

tabname

select 文の from 句で指定されている、NULL で終了するテーブル名を指すポインタです。*tabnum* に -1 が渡された場合を除いて、*tabname* は無視されます。

戻り値

指定したテーブルの現在のローに対する、NULL で終了する where 句を指すポインタです。このバッファは動的に割り付けられます。dbfreequal を使用してこのバッファを解放する処理はアプリケーション側で行います。

指定したテーブルがブラウズ可能でない場合、dbqual は、NULL ポインタを返します。テーブルが「ブラウズ可能」となるには、ユニーク・インデックスとタイムスタンプ・カラムが必要です。

すでに実行された select に for browse オプションが含まれていなかった場合にも、dbqual は NULL ポインタを返します。

使用法

- dbqual は、DB-Library ブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- dbqual は、ブラウズ可能なテーブル内の 1 つのローを更新するためにアプリケーションが使用できる where 句を生成します。このローのカラムは、ブラウズ・モードの select クエリ (for browse というキーワードで終了する select) によって前もってアプリケーションに取り出されていなければなりません。

  dbqual によって生成された where 句は、キーワード where で始まり、ローのユニーク・インデックスとタイムスタンプ・カラムへの参照が含まれています。アプリケーションは、update 文または delete 文の最後にこの where 句をそのまま付加します。where 句を検査したり、なんらかの方法で操作したりする必要はありません。

タイムスタンプ・カラムは、特定のローが最後に更新された時間を示します。dbqual が生成した where 句内のタイムスタンプ・カラムが、ブラウズ可能なテーブル内のタイムスタンプ・カラムと異なっている場合は、そのテーブルに対する更新は失敗します。そのような状態では、アプリケーションがブラウズするローを選択した時点とそのローを更新しようとした時点の間に他のユーザがそのローを更新したことを示す、Adaptive Server Enterprise のエラー・メッセージ 532 が生成されます。更新の失敗を処理する論理は、アプリケーション側で用意する必要があります。次のコードは、アプローチの例です。

```
/* This code fragment illustrates a technique for
 ** handling the case where a browse-mode update fails
 ** because the row has already been updated
 ** by another user. In this example, we simply retrieve
 ** the entire row again, allow the user to examine and
 ** modify it, and try the update again.
 **
 ** Note that "q_dbproc" is the DBPROCESS used to query
 ** the database, and "u_dbproc" is the DBPROCESS used
 ** to update the database.
 */

/* First, find out which employee record the user
 ** wants to update.
 */
 employee_id = which_employee();

while (1)
  {
      /* Retrieve that employee record from the database.
       ** We'll assume that "empid" is a unique index,
       ** so this query will return only one row.
       */
      dbfcmd (q_dbproc, "select * from employees where ¥
          empid = %d for browse", employee_id);
       dbsqlexec(q_dbproc);
       dbresults(q_dbproc);
       dbnextrow(q_dbproc);

     /* Now, let the user examine or edit the employee's
       ** data, first placing the data into program
       ** variables.
       */
       extract_employee_data(q_dbproc, employee_struct);
       examine_and_edit(employee_struct, &edit_flag);

     if (edit_flag == FALSE)
```

```
                    {
                        /* The user didn't edit this record,

                        ** so we're done.
                        */
                        break;
                    }
                    else
                    {
                        /* The user edited this record, so we'll use
                        ** the edited data to update the
                        ** corresponding row in the database.
                        */
                        qualptr = dbqual(q_dbproc, -1, "employees");
                        dbcmd(u_dbproc, "update employees");
                        dbfcmd (u_dbproc, " set address = '%s', ¥
                            salary = %d %s",
                            employee_struct->address,
                            employee_struct->salary, qualptr);
                        dbfreequal(qualptr);
                        if ((dbsqlexec(u_dbproc) == FAIL) ||
                            (dbresults(u_dbproc) == FAIL))
                        {
                            /* Our update failed. In a real program,
                            ** it would be necessary to examine the
                            ** messages returned from the Adaptive
Server Enterprise
                            ** to determine why it failed. In this
                            ** example, we'll assume that the update
                          ** failed because someone else has already
                            ** updated this row, thereby changing
                            ** the timestamp.
                            **
                          ** To cope with this situation, we'll just
                          ** repeat the loop, retrieving the changed
                            ** row for our user to examine and edit.
                          ** This will give our user the opportunity
                            ** to decide whether to overwrite the
                            ** change made by the other user.
                             */
                             continue;
                        }
```

```
                else
                {
                    /* The update succeeded, so we're done.*/
                     break;
                }
            }
        }
```

- dbqual は、ブラウズ可能なテーブルに対する where 句の構成だけを行います。テーブルがブラウズ可能かどうかを調べるには、dbtabbrowse を使用します。
- dbqual は、通常、dbnextrow の後で呼び出します。
- dbqual を使用してブラウズ・モードの更新を行う例については、DB-Library とともに提供されるオンラインのサンプル・プログラムを参照してください。

参照

dbcolbrowse、dbcolsource、dbfreequal、dbtabbrowse、dbtabcount、dbtabname、dbtabsource、dbtsnewlen、dbtsnewval、dbtsput

DBRBUF

説明

(UNIX のみ) DB-Library ネットワーク・バッファに、まだ読み込まれていないバイトがあるかどうかを調べる。

構文

```
DBBOOL DBRBUF(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

「TRUE」(バッファ内にバイトがある場合) または「FALSE」(バッファ内にバイトがまったくない場合)。

実際には、読み込みバッファ内にバイトがあるときと、処理できる結果がそれ以上ないときのどちらも、DBRBUF は「TRUE」を返します。

これは、その時点でアプリケーションによる読み込みが可能であり、アプリケーションがハングしないことを保証するということの通知が DBRBUF の目的であるからです。それ以上結果がないときに DBRBUF が「TRUE」を返さないとすれば、DBRBUF が「FALSE」を返す間ループするアプリケーションは、すべての結果がすでに処理されている場合に永久にループを繰り返すことになります。

使用法

- このルーチンを使用すると、DB-Library ネットワーク・バッファ内にまだ読み込まれていないバイトがあるかどうかをアプリケーションで調べることができます。
- DBRBUF は、通常、dbsqlok と DBIORDESC とともに使用します。
- サーバの応答が DBPROCESS に到着しているかどうかチェックする DB-Library/C ルーチンである dbpoll を、DBRBUF の代わりに使用できます。UNIX 専用のルーチンである DBRBUF と DBIORDESC は移植できないため、これらはなるべく使用しないでください。ただし、これらのルーチンを利用すれば、DB-Library/C ソケットの処理を、アプリケーションが使用している他のソケットと統合できます。
- これらのルーチンは、アプリケーションで複数の入力データ・ストリームを管理するときに使用します。このストリームを効率的に管理するために、dbsqlok を使用するアプリケーションでは、dbresults を呼び出す前に、ネットワーク・バッファまたはネットワーク自体にバイトが残っているかどうかをチェックするようにしてください。
- ネットワーク・バッファ内にバイトが残っているかどうかをアプリケーションでテストするには、DBRBUF を呼び出します。ネットワーク自体にバイトが残っているかどうかをテストするには、UNIX の select と DBIORDESC を呼び出すか、または dbpoll を呼び出します。

参照　DBIORDESC、dbpoll、dbsqlok、dbresults

dbreadpage

説明　サーバから 1 ページ分のバイナリ・データを読み込みます。

構文

DBINT dbreadpage(dbproc, dbname, pageno, buf)

```
DBPROCESS    *dbproc;
char         *dbname;
DBINT        pageno;
BYTE         buf[];
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

dbname
対象となるデータベースの名前です。

pageno
読み込まれるデータベース・ページのページ番号です。

buf
受け取ったページ・データを保持するバッファを指すポインタです。現時点では、Adaptive Server Enterprise のページの長さは 2048 バイトです。

戻り値

サーバから読み込まれたバイト数です。オペレーションが失敗した場合、dbreadpage は -1 を返します。

使用法

- dbreadpage は、サーバから 1 ページ分のバイナリ・データを読み込みます。このルーチンは、主に損傷したデータベースのページを検査および修復するのに使用します。dbreadpage を呼び出した後は、DBPROCESS にサーバからのエラー・メッセージまたは情報メッセージが格納されていることがあります。これらのメッセージには、ユーザ提供のメッセージ・ハンドラによってアクセスできます。
- dbreadpage により、DBPROCESS コマンド・バッファの内容が変更されます。

警告！このルーチンがどのような処理を行うかを十分理解したうえで、このルーチンを使用してください。

参照

dbmsghandle、dbwritepage

dbreadtext

説明

サーバから text 値または image 値の一部を読み込みます。

構文

STATUS dbreadtext(dbproc, buf, bufsize)

```
DBPROCESS    *dbproc;
void         *buf;
DBINT        bufsize;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

buf
　呼び出し側が割り付けたバッファを指すポインタです。このバッファに、text または image のデータのまとまりが格納されます。

bufsize
　呼び出し側のバッファのバイト単位でのサイズです。

戻り値

次の表は、dbreadtext の戻り値のリストです。

dbreadtext の戻り値	意味
>0	呼び出し側のバッファに格納されたバイト数。
0	ローの終わり。
-1	ネットワーク・エラーやメモリ不足エラーなどのエラーが発生した。
NO_MORE_ROWS	すべてのローが読み込まれた。

使用法

- dbreadtext は、サーバからの大きな SYBTEXT 値または SYBIMAGE 値を、いくつかの小さなまとまりの形で読み込みます。これは、非常に大きなデータ・バッファを割り付けることができないオペレーティング・システムの場合に特に有効です。
- 同じ SYBTEXT または SYBIMAGE 値を構成するまとまりを連続して読み込むには、dbreadtext が 0 を返す (ローの終わり) まで dbreadtext を呼び出してください。
- SYBTEXT 値と SYBIMAGE 値を読み込むには、dbnextrow の代わりに dbreadtext を使用してください。
- Transact-SQL クエリが 1 カラムだけを返すものであり、そのカラムの内容が text または image のデータである場合に、dbreadtext を使用してその Transact-SQL クエリの結果を処理できます。Transact-SQL の readtext コマンドは、このタイプの結果を返します。

- DB-Library/C の DBTEXTSIZE オプションは、サーバのグローバル変数 @@*textsize* の値に反映されます。この変数は、サーバが返す text 値または image 値のサイズを制限するものです。@@*textsize* のデフォルト値は 32,768 バイトです。アプリケーションで 32,768 バイトを超える text 値または image 値を取得する場合は、dbsetopt を呼び出して @@*textsize* を大きくする必要があります。

  DB-Library/C の DBTEXTLIMIT オプションは、DB-Library/C が読み込む text 値または image 値のサイズを制限します。DB-Library/C は、その制限を超えるテキストを破棄します。

- 次のコードは、dbreadtext の例です。

```
DBPROCESS     *dbproc;
long          bytes;
RETCODE       ret;
char          buf[BUFSIZE + 1];
/*
 ** Install message and error handlers...
 ** Log in to server...
 ** Send a "use database" command...
 */
/* Select a text column:*/
 dbfcmd(dbproc, "select textcolumn from bigtable");
 dbsqlexec(dbproc);

/* Process the results:*/
 while( (ret = dbresults(dbproc)) !=
      NO_MORE_RESULTS)
 {
      if (ret == FAIL)
      {
           /* dbresults() failed */
      }
     while( (bytes =
dbreadtext(dbproc,
           (void *)buf, BUFSIZE)) != NO_MORE_ROWS )
      {
           if( bytes == -1 )
           {
                /* dbreadtext() failed */
           }
           else if( bytes == 0 )
           {
                /* We've reached the end of a row*/
                printf("End of Row!¥n¥n");
           }
```

```
                    else
                    {
                        /*
                        ** 'bytes' bytes have been placed
                        ** into our buffer.
                        ** Print them:
                        */
                        buf[bytes] = '¥0';
                        printf("%s¥n", buf);
                    }
                }
            }
```

参照　dbmoretext、dbnextrow、dbwritetext

dbrecftos

説明　アプリケーションからサーバへ送信されたすべての SQL コマンドを記録します。

構文　void dbrecftos(filename)

```
char        *filename;
```

パラメータ　filename

SQL セッション・ファイルの名前の基となる、null で終了する文字列を指すポインタです。

戻り値　なし。

使用法

- dbrecftos は、フロントエンド・アプリケーション・プログラムからサーバに送信されたすべての SQL コマンドを、人間が判読可能なファイルに記録します。この SQL セッション情報は、デバッグに役立ちます。
- DB-Library は、dbrecftos が呼び出された後の dbopen の呼び出しごとに 1 つの SQL セッション・ファイルを作成します。ファイル名は、*filename.n* となります。*filename* は dbrecftos の呼び出しで指定される名前で、*n* は 0 から始まる整数です。

  たとえば、*filename* が「foo」ならば、最初に作成されるファイルは *foo.0*、次は *foo.1* というように命名されます。

参照　dbopen

dbrecvpassthru

説明

サーバから TDS パケットを受信します。

構文

RETCODE dbrecvpassthru(dbproc, recv_bufp)

```
DBPROCESS     *dbproc;
DBVOIDPTR     *recv_bufp;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

recv_bufp

変数へのポインタです。この変数には、この DBPROCESS 接続によって受信された最後の TDS パケットが格納されているバッファのアドレスが dbrecvpassthru によって格納されます。アプリケーションがこのバッファを割り付ける必要はありません。

戻り値

DB_PASSTHRU_MORE、DB_PASSTHRU_EOM、または FAIL。

使用法

- dbrecvpassthru は、サーバから TDS (Tabular Data Stream) パケットを受信します。
- TDS は、クライアントとサーバの間で、要求と要求結果を転送するために使用されるアプリケーション・プロトコルです。通常は、DB-Library/C がデータ・ストリームを管理しているため、DB-Library/C アプリケーションが直接 TDS を取り扱う必要はありません。
- dbrecvpassthru および dbsendpassthru は、ゲートウェイ・アプリケーションに役立ちます。2 つのサーバを仲介する役割を持つアプリケーションでこれらのルーチンを使用すると、情報を解釈して再エンコードすることなく 1 つのサーバから他のサーバへ TDS ストリームを転送できます。
- dbrecvpassthru は、*dbproc* で指定されたサーバ接続から 1 パケット分のバイトを読み込み、そのバイトが格納されているバッファを指すように **recv_bufp* を設定します。
- 1 パケットのデフォルト・サイズは、512 バイトです。アプリケーションでこのパケット・サイズを変更するには、DBSETLPACKET を使用します。詳細については、dbgetpacket および DBSETLPACKET のリファレンス・ページを参照してください。

- サーバによって TDS パケットに EOM (End Of Message) マークが付けられていた場合は、dbrecvpassthru は DB_PASSTHRU_EOM を返します。TDS パケットがストリームの最後でない場合、dbrecvpassthru は DB_PASSTHRU_MORE を返します。
- dbrecvpassthru オペレーションで使用される DBPROCESS 接続は、DB_PASSTHRU_EOM が受信されるまでは、他の DB-Library/C 関数で使用することはできません。
- 次のコードは、dbrecvpassthru の例です。

```
/*
** The following code fragment illustrates the
** use of dbrecvpassthru() in an Open Server
** gateway application. It will continually get
** packets from a remote server, and pass them
** through to the client.
**
** The routine srv_sendpassthru() is the Open
* Server counterpart required to complete
 ** this passthru operation.
*/
DBPROCESS     *dbproc;
SRV_PROC      *srvproc;
int           ret;
BYTE          *packet;

while (1)
 {
      /* Get a TDS packet from the remote server */
      ret = dbrecvpassthru(dbproc, &packet);

    if (ret == FAIL)
      {
           fprintf(stderr, "ERROR - dbrecvpassthru¥
                failed in handle_results.¥n");
           exit();
      }
     /* Now send the packet to the client */
      if( srv_sendpassthru(srvproc, packet,
           (int *)NULL) == FAIL )
      {
      fprintf(stderr, "ERROR - srv_sendpassthru ¥
                failed in handle_results.¥n");
           exit();
      }
     /*
      ** We've sent the packet, so let's see if
```

```
** there's any more.
*/
if( ret == DB_PASSTHRU_MORE )
     continue;
else
     break;
       }
```

参照 dbsendpassthru

dbregdrop

説明 レジスタード・プロシージャを削除するため。

構文 RETCODE dbregdrop(dbproc, procedure_name, namelen)

```
DBPROCESS    *dbproc;
DBCHAR       *procedure_name;
DBSMALLINT   namelen;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

procedure_name
DBPROCESS 接続が削除しようとするレジスタード・プロシージャの名前を指すポインタです。

namelen
procedure_name のバイト単位の長さです。*procedure_name* が NULL で終了する場合は、*namelen* を DBNULLTERM として渡してください。

戻り値 SUCCEED、DBNOPROC、または FAIL。

使用法

- dbregdrop は、Open Server からレジスタード・プロシージャを削除します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、dbregdrop によってノーティフィケーション・プロシージャも削除できます。

- DBPROCESS 接続は、他の DBPROCESS 接続によって作成されるプロシージャや、アプリケーションによって作成されるプロシージャも含めて Open Server 内に定義されるすべてのレジスタード・プロシージャを削除できます。プロシージャを保護するためのメカニズムがサーバ・アプリケーションに組み込まれていなければなりません。
- *procedure_name* によって参照されるプロシージャが Open Server 内に定義されていない場合、dbregdrop は DBNOPROC を返します。アプリケーションで dbreglist を使用すると、Open Server 内に現在定義されているレジスタード・プロシージャのリストを得ることができます。
- 次のコードは、dbregdrop の使用例です。

```
/*
** The following code fragment illustrates
 ** dropping a registered procedure.
 */
DBPROCESS     *dbproc;
RETCODE       ret;
char          *procname;

procname = "some_event";
ret = dbregdrop(dbproc, procname, DBNULLTERM);
 if (ret == FAIL)
 {
      fprintf(stderr, "ERROR:dbregdrop() ¥
           failed!!¥n");
 }
 else if (ret == DBNOPROC)
 {
      fprintf(stderr, "ERROR:procedure %s was not¥
           registered!¥n", procname);
 }
```

参照　dbnpcreate、dbreglist

dbregexec

説明　レジスタード・プロシージャを実行します。

構文

RETCODE dbregexec(dbproc, options)

```
DBPROCESS     *dbproc;
DBUSMALLINT   options;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

オプション

2 バイトのビットマスクです。DBNOTIFYALL または DBNOTIFYNEXT のいずれかです。

options が DBNOTIFYALL の場合、Open Server は、このレジスタード・プロシージャの実行を監視しているすべての DBPROCESSes に通知します。

options が DBNOTIFYNEXT の場合、Open Server は最も長時間監視している DBPROCESS だけに通知します。

戻り値

SUCCEED または FAIL。

使用法

- dbregexec は、レジスタード・プロシージャを実行するプロセスを完了します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、ノーティフィケーション・プロシージャも dbregexec を使用して実行できます。
- プロシージャ名とパラメータは、dbreginit と dbregparam を使用してあらかじめ定義しておく必要があります。
- DB-Library/C アプリケーションでレジスタード・プロシージャを実行するには、次のことを行う必要があります。
  - dbreginit を使用して呼び出しを開始する。
  - プロシージャのパラメータがある場合は、dbregparam を使用してそのパラメータを記述する。
  - dbregexec を使用してプロシージャを実行する。
- Open Server 内に定義されていないレジスタード・プロシージャをアプリケーションで実行することはできません。dbreglist を実行すると、Open Server 内に現在定義されているレジスタード・プロシージャのリストが返されます。

- アプリケーションはレジスタード・プロシージャが実行されたときに通知 (ノーティフィケーション) を受け取ることを要求できるので、レジスタード・プロシージャはアプリケーション間の通信や同期に便利です。
- レジスタード・プロシージャは、Open Server 内でだけ作成されます。現時点では、Adaptive Server Enterprise はノーティフィケーション・プロシージャをサポートしていません。アプリケーションは、dbnpcreate、dbregparam、および dbnpcreate を使用して、レジスタード・プロシージャを作成できます。
- DB-Library/C アプリケーションで、レジスタード・プロシージャの実行のノーティフィケーションを要求するには、dbregwatch を使用します。アプリケーションは、同期と非同期のどちらのノーティフィケーションを受け取るかを要求できます。
- レジスタード・プロシージャの実行例を次に示します。

```
DBPROCESS    *dbproc;
DBINT        newprice = 55;
DBINT        status;
/*
 ** Initiate execution of the registered procedure
 ** "price_change"
 */
 dbreginit (dbproc, "price_change", DBNULLTERM);

/*
 ** The registered procedure has two parameters:
 **      name        varchar(255)
 **      newprice    int
 ** So pass these parameters to the registered
 ** procedure.
 */
dbregparam (dbproc, "name", SYBVARCHAR, NULL,
      "sybase");
 dbregparam (dbproc, "newprice", SYBINT4, 4,
      &newprice);
/*  Execute the registered procedure:*/
 status = dbregexec (dbproc, DBNOTIFYALL);
 if (status == FAIL)
 {
      fprintf(stderr, "ERROR:Failed to execute ¥
           price_change!¥n");
 }

else if (status == DBNOPROC)
 {
```

```
            fprintf(stderr, "ERROR:Price_change does ¥
                not exist!¥n");
        }
        else
        {
            fprintf(stdout, "Success in executing ¥
                price_change!¥n");
        }
```

参照 dbreginit、dbregparam、dbregwatch、dbregnowatch

dbreghandle

説明 レジスタード・プロシージャ・ノーティフィケーションのハンドラ・ルーチンをインストールします。

構文

```
RETCODE dbreghandle(dbproc, procedure_name, namelen,
handler)

DBPROCESS    *dbproc;
DBCHAR       *procedure_name;
DBSMALLINT   namelen;
INTFUNCPTR   handler;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

procedure_name

ハンドラをインストールするレジスタード・プロシージャの名前を指すポインタです。

procedure_name に NULL を渡すと、ハンドラはデフォルト・ハンドラとしてインストールされます。デフォルト・ハンドラは、この DBPROCESS 接続によって読み込まれるレジスタード・プロシージャ・ノーティフィケーションのうち、他のハンドラがインストールされていないものすべてに対して呼び出されます。

namelen

procedure_name のバイト単位の長さです。*procedure_name* が NULL で終了する場合は、*namelen* を DBNULLTERM として渡してください。

ハンドラ
レジスタード・プロシージャ・ノーティフィケーションが読み込まれたときに DB-Library/C によって呼び出される関数を指すポインタです。

handler が NULL として渡されると、このレジスタード・プロシージャに対してインストールされているすべてのハンドラのインストールが解除されます。

戻り値

SUCCEED または FAIL。

使用法

- dbreghandle は、レジスタード・プロシージャが実行されたことを示す非同期のノーティフィケーション (通知) を DBPROCESS 接続が読み込んだときに DB-Library/C によって呼び出される、ユーザ提供のハンドラ・ルーチンをインストールします。

  ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、ノーティフィケーション・プロシージャに対するハンドラも dbreghandle によってインストールできます。

- アプリケーションが非同期ノーティフィケーションを受け取るのは、事前に実行した dbregwatch の *options* に DBNOWAITONE または DBNOWAITALL を指定した場合だけです。アプリケーションは、この呼び出しを行うことで、レジスタード・プロシージャの実行を監視し、ノーティフィケーションを非同期に受信し、特定の DBPROCESS 接続を通してノーティフィケーションを読み込むことを Open Server に通知します。

- ノーティフィケーションに対するハンドラがインストールされていない場合は、DBPROCESS 接続がノーティフィケーションを読み込むと、DB-Library/C のエラーが発生します。

- Either *procedure_name* または *handler* のどちらにも NULL を指定できます。

  - *procedure_name* と *handler* の両方が指定された場合は、dbreghandle は *handler* で指定されたハンドラを、*procedure_name* で指定されたレジスタード・プロシージャに対してインストールします。

  - *procedure_name* が NULL で *handler* も NULL の場合、dbreghandle はこの DBPROCESS 接続に対するすべてのハンドラのインストールを解除します。

    - *procedure_name* が NULL で *handler* の値が指定されている場合、dbreghandle は、*handler* で指定されたハンドラをこの DBPROCESS 接続の「デフォルト」のハンドラとしてインストールします。このデフォルト・ハンドラは、DBPROCESS 接続が読み込んだレジスタード・プロシージャ・ノーティフィケーションに対して他のハンドラがインストールされていないときに呼び出されます。
    - *procedure_name* の値が指定されていて、*handler* が NULL の場合、dbreghandle は、このレジスタード・プロシージャに対してインストールされているすべてのハンドラのインストールを解除します。この DBPROCESS 接続に対してデフォルト・ハンドラがインストールされている場合、デフォルト・ハンドラは有効のままであり、*procedure_name* のノーティフィケーションが読み込まれると呼び出されます。

- 複数の DBPROCESS 接続で同一のハンドラを使用できますが、DBPROCESS 接続のそれぞれに対して、別の dbreghandle 呼び出しを使用してハンドラをインストールする必要があります。複数の DBPROCESS がそれぞれノーティフィケーションを読み込んだときに同じノーティフィケーション・ハンドラが呼び出される可能性があるため、ハンドラはすべてリエントラントとなるように作成してください。
- 1 つの DBPROCESS 接続が、複数のレジスタード・プロシージャの実行を監視している場合があります。この接続は、読み込まれるさまざまなノーティフィケーションを処理するために、複数のハンドラをインストールしていることがあります。それぞれのハンドラは、別々の dbreghandle の呼び出しによってインストールされなければなりません。
- レジスタード・プロシージャ・ノーティフィケーションが到着したとき、DBPROCESS 接続の状態は、アイドル状態、コマンドの送信中、結果の読み込み中、または保留中の結果を持つアイドル状態のいずれかです。
    - DBPROCESS 接続がアイドル状態の場合は、接続がノーティフィケーションを読み込めるようにするために、アプリケーションで dbpoll を呼び出す必要があります。ノーティフィケーションに対するハンドラがインストールされている場合、そのハンドラは dbpoll が呼び出し元に戻る前に呼び出されます。

- DBPROCESS 接続がコマンド送信中の場合は、ノーティフィケーションが読み込まれ、dbsqlexec または dbsqlok の実行中にノーティフィケーション・ハンドラが呼び出されます。ノーティフィケーション・ハンドラが呼び出し元に戻った後は、制御のフローは通常どおりに続行します。
- DBPROCESS 接続が結果の読み込み中の場合は、ノーティフィケーションが読み込まれ、dbresults または dbnextrow の実行中にノーティフィケーション・ハンドラが呼び出されます。ノーティフィケーション・ハンドラが呼び出し元に戻った後は、制御のフローは通常どおりに続行します。
- DBPROCESS 接続が保留中の結果を持つアイドル状態の場合は、ストリーム内の、ノーティフィケーションの直前までのすべての結果を接続が読み込んで処理するまで、ノーティフィケーションは読み込まれません。

- ノーティフィケーションが読み込まれるときの DBPROCESS 接続の状態は一様ではないので、ノーティフィケーション・ハンドラが実行できるアクションは制限されます。ノーティフィケーション・ハンドラは、既存の DBPROCESS を使用してサーバにクエリを送信したり、クエリの結果を処理したり、dbcancel や dbcanquery を呼び出したりすることはできません。ただし、ノーティフィケーション・ハンドラが新しい DBPROCESS をオープンすることは可能で、これを使用してクエリを送信したり、ハンドラ内の結果を処理したりできます。
- ノーティフィケーション・ハンドラは、実行時にレジスタード・プロシージャに渡された引数を読み込むことができます。このことを行うには、DB-Library/C ルーチンの dbnumrets、dbrettype、dbretlen、dbretname、dbretdata を使用します。
- すべてのノーティフィケーション・ハンドラは、DB-Library/C によって呼び出されるときに次のパラメータを受け取ります。
  - *dbproc* ― ノーティフィケーションを監視している DBPROCESS 接続を指すポインタです。
  - *procedure_name* ― 実行されたレジスタード・プロシージャの名前を指すポインタです。
  - *reserved1* ― 今後のために予約されている DBUSMALLINT パラメータです。
  - *reserved2* ― 今後のために予約されている DBUSMALLINT パラメータです。

- ノーティフィケーション・ハンドラは、正常に終了したことを示す INT_CONTINUE か、またはアプリケーションのアボートを DB-Library/C に指示する INT_EXIT を返し、オペレーティング・システムに制御を戻します。
- Windows プラットフォームのノーティフィケーション・ハンドラは、次の例に示すように、CS_PUBLIC で宣言してください。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。
- ノーティフィケーション・ハンドラの例を次に示します。

```
DBINT CS_PUBLIC my_procedure_handler(dbproc,
            procedure_name, reserved1, reserved2)
/* The client connection */
 DBPROCESS       *dbproc;
 /* A null-terminated string */
 DBCHAR          *procedure_name;
 /* Reserved for future use */
 DBUSMALLINT     reserved1;
 /* Reserved for future use */
 DBUSMALLINT     reserved2;

{
      int     i, type;
      DBINT   len;
      char    *name;
      BYTE    *data;
      int     params;

     /*
      ** Find out how many parameters this
      ** procedure received.
      */
      params = dbnumrets(dbproc);

     i = 0;    /* Initialize counter */

    /* Now process each parameter in turn */
      while(i++ < params)
      {
           /* Get the parameter's datatype */
           type = dbrettype(dbproc, i);

          /* Get the parameter's length */
           len = dbretlen(dbproc, i);

         /* Get the parameter's name */
           name = dbretname(dbproc, i);
```

```
                    /* Get a pointer to the parameter */
                      data = dbretdata(dbproc, i);

                     /* Process the parameter here */
                }
                return(INT_CONTINUE);
        }
```

参照　dbregwatch、dbregnowatch、dbregparam、dbregexec

dbreginit

説明　レジスタード・プロシージャの実行を開始します。

構文　RETCODE dbreginit(dbproc, procedure_name, namelen)

```
DBPROCESS    *dbproc;
DBCHAR       *procedure_name;
DBSMALLINT   namelen;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

procedure_name
　実行するレジスタード・プロシージャの名前を指すポインタです。

namelen
　procedure_name のバイト単位の長さです。*procedure_name* が NULL で終了する場合は、*namelen* を DBNULLTERM として渡してください。

戻り値　SUCCEED または FAIL。

使用法

- dbreginit は、レジスタード・プロシージャの実行を開始します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、dbreginit を使用してノーティフィケーション・プロシージャの実行も開始できます。
- DB-Library/C アプリケーションでレジスタード・プロシージャを実行するには、次のことを行う必要があります。
  - dbreginit を使用して呼び出しを開始する

- プロシージャのパラメータがある場合は、dbregparam を使用してそのパラメータを渡す
- dbregexec を使用してプロシージャを実行する

- レジスタード・プロシージャの実行例を次に示します。

```
DBPROCESS     *dbproc;
DBINT         newprice = 55;
DBINT         status;

/*
 ** Initiate execution of the registered procedure
 ** "price_change".
 */
 dbreginit (dbproc, "price_change", DBNULLTERM);

/*
 ** The registered procedure has two parameters:
 **      name        varchar(255)
 **      newprice    int
 ** So pass these parameters to the registered
 ** procedure.
 */

dbregparam (dbproc, "name", SYBVARCHAR, NULL,
     "sybase");
 dbregparam (dbproc, "newprice", SYBINT4, 4, 4,
      &newprice);

/*  Execute the registered procedure:*/
 status = dbregexec (dbproc, DBNOTIFYALL);
 if (status == FAIL)
 {
      fprintf(stderr, "ERROR:Failed to execute ¥
           price_change!¥n");
 }
 else if (status == DBNOPROC)
 {
      fprintf(stderr, "ERROR:Price_change does ¥
           not exist!¥n");
 }
 else
 {
      fprintf(stdout, "Success in executing ¥
           price_change!¥n");
 }
```

参照 dbregparam、dbregexec、dbregwatch、dbreglist、dbregwatchlist

dbreglist

説明

Open Server 内に現在定義されているレジスタード・プロシージャのリストを返します。

構文

RETCODE dbreglist(dbproc)

DBPROCESS　*dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

使用法

- dbreglist は、Open Server 内に現在定義されているレジスタード・プロシージャのリストを返します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、ノーティフィケーション・プロシージャもレジスタード・プロシージャのリストに含まれます。
- レジスタード・プロシージャのリストはローとして返されます。アプリケーションは、dbreglist を呼び出した後にこのローを明示的に処理しなければなりません。それぞれのローは、Open Server で定義されている単一のレジスタード・プロシージャの名前を表しています。ローには、SYBVARCHAR 型のカラムが 1 つだけ含まれています。
- dbreglist のアプリケーションでの使用例を次に示します。

```
DBPROCESS     *dbproc;
DBCHAR        *procedurename;
DBINT         ret;

/* request the list of procedures */
 if( (ret = dbreglist(dbproc)) == FAIL)
 {
      /* Handle failure here */
 }
 dbresults(dbproc);
 while (dbnextrow(dbproc) != NO_MORE_ROWS)
 {
      procedurename = (DBCHAR *)dbdata(dbproc, 1);
      procedurename[dbdatlen(dbproc, 1)] = '¥0';

     fprintf(stdout, "The procedure '%s' is ¥
           defined.¥n", procedurename);
```

```
        }
        /* All done */
```

参照 dbregwatchlist、dbregwatch

dbregnowatch

説明 レジスタード・プロシージャが実行されたときにノーティフィケーションを受け取ることの要求をキャンセルします。

構文

```
RETCODE dbregnowatch(dbproc, procedure_name,
namelen)

DBPROCESS    *dbproc;
DBCHAR       *procedure_name;
DBSMALLINT   namelen;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

procedure_name
今後この DBPROCESS 接続での監視対象としないレジスタード・プロシージャの名前を指すポインタです。

namelen
procedure_name のバイト単位の長さです。*procedure_name* が NULL で終了する場合は、*namelen* を DBNULLTERM として渡してください。

戻り値 SUCCEED、DBNOPROC、または FAIL。

使用法

- dbregnowatch は、レジスタード・プロシージャが実行されたときに DBPROCESS 接続がノーティフィケーションを受け取ることの要求をキャンセルします。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、dbregnowatch は、ノーティフィケーション・プロシージャ実行時に DBPROCESS 接続がノーティフィケーションを受け取ることの要求もキャンセルします。
- dbregnowatch の呼び出しが意味をもつのは、DBPROCESS 接続が dbregwatch によって事前に非同期ノーティフィケーションを要求している場合だけです。

- *procedure_name* によって参照されるプロシージャが Open Server 内に定義されていない場合、dbregnowatch は DBNOPROC を返します。Open Server 内に現在登録されているプロシージャのリストを取得するには、dbreglist を使用します。
- 監視しているレジスタード・プロシージャのリストを取得するには、dbregwatchlist を使用します。
- ノーティフィケーション要求を取り消す例を次に示します。

```
DBPROCESS     *dbproc;
DBINT         ret;

/*
 ** Inform the server that we no longer wish to
 ** be notified when "price_change" executes:
 */
 ret = dbregnowatch (dbproc, "price_change",
      DBNULLTERM);
 if (ret == DBNOPROC)
 {
      /* The registered procedure must not exist */
      fprintf(stderr, "ERROR:price_change ¥
           doesn't exist!¥n");
 }
```

参照　dbregwatch、dbregwatchlist、dbreghandle、dbregexec

dbregparam

説明　レジスタード・プロシージャのパラメータを定義または記述します。

構文

```
RETCODE dbregparam(dbproc,param_name, type, datalen,
          data)

DBPROCESS     *dbproc;
char          *param_name;
int           type;
DBINT         datalen;
BYTE          *data;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

param_name
パラメータの名前を指すポインタです。

レジスタード・プロシージャを作成するときは、*param_name* は必須です。

レジスタード・プロシージャを実行するときは、*param_name* に NULL の指定もできます。この場合、レジスタード・プロシージャは、初めに定義された順序でパラメータを受け取ることになります。

type
パラメータのデータ型を示す記号値です。有効なデータ型は、SYBINT1、SYBINT2、SYBINT4、SYBREAL、SYBFLT8、SYBCHAR、SYBBINARY、SYBVARCHAR、SYBDATETIME4、SYBDATETIME、SYBMONEY4、SYBMONEY です。

SYBTEXT と SYBIMAGE は、パラメータのデータ型としては無効であることに注意してください。

datalen
パラメータの長さです。

レジスタード・プロシージャを作成するときは、次のことに注意してください。

- *datalen* を使用して、このパラメータにデフォルト値が指定されていないことを示すことができます。デフォルト値がないことを示す場合は、*datalen* を DBNODEFAULT として渡してください。
- *datalen* を使用して、パラメータのデフォルト値が NULL であることを示すことができます。これは、デフォルト値を持たないということではありません。デフォルト値が NULL であることを示す場合は、*datalen* を 0 として渡してください。

レジスタード・プロシージャを実行するときは、次のことに注意してください。

- *datalen* に 0 を指定できます。このようにすると、*data* は無視され、このパラメータの値として NULL がレジスタード・プロシージャに渡されます。

data
パラメータを指すポインタです。

レジスタード・プロシージャを作成するときは、*data* を使用してパラメータのデフォルト値を指定できます。*data* には、デフォルト値を指すポインタを渡してください。デフォルト値が必要ない場合は、*datalen* を DBNODEFAULT として渡してください。

レジスタード・プロシージャを実行するときは、*data* を NULL として渡すことができます。

戻り値

SUCCEED または FAIL。

使用法

- dbregparam は、レジスタード・プロシージャのパラメータを定義します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、dbregparam は、ノーティフィケーション・プロシージャのパラメータも定義します。
- dbregparam は、レジスタード・プロシージャを作成するときにレジスタード・プロシージャ・パラメータを定義するために呼び出します。また、レジスタード・プロシージャを実行するときにパラメータを記述するために呼び出します。

  注意　DB-Library/C アプリケーションで作成できるレジスタード・プロシージャは、ノーティフィケーション・プロシージャという特殊なタイプのものだけです。ノーティフィケーション・プロシージャは、通常の Open Server レジスタード・プロシージャとは異なり、実行可能な文は含まれません。詳細については、dbnpdefine および dbnpcreate のリファレンス・ページを参照してください。

- アプリケーションで dbregparam を呼び出す前に、ノーティフィケーション・プロシージャの作成処理を開始する dbnpdefine、またはレジスタード・プロシージャの実行処理を開始する dbreginit を呼び出す必要があります。
- レジスタード・プロシージャを作成するときは、次のことに注意してください。
  - デフォルト値が指定されていないことを示すには、*datalen* を DBNODEFAULT として渡してください。この場合、*data* は無視されます。
  - デフォルト値として NULL を指定するには、*datalen* を 0 として渡してください。この場合、*data* は無視されます。

- NULL 以外のデフォルト値を指定するには、*datalen* に値の長さ (固定長型の場合は -1) を渡し、*data* にはその値を指すポインタを渡します。

- レジスタード・プロシージャを実行するときは、次のことに注意してください。
  - パラメータの値として NULL を渡すには、*datalen* を 0 として渡してください。この場合、*data* は無視されます。
  - このパラメータの値を渡すには、*datalen* に値の長さ (固定長型の場合は -1) を渡し、*data* にはその値を指すポインタを渡します。
- DB-Library/C アプリケーションでノーティフィケーション・プロシージャを作成するには、次のことを行う必要があります。
  - dbnpdefine を使用してプロシージャを定義する
  - プロシージャのパラメータがある場合は、dbregparam を使用してそのパラメータを記述する
  - dbnpcreate を使用してプロシージャを作成する
- 次に、ノーティフィケーション・プロシージャを作成する例を示します。

```
DBPROCESS    *dbproc;
DBINT        status;
/*
 ** Let's create a notification procedure called
 ** "message" which has two parameters:
 **     msg     varchar(255)
 **      userid  int
 */
/*

 ** Define the name of the notification procedure
 ** "message"
 */
 dbnpdefine (dbproc, "message", DBNULLTERM);
/* The notification procedure has two parameters:
 **     msg     varchar(255)
 **      userid  int
 ** So, define these parameters. Note that both
 ** of these parameters are defined with a default
 ** value of NULL. Passing datalen as 0
 ** accomplishes this.
 */
```

```
dbregparam (dbproc, "msg", SYBVARCHAR, 0, NULL);
dbregparam (dbproc, "userid", SYBINT4, 0, NULL);

/*  Create the notification procedure:*/
 status = dbnpcreate (dbproc);
 if (status == FAIL)
 {
      fprintf(stderr, "ERROR:Failed to create ¥
           message!¥n");
 }
 else
 {
      fprintf(stdout, "Success in creating ¥
           message!¥n");
 }
```

- DB-Library/C アプリケーションでレジスタード・プロシージャを実行するには、次のことを行う必要があります。
  - dbreginit を使用して呼び出しを開始する
  - プロシージャのパラメータがある場合は、dbregparam を使用してそのパラメータを渡す
  - dbregexec を使用してプロシージャを実行する
- レジスタード・プロシージャの実行例を次に示します。

```
DBPROCESS    *dbproc;
DBINT        newprice = 55;
DBINT        status;

/*
 ** Initiate execution of the registered procedure
 ** "price_change".
 */
 dbreginit (dbproc, "price_change", DBNULLTERM);

/*
 ** The registered procedure has two parameters:
 **      name        varchar(255)
 **      newprice    int
 ** So pass these parameters to the registered
 ** procedure.
 */
 dbregparam (dbproc, "name", SYBVARCHAR, 6,
      "sybase");
 dbregparam (dbproc, "newprice", SYBINT4, -1,
      &newprice);
```

```
/*  Execute the registered procedure:*/
 status = dbregexec (dbproc, DBNOTIFYALL);
 if (status == FAIL)
 {
      fprintf(stderr, "ERROR:Failed to execute ¥
           price_change!¥n");
 }
 else if (status == DBNOPROC)
 {
      fprintf(stderr, "ERROR:Price_change does ¥
           not exist!¥n");
 }
 else
 {
      fprintf(stdout, "Success in executing ¥
           price_change!¥n");
 }
```

参照 dbreginit、dbregexec、dbnpdefine、dbnpcreate、dbregwatch

dbregwatch

説明 レジスタード・プロシージャが実行されたときにノーティフィケーションを受け取ることを要求します。

構文

```
RETCODE dbregwatch(dbproc, procedure_name, namelen,
          options)

DBPROCESS     *dbproc;
DBCHAR        *procedure_name;
DBSMALLINT    namelen;
DBUSMALLINT   options;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

procedure_name
レジスタード・プロシージャの名前を指すポインタです。レジスタード・プロシージャは、Open Server 内に定義されていなければなりません。

namelen

procedure_name のバイト単位の長さです。*procedure_name* が NULL で終了する場合は、*namelen* を DBNULLTERM として渡してください。

options

2 バイトのビットマスクです。DBWAIT、DBNOWAITONE、または DBNOWAITALL。

options が DBWAIT として渡された場合は、レジスタード・プロシージャが実行された同期ノーティフィケーションを DBPROCESS 接続が読み込むまでは、dbregwatch は呼び出し元に戻りません。

options が DBNOWAITONE として渡された場合は、dbregwatch は即座に呼び出し元に戻ります。DBPROCESS 接続は、レジスタード・プロシージャが実行されるときに非同期ノーティフィケーションを受信します。そのレジスタード・プロシージャが 2 回以上実行される場合でも、接続が受信するノーティフィケーションは 1 つだけです。

options が DBNOWAITALL として渡された場合は、dbregwatch は即座に呼び出し元に戻ります。DBPROCESS 接続は、レジスタード・プロシージャが実行されるときに非同期ノーティフィケーションを受信します。この接続は、ノーティフィケーションを今後受信しないことを Open Server に通知するまで、このレジスタード・プロシージャの実行のたびにノーティフィケーションを受け取ります。

戻り値

SUCCEED、DBNOPROC、または FAIL。

レジスタード・プロシージャに対してハンドラがインストールされていない場合、dbregwatch は FAIL を返します。

使用法

- dbregwatch は、特定のレジスタード・プロシージャが実行されたときに DBPROCESS 接続に通知することを Open Server に指示します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、dbregwatch は、特定のノーティフィケーション・プロシージャが実行されたときに DBPROCESS 接続に通知することも Open Server に指示します。
- 接続が受け取りを要求できるノーティフィケーションには、同期と非同期があります。
  - 同期ノーティフィケーションを要求するには、dbregwatch を呼び出すときに *options* を DBWAIT として渡します。この場合、レジスタード・プロシージャが実行されたというノーティフィケーションを DBPROCESS 接続が読み込むまで、dbregwatch は呼び出し元に戻りません。

Open Server は、同期ノーティフィケーション要求の結果としてノーティフィケーションを 1 つだけ送信します。要求された同期ノーティフィケーションが送信された後に、そのレジスタード・プロシージャが再び呼び出されても、別のノーティフィケーション要求が行われていなければ、クライアントは 2 回目のノーティフィケーションを受信しません。

- 非同期ノーティフィケーションを要求するには、dbregwatch を呼び出すときに *options* を DBNOWAITONE または DBNOWAITALL として渡します。この場合は、dbregwatch は即座に呼び出し元に戻ります。リターン・コード SUCCEED は、OpenServer が要求を受け取ったことを示します。

  options が DBNOWAITONE の場合は、レジスタード・プロシージャが 2 回以上実行される場合でも、Open Server が送信するノーティフィケーションは 1 つだけです。

  options が DBNOWAITALL の場合は、今後はノーティフィケーションを受け取らないことをクライアントが dbregnowatch を使用して Open Server に通知するまでは、引き続きレジスタード・プロシージャが実行されるたびに Open Server はノーティフィケーションを送信します。

- 非同期レジスタード・プロシージャ・ノーティフィケーションが到着したとき、DBPROCESS 接続の状態は、アイドル状態、コマンドの送信中、結果の読み込み中、または保留中の結果を持つアイドル状態のいずれかです。

  - DBPROCESS 接続がアイドル状態の場合は、接続がノーティフィケーションを読み込めるようにするために、アプリケーションで dbpoll を呼び出す必要があります。ノーティフィケーションに対するハンドラがインストールされている場合、そのハンドラは dbpoll が呼び出し元に戻る前に呼び出されます。
  - DBPROCESS 接続がコマンド送信中の場合は、ノーティフィケーションが読み込まれ、dbsqlexec または dbsqlok の実行中にノーティフィケーション・ハンドラが呼び出されます。ノーティフィケーション・ハンドラが呼び出し元に戻った後は、制御のフローは通常どおりに続行します。
  - DBPROCESS 接続が結果の読み込み中の場合は、ノーティフィケーションが読み込まれ、dbresults または dbnextrow の実行中にノーティフィケーション・ハンドラが呼び出されます。ノーティフィケーション・ハンドラが呼び出し元に戻った後は、制御のフローは通常どおりに続行します。

    - DBPROCESS 接続が保留中の結果を持つアイドル状態の場合は、ストリーム内の、ノーティフィケーションの直前までのすべての結果を接続が読み込んで処理するまで、ノーティフィケーションは読み込まれません。

- アプリケーションは、dbregwatch を呼び出す前に、レジスタード・プロシージャ・ノーティフィケーションを処理するハンドラをインストールしなければなりません。ハンドラがインストールされていない場合、dbregwatch は FAIL を返します。アプリケーションでノーティフィケーション・ハンドラをインストールするには、dbreghandle を使用します。

  アプリケーションが dbregwatch を呼び出してから、レジスタード・プロシージャ・ノーティフィケーションを受信するまでの間にハンドラのインストールが解除されると、DB-Library/C は、ノーティフィケーションの受信時にエラーを生成します。

- *procedure_name* によって参照されるプロシージャが Open Server 内に定義されていない場合は、dbregwatch は DBNOPROC を返します。Open Server 内に現在登録されているプロシージャのリストを取得するには、dbreglist を使用します。

- アプリケーションで dbregwatchlist を使用すると、監視しているレジスタード・プロシージャのリストを取得できます。

- 同期ノーティフィケーションを要求する例を次に示します。

```
DBPROCESS    *dbproc;
DBINT        handlerfunc;
DBINT        ret;

/*
 ** The registered procedure is defined in Open
 ** Server as:
 **     shutdown   msg_param   varchar(255)
 */

/*
 ** First install the handler for this registered
 ** procedure:
 */
 dbreghandle(dbproc, "shutdown", DBNULLTERM,
      handlerfunc);

/* Make the notification request and wait:*/
 ret = dbregwatch(dbproc, "shutdown", DBNULLTERM,
      DBWAIT);

if (ret == FAIL)
```

```
{
     fprintf(stderr, "ERROR:dbregwatch() ¥
          failed!¥n");
}
else if (ret == DBNOPROC)
{
     fprintf(stderr, "ERROR:procedure shutdown ¥
          not defined.¥n");
}
else
{
     /*
     ** The registered procedure notification has
     ** been returned, and our registered
     ** procedure handler has already been called.
     */
}
```

- 非同期ノーティフィケーションを要求する例を次に示します。

```
DBPROCESS     *dbproc;
DBINT         handlerfunc;
DBINT         ret;

/*
 ** The registered procedure is defined in Open
 ** Server as:
 **     shutdown   msg_param   varchar(255)
 */

/*
 ** First install the handler for this registered
 ** procedure:
 */
 dbreghandle(dbproc, "shutdown", DBNULLTERM,
      handlerfunc);

/* Make the asynchronous notification request:*/
 ret = dbregwatch(dbproc, "shutdown", DBNULLTERM,
      DBNOWAITALL);

if (ret == FAIL)
 {
     fprintf(stderr, "ERROR:dbregwatch() ¥
          failed!¥n");
 }
 else if (ret == DBNOPROC)
 {
     fprintf(stderr, "ERROR:procedure shutdown ¥
```

```
                not defined.¥n");
        }

        /*
         ** Since we are making use of the asychronous
         ** registered procedure notification mechanism,
         ** the application can continue doing other work
         ** while waiting for the notification. All we
         ** have to do is call dbpoll() once in a while to
         ** read the registered procedure notification
         ** when it arrives.
         */

        while (1)
         {
              /* Have dbpoll() block for one second */
              dbpoll (NULL, 1000, NULL, &ret);

             /*
              ** If we got the notification, then exit
              ** the loop
              */
              if (ret == DBNOTIFICATION)
                   break;
              /*  Handle other program tasks here */
         }
```

参照　dbpoll、dbregexec、dbregparam、dbreglist、dbregwatchlist、dbregnowatch

dbregwatchlist

説明　DBPROCESS が監視しているレジスタード・プロシージャのリストを返します。

構文　RETCODE dbregwatchlist(dbproc)

DBPROCESS　*dbproc;

パラメータ　dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　SUCCEED または FAIL。

使用法

- dbregwatchlist は、DBPROCESS 接続が監視しているレジスタード・プロシージャのリストを返します。ノーティフィケーション・プロシージャは特殊なタイプのレジスタード・プロシージャなので、dbregwatchlist によって返されるリストにはノーティフィケーション・プロシージャも含まれます。
- レジスタード・プロシージャのリストはローとして返されます。アプリケーションは、dbregwatchlist を呼び出した後にこのローを明示的に処理しなければなりません。それぞれのローは、DBPROCESS がノーティフィケーションを要求したレジスタード・プロシージャの名前を表します。ローには、SYBVARCHAR 型のカラムが 1 つだけ含まれています。
- アプリケーションでの dbregwatchlist の使用例を次に示します。

```
DBPROCESS     *dbproc;
DBCHAR        *procedurename;
DBINT         ret;

/* Request the list of procedures */
 if( (ret = dbregwatchlist(dbproc)) == FAIL)
 {
      /* Handle failure here */
 }
 dbresults(dbproc);
 while (dbnextrow(dbproc) != NO_MORE_ROWS)
 {
      procedurename = (DBCHAR *)dbdata(dbproc, 1);
      procedurename[dbdatlen(dbproc, 1)] = '¥0';

   fprintf(stdout, "we're waiting for ¥
          procedure '%s'.¥n", procedurename);
 }
 /* All done */
```

参照 dbregwatch、dbresults、dbnextrow

dbresults

説明 次のクエリの結果を設定します。

構文 RETCODE dbresults(dbproc)

DBPROCESS *dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED、FAIL または NO_MORE_RESULTS。

バッファ内のすべてのコマンドがすでに処理されている場合、dbresults は NO_MORE_RESULTS を返します。失敗の主な原因は、データベースのパーミッション違反などの実行時エラーです。

dbsqlexec または dbresults が実行時エラーをトラップするかどうかは、コマンド・バッファ内のコマンドの数によって決まります。バッファ内にコマンドが 1 つしかない場合、実行時エラーが発生すると dbsqlexec は失敗します。コマンド・バッファに複数のコマンドがある場合は、実行時エラーが発生しても dbsqlexec が失敗することはありません。代わりに、実行時エラーの原因となったコマンドを処理する dbresults の呼び出しが失敗します。

実行時エラーとストアド・プロシージャについては、状況がやや複雑になります。前述のルールにより、execute コマンドの実行時エラーが発生すると、dbresults は失敗します。しかし、ストアド・プロシージャ内の文で実行時エラーが発生しても、dbresults は失敗しません。たとえば、ストアド・プロシージャに insert 文が含まれているけれども、データベース・テーブルへの挿入のパーミッションがユーザに付与されていない場合は、insert 文は失敗しますが、dbresults はこの場合も SUCCEED を返します。ストアド・プロシージャ内で実行時エラーが発生したかどうかを調べるには、dbretstatus ルーチンを使用してプロシージャのリターン・ステータスを確認し、関連するサーバ・メッセージをメッセージ・ハンドラ内でトラップしてください。

使用法

- このルーチンは、コマンド・バッチ内の次のコマンドの処理を設定します。アプリケーションは、dbsqlexec または dbsqlok が SUCCEED を返した後で、このルーチンを呼び出します。dbsqlexec または dbsqlok の呼び出しから SUCCEED が返されていれば、dbresults の最初の呼び出しでは必ず SUCCEED または NO_MORE_RESULTS が返されます。dbresults から SUCCEED が返されたときは、アプリケーションは、一般に dbnextrow を使用して結果ローを処理します。

- コマンド・バッチ内にコマンドが 1 つしか含まれておらず、そのコマンドが「use database」などのローを返さないコマンドの場合は、DB-Library/C アプリケーションでコマンドの結果を処理するために dbresults を呼び出す必要はありません。ただし、コマンド・バッチ内に複数のコマンドがある場合は、コマンドがローを返すものかどうかにかかわらず、バッチ内のコマンドのそれぞれに対して 1 回ずつ dbresults を呼び出す必要があります。

  また、コマンド・バッチ内でストアド・プロシージャを実行する場合は、そのストアド・プロシージャがローを返すものかどうかにかかわらず、dbresults を最低 1 回は呼び出す必要があります。ストアド・プロシージャに複数の Transact-SQL select が含まれている場合は、select のそれぞれに対して 1 回ずつ dbresults を呼び出す必要があります。

  dbresults を適切な回数だけ呼び出すために、dbresults の呼び出しは常にループ内で行い、dbresults が NO_MORE_RESULTS を返したときにこのループを終了するようにすることを強くおすすめします。

  注意 dbresults は、すべての Transact-SQL コマンドをコマンドであるとみなします。Transact-SQL コマンドのリストについては、『ASE リファレンス・マニュアル』を参照してください。

- コマンド・バッチに残っている結果をキャンセルするには、dbcancel を呼び出してください。このようにすれば、NO_MORE_RESULTS が返されるまで dbresults を繰り返し呼び出さなくても済みます。
- 特定のコマンドが、ローを返すものであり、dbnextrow で結果を処理する必要があるかどうかを判断するには、dbresults の呼び出しの後に DBROWS を呼び出してください。
- dbresults を dbsqlexec とともに使用する場合の標準的な呼び出しのシーケンスを次に示します。

```
DBINT        xvariable;
DBCHAR       yvariable[10];
RETCODE      return_code;

/* Read the query into the command buffer */
 dbcmd(dbproc, "select x = 100, y = 'hello'");

/* Send the query to Adaptive Server Enterprise */
 dbsqlexec(dbproc);

/*
 ** Get ready to process the results of the query.
```

```
** Note that dbresults is called in a loop even
** though only a single set of results is expected.
** This is simply because it is good programming
** practice to always code dbresults calls in loop.
*/

while ((return_code
 =dbresults(dbproc)!=NO_MORE_RESULTS)
 {
     if ((return_code == SUCCEED)
           & & (DBROWS(dbproc) == SUCCEED))
     {
         /* Bind column data to program variables */
         dbbind(dbproc, 1, INTBIND, (DBINT) 0,
                (BYTE *) &xvariable);
         dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
                 yvariable);

       /* Now process each row */
         while (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
           C-code to print or process row data
        }
    }
 }
```

オンラインのサンプル・プログラムの *example1.c* では、マルチクエリ・コマンド・バッチを処理するための dbresults の使い方を示しています。

- アプリケーションで複数の入力データ・ストリームを効率的に管理するために、DB-Library ネットワーク・バッファまたはネットワーク自体にまだ読み込んでいないバイトがあるかどうかを確認できます。アプリケーションは、次のいずれかを行うことができます。
  - (UNIX のみ) ネットワーク・バッファにバイトが残っているかどうかをテストするには、DBRBUF を呼び出します。ネットワーク自体にバイトが残っているかどうかをテストするには、DBIORDESC と UNIX の select を呼び出します。
  - (すべてのシステム) dbpoll を呼び出します。
- dbresults のもう 1 つの用途は、dbrpcsend で実行するリモート・プロシージャ・コールの結果を処理することです。詳細については、dbrpcsend のリファレンス・ページを参照してください。

参照　dbbind、dbcancel、dbnextrow、dbpoll、DBRBUF、dbretstatus、DBROWS、dbrpcsend、dbsqlexec、dbsqlok

dbretdata

説明

ストアド・プロシージャによって生成されたリターン・パラメータ値を指すポインタを返します。

構文

```
BYTE *dbretdata(dbproc, retnum)

DBPROCESS    *dbproc;
int          retnum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

retnum

対象となるリターン・パラメータ値の番号です。最初のリターン・パラメータ値は 1 です。値が返される順序は、ストアド・プロシージャの create procedure 文でパラメータが指定された順序と同じです (これは、リモート・プロシージャ・コールで指定された順序と必ずしも同じではないことに注意してください)。*retnum* を指定するとき、リターン・パラメータ以外のパラメータは数えません。たとえば、ストアド・プロシージャの 2 番目のパラメータだけがリターン・パラメータである場合は、その *retnum* は 2 ではなく 1 になります。

戻り値

指定したリターン・パラメータ値を指すポインタです。*retnum* が範囲外の場合、dbretdata は NULL を返します。データの値が本当に NULL である (単に *retnum* が範囲外であるのではない) かどうかを調べるには、dbretlen の戻り値が 0 であることを確認してください。

使用法

- dbretdata は、ストアド・プロシージャによって生成されたリターン・パラメータ値を指すポインタを返します。これは、リモート・プロシージャ・コールや、ストアド・プロシージャに対する execute 文とともに使用すると便利です。
- Transact-SQL ストアド・プロシージャは、指定された「リターン・パラメータ」の値を返すことができます。ストアド・プロシージャを呼び出したアプリケーションでは、そのプロシージャ内で変更されたリターン・パラメータの値を使用できます。これは、いくつかのプログラミング言語で使用可能な「参照渡し」の機能に似ています。

パラメータがリターン・パラメータとして機能するには、ストアド・プロシージャ内でそのように宣言されていなければなりません。ストアド・プロシージャを呼び出す execute 文およびリモート・プロシージャ・コールも同様に、そのパラメータがリターン・パラメータとして機能することを示していなければなりません。リモート・プロシージャ・コールの場合は、dbrpcparam ルーチンを使用して、パラメータがリターン・パラメータであるかどうかを指定します。

- ストアド・プロシージャを実行するとき、サーバは、他のすべての結果を返した直後にパラメータ値を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbretdata を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションは、dbretdata、またはリターン・パラメータを処理する他のルーチンを呼び出す前に、必要な回数だけ dbresults と dbnextrow を呼び出してすべての結果を処理する必要があります)。
- ストアド・プロシージャをリモート・プロシージャ・コールで呼び出した場合、リターン・パラメータ値は自動的にアプリケーションで使用できるようになります。一方、ストアド・プロシージャを execute 文で呼び出した場合は、その execute 文が含まれているコマンド・バッチが定数ではなくローカル変数をリターン・パラメータとして使用している場合にだけ、リターン・パラメータ値が使用可能となります。
- dbnumrets ルーチンは、使用可能なリターン・パラメータ値の数を示します。dbnumrets が 0 以下を返した場合、使用可能なリターン・パラメータはありません。
- RPC コマンドを使用してストアド・プロシージャを呼び出すときは (dbrpcinit、dbrpcparam、dbrpcsend を使用)、リターン・パラメータ値は、他の結果がすべて処理された後で取り出すことができます。この使用方法については、オンラインのサンプル・プログラムの *example8.c* を参照してください。
- ストアド・プロシージャを Transact-SQL コマンドのバッチから実行した場合は (dbsqlexec または dbsqlsend を使用)、ストアド・プロシージャの実行後に他のコマンドが実行される可能性があります。このため、リターン・パラメータ値の検索が多少複雑になります。

- ストアド・プロシージャ・コマンドがバッチ内にある唯一のコマンドであることが確実な場合は、オンラインのサンプル・プログラムの *example8.c* のように、dbresults ループの後でリターン・パラメータ値を取り出します。
- バッチ内で複数のコマンドが使用されている可能性がある場合は、dbnextrow を使用してすべてのローをフェッチしてから、dbresults ループ内でリターン・パラメータ値を検索してください。次のコードは、そのような状況でリターン・パラメータ値を取得すべき場所を示したものです。

```
while ( (result_code = dbresults(dbproc)
         != NO_MORE_RESULTS)
 {
   if (result_code == SUCCEED)
   {
     ... bind rows here ...
     while ((row_code = dbnextrow(dbproc))
            != NO_MORE_ROWS)
     {
       ... process rows here ...
     }
     /* Now check for a return status */
     if (dbhasretstat(dbproc) == TRUE
     {
       printf("(return status %d)¥n",
              dbretstatus(dbproc));
     }
     /* Now check for return parameter values */
     if (dbnumrets(dbproc) > 0)
     {
       ... retrieve output parameters here ...
     }
   } /* if result_code */
   else
   {
     printf("Query failed.¥n");
   }
 } /* while dbresults */
```

- 次に、リターン・パラメータ値の取得に使用するルーチンを示します。
  - dbnumrets は、リターン・パラメータ値の総数を返します。
  - dbretlen は、パラメータ値の長さを返します。
  - dbretname は、パラメータ値の名前を返します。

- dbrettype は、パラメータ値のデータ型を返します。
- dbconvert は、必要に応じて値を他のデータ型に変換します。

次のコードでは、これらのルーチンを組み合わせて使用する例を示します。

```
char      dataval[512];
char      *dataname;
DBINT     datalen;
int i,    numrets;
numrets = dbnumrets(dbproc);

 for (i = 1; i <= numrets; i++)
 {
   dataname = dbretname(dbproc, i);
   datalen = dbretlen(dbproc, i);
   if (datalen == 0)
   {
     /* The parameter's value is NULL */
     strcpy(dataval, "NULL");
   }
   else
   {
     /*
     ** Convert to char. dbconvert appends a null
     ** terminator because we pass the last
     ** parameter, destlen, as -1.
     */
     result = dbconvert(dbproc,
              dbrettype(dbproc, i),
              dbretdata(dbproc, i), datalen,
              SYBCHAR, (BYTE *)dataval, -1);
   } /* else */
   /* Now print out the converted value */
   if (dataname == NULL || *dataname == '¥0')
     printf("¥t%s¥n", dataval);
   else
     printf("¥t%s:%s¥n", dataname, dataval);
 }
```

参照　dbnextrow、dbnumrets、dbresults、dbretlen、dbretname、dbrettype、dbrpcinit、dbrpcparam

dbretlen

説明

ストアド・プロシージャによって生成されたリターン・パラメータ値の長さを調べます。

構文

```
DBINT dbretlen(dbproc, retnum)

DBPROCESS    *dbproc;
int          retnum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

retnum

対象となるリターン・パラメータ値の番号です。最初のリターン・パラメータ値は 1 です。値が返される順序は、ストアド・プロシージャの create procedure 文でパラメータが指定された順序と同じです (これは、リモート・プロシージャ・コールで指定された順序と必ずしも同じではないことに注意してください)。*retnum* を指定するとき、リターン・パラメータ以外のパラメータは数えません。たとえば、ストアド・プロシージャの 2 番目のパラメータだけがリターン・パラメータである場合は、その *retnum* は 2 ではなく 1 になります。

戻り値

指定されたリターン・パラメータ値の長さです。*retnum* が範囲外の場合、dbretlen は -1 を返します。リターン・パラメータ値が NULL の場合、dbretlen は 0 を返します。

使用法

- dbretlen は、ストアド・プロシージャによって生成された特定のリターン・パラメータ値の長さを返します。これは、リモート・プロシージャ・コールや、ストアド・プロシージャに対する execute 文とともに使用すると便利です。
- Transact-SQL ストアド・プロシージャは、指定された「リターン・パラメータ」の値を返すことができます。ストアド・プロシージャを呼び出したアプリケーションでは、そのプロシージャ内で変更されたリターン・パラメータの値を使用できます。これは、いくつかのプログラミング言語で使用可能な「参照渡し」の機能に似ています。

パラメータがリターン・パラメータとして機能するには、ストアド・プロシージャ内でそのように宣言されていなければなりません。ストアド・プロシージャを呼び出す execute 文およびリモート・プロシージャ・コールも同様に、そのパラメータがリターン・パラメータとして機能することを示していなければなりません。リモート・プロシージャ・コールの場合は、dbrpcparam ルーチンを使用して、パラメータがリターン・パラメータであるかどうかを指定します。

- ストアド・プロシージャを実行するとき、サーバは、他のすべての結果を返した直後にパラメータ値を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbretlen を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションは、dbretlen、またはリターン・パラメータを処理する他のルーチンを呼び出す前に、必要な回数だけ dbresults と dbnextrow を呼び出してすべての結果を処理する必要があります)。
- ストアド・プロシージャをリモート・プロシージャ・コールで呼び出した場合、リターン・パラメータ値は自動的にアプリケーションで使用できるようになります。一方、ストアド・プロシージャを execute 文で呼び出した場合は、その execute 文が含まれているコマンド・バッチが定数ではなくローカル変数をリターン・パラメータとして使用している場合にだけ、リターン・パラメータ値が使用可能となります。
- 次に、リターン・パラメータ値についてその他の情報を返すルーチンを示します。
  - dbnumrets は、リターン・パラメータ値の総数を返します。
  - dbretdata は、パラメータ値を指すポインタを返します。
  - dbretname は、パラメータ値の名前を返します。
  - dbrettype は、パラメータ値のデータ型を返します。
  - dbconvert は、必要に応じて値を他のデータ型に変換します。
- このルーチンの例については、dbretdata のリファレンス・ページを参照してください。

参照　dbnextrow、dbnumrets、dbresults、dbretdata、dbretname、dbrettype、dbrpcinit、dbrpcparam

dbretname

説明

特定のリターン・パラメータ値に関連付けられているストアド・プロシージャ・パラメータの名前を調べます。

構文

```
char *dbretname(dbproc, retnum)

DBPROCESS    *dbproc;
int          retnum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

retnum

対象となるリターン・パラメータ値の番号です。最初のリターン・パラメータ値は 1 です。値が返される順序は、ストアド・プロシージャの create procedure 文でパラメータが指定された順序と同じです (これは、リモート・プロシージャ・コールで指定された順序と必ずしも同じではないことに注意してください)。*retnum* を指定するとき、リターン・パラメータ以外のパラメータは数えません。たとえば、ストアド・プロシージャの 2 番目のパラメータだけがリターン・パラメータである場合は、その *retnum* は 2 ではなく 1 になります。

戻り値

指定されたリターン・パラメータ値の、NULL で終了するパラメータ名を指すポインタです。*retnum* が範囲外の場合、dbretname は NULL を返します。

使用法

- dbretname は、ストアド・プロシージャからのリターン・パラメータ値に関連付けられている、NULL で終了するパラメータ名を指すポインタを返します。これは、リモート・プロシージャ・コールや、ストアド・プロシージャに対する execute 文とともに使用すると便利です。
- Transact-SQL ストアド・プロシージャは、指定された「リターン・パラメータ」の値を返すことができます。ストアド・プロシージャを呼び出したアプリケーションでは、そのプロシージャ内で変更されたリターン・パラメータの値を使用できます。これは、いくつかのプログラミング言語で使用可能な「参照渡し」の機能に似ています。

パラメータがリターン・パラメータとして機能するには、ストアド・プロシージャ内でそのように宣言されていなければなりません。ストアド・プロシージャを呼び出す execute 文およびリモート・プロシージャ・コールも同様に、そのパラメータがリターン・パラメータとして機能することを示していなければなりません。リモート・プロシージャ・コールの場合は、dbrpcparam ルーチンを使用して、パラメータがリターン・パラメータであるかどうかを指定します。

- ストアド・プロシージャを実行するとき、サーバは、他のすべての結果を返した直後にパラメータ値を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbretname を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションは、dbretname、またはリターン・パラメータを処理する他のルーチンを呼び出す前に、必要な回数だけ dbresults と dbnextrow を呼び出してすべての結果を処理する必要があります)。
- ストアド・プロシージャをリモート・プロシージャ・コールで呼び出した場合、リターン・パラメータ値は自動的にアプリケーションで使用できるようになります。一方、ストアド・プロシージャを execute 文で呼び出した場合は、その execute 文が含まれているコマンド・バッチが定数ではなくローカル変数をリターン・パラメータとして使用している場合にだけ、リターン・パラメータ値が使用可能となります。
- 次に、リターン・パラメータ値についてその他の情報を返すルーチンを示します。
  - dbnumrets は、リターン・パラメータ値の総数を返します。
  - dbretdata は、パラメータ値を指すポインタを返します。
  - dbretlen は、パラメータ値の長さを返します。
  - dbrettype は、パラメータ値のデータ型を返します。
  - dbconvert は、必要に応じて値を他のデータ型に変換します。
- このルーチンの例については、dbretdata のリファレンス・ページを参照してください。

参照　dbnextrow、dbnumrets、dbresults、dbretdata、dbretlen、dbrettype、dbrpcinit、dbrpcparam

dbretstatus

説明

現在のコマンドまたはリモート・プロシージャ・コールによって返されたストアド・プロシージャのステータス番号を調べます。

構文

```
DBINT dbretstatus(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

現在のコマンドに対するリターン・ステータス番号です。

使用法

- dbretstatus は、ストアド・プロシージャのステータス番号をフェッチします。Adaptive Server Enterprise で実行されるすべてのストアド・プロシージャは、ステータス番号を返します。正常に終了したストアド・プロシージャは、ステータス番号 0 を返します。リターン・ステータス番号のリストについては、『リファレンス・マニュアル』を参照してください。
- dbhasretstat ルーチンは、現在の Transact-SQL コマンドまたはリモート・プロシージャ・コールが実際にリターン・ステータス番号を生成したかどうかを判断します。ステータス番号はストアド・プロシージャの機能の 1 つなので、リモート・プロシージャ・コール、またはストアド・プロシージャを実行する Transact-SQL コマンドだけがステータス番号を生成できます。
- ストアド・プロシージャを実行するとき、サーバは、その他のすべての結果を返した直後にステータス番号を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbretstatus を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションは、dbretstatus または dbhasretstat を呼び出す前に、必要な回数だけ dbresults と dbnextrow を呼び出してすべての結果を処理する必要があります)。
- アプリケーションがステータス番号およびリターン・パラメータ値を処理する順序は重要ではありません。

- ストアド・プロシージャを Transact-SQL コマンドのバッチから実行した場合は (dbsqlexec または dbsqlsend を使用)、ストアド・プロシージャの実行後に他のコマンドが実行される可能性があります。このような状況では、リターン・ステータスの取得が多少複雑になります。
  - ストアド・プロシージャ・コマンドがバッチ内にある唯一のコマンドであることが確実な場合は、オンラインのサンプル・プログラムの *example8.c* のように、dbresults ループの後でリターン・ステータスを取り出します。
  - バッチ内に複数のコマンドが存在する可能性がある場合は、リターン・ステータスの取り出しは、dbresults ループの内側で、dbnextrow を使用してすべてのローをフェッチした後に行ってください。このような状況でリターン・ステータスを取り出す方法については、dbhasretstat のリファレンス・ページを参照してください。
- このルーチンの例については、dbhasretstat のリファレンス・ページを参照してください。

参照　dbhasretstat、dbnextrow、dbresults、dbretdata、dbrpcinit、dbrpcparam、dbrpcsend

dbrettype

説明　ストアド・プロシージャによって生成されたリターン・パラメータ値のデータ型を調べます。

構文

```
int dbrettype(dbproc, retnum)

DBPROCESS    *dbproc;
int          retnum;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

retnum

対象となるリターン・パラメータ値の番号です。最初のリターン・パラメータ値は 1 です。値が返される順序は、ストアド・プロシージャの create procedure 文でパラメータが指定された順序と同じです (これは、リモート・プロシージャ・コールで指定された順序と必ずしも同じではないことに注意してください)。*retnum* を指定するとき、リターン・パラメータ以外のパラメータは数えません。たとえば、ストアド・プロシージャの 2 番目のパラメータだけがリターン・パラメータである場合は、その *retnum* は 2 ではなく 1 になります。

戻り値

指定されたリターン・パラメータ値のデータ型を表すトークン値です。

このルーチンから返されるトークン値は、カラムのサーバ・データ型と正確に対応しない場合もあります。

- SYBVARCHAR は、SYBCHAR として返されます。
- SYBVARBINARY は、SYBBINARY として返されます。
- SYBDATETIMN は、SYBDATETIME として返されます。
- SYBMONEYN は、SYBMONEY として返されます。
- SYBFLTN は、SYBFLT8 として返されます。
- SYBINTN は、実際の SYBINTN のデータ型に応じて、SYBINT1、SYBINT2、または SYBINT4 として返されます。

  retnum が範囲外の場合は、-1 が返されます。

使用法

- dbrettype は、ストアド・プロシージャが生成したリターン・パラメータ値のデータ型を返します。これは、リモート・プロシージャ・コールや、ストアド・プロシージャに対する execute 文とともに使用すると便利です。
- Transact-SQL ストアド・プロシージャは、指定された「リターン・パラメータ」の値を返すことができます。ストアド・プロシージャを呼び出したアプリケーションでは、そのプロシージャ内で変更されたリターン・パラメータの値を使用できます。これは、いくつかのプログラミング言語で使用可能な「参照渡し」の機能に似ています。

パラメータがリターン・パラメータとして機能するには、ストアド・プロシージャ内でそのように宣言されていなければなりません。ストアド・プロシージャを呼び出す execute 文およびリモート・プロシージャ・コールも同様に、そのパラメータがリターン・パラメータとして機能することを示していなければなりません。リモート・プロシージャ・コールの場合は、dbrpcparam ルーチンを使用して、パラメータがリターン・パラメータであるかどうかを指定します。

- ストアド・プロシージャを実行するとき、サーバは、他のすべての結果を返した直後にパラメータ値を返します。したがって、アプリケーションは、dbresults および dbnextrow (該当する場合) を呼び出してストアド・プロシージャの結果を処理した後でなければ、dbrettype を呼び出すことはできません (ストアド・プロシージャは、その中の select ごとに 1 つずつ、複数の結果セットを生成します)。アプリケーションは、dbrettype、またはリターン・パラメータを処理する他のルーチンを呼び出す前に、必要な回数だけ dbresults と dbnextrow を呼び出してすべての結果を処理する必要があります。)。
- ストアド・プロシージャをリモート・プロシージャ・コールで呼び出した場合、リターン・パラメータ値は自動的にアプリケーションで使用できるようになります。一方、ストアド・プロシージャを execute 文で呼び出した場合は、その execute 文が含まれているコマンド・バッチが定数ではなくローカル変数をリターン・パラメータとして使用している場合にだけ、リターン・パラメータ値が使用可能となります。
- dbrettype は、実際にはデータ型を整数で表したトークン値 (SYBCHAR、SYBFLT8 など) を返します。トークン値を読み込み可能なトークン文字列に変換するには、dbprtype を使用してください。すべてのトークン値とそれに相当するトークン文字列のリストについては、dbprtype のリファレンス・ページを参照してください。
- サーバ・データ型のリストについては、「データ型」(443 ページ) を参照してください。
- 次に、リターン・パラメータ値についてその他の情報を返すルーチンを示します。
  - dbnumrets は、リターン・パラメータ値の総数を返します。
  - dbretdata は、パラメータ値を指すポインタを返します。
  - dbretlen は、パラメータ値の長さを返します。

- dbretname は、パラメータ値の名前を返します。
- dbconvert は、必要に応じて値を他のデータ型に変換します。

- このルーチンの例については、dbretdata のリファレンス・ページを参照してください。

参照

dbnextrow、dbnumrets、dbprtype、dbresults、dbretdata、dbretlen、dbretname、dbrpcinit、dbrpcparam

DBROWS

説明

現在のコマンドが実際にローを返したかどうかを示します。

構文

```
RETCODE DBROWS(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL で、現在のコマンドがローを返したかどうかを示します。

使用法

- このマクロは、dbresults で現在処理中のコマンドがローを返したかどうかを調べます。アプリケーションは、dbresults が SUCCEED を返した後に、このルーチンを呼び出すことができます。
- dbnextrow の後に DBROWS を呼び出してはなりません。そのようにすると、マクロは誤った結果を返すことがあります。
- DBROWS を使用すると、dbnextrow を呼び出して結果ローを処理する必要があるかどうかを判断できます。DBROWS が FAIL を返した場合は、dbnextrow の呼び出しを省略できます。
- DBCMDROW マクロは、現在のコマンドがローを返せるもの (Transact-SQL の select 文か、select が含まれているストアド・プロシージャを実行する execute) であるかどうかを調べます。

参照

DBCMDROW、dbnextrow、dbresults、DBROWTYPE

DBROWTYPE

説明　現在のローのタイプを返します。

構文　STATUS DBROWTYPE(dbproc)

```
DBPROCESS    *dbproc;
```

パラメータ　dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　次の 3 つのタイプの値が返されます。

- 現在のローが通常ローの場合は、REG_ROW が返されます。
- 現在のローが計算ローの場合は、ローの *computeid* が返されます (*computeid* については、dbaltbind のリファレンス・ページを参照してください)。
- ローが読み込まれていない場合、またはルーチンが何らかの理由で失敗した場合は、NO_MORE_ROWS が返されます。

使用法
- このマクロは、現在のローのタイプ (通常ローまたは計算ロー) を返します。dbnextrow もローのタイプを返すため、通常はローのタイプはすでにわかっています。

参照　dbnextrow

dbrpcinit

説明　リモート・プロシージャ・コールを初期化します。

構文　RETCODE dbrpcinit(dbproc, rpcname, options)

```
DBPROCESS    *dbproc;
char         *rpcname;
DBSMALLINT   options;
```

パラメータ　dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

rpcname
実行するストアド・プロシージャの名前を指すポインタです。

options
RPC オプションの 2 バイトのビットマスクです。現時点では、使用可能なオプションは DBRPCRECOMPILE だけです。これは、ストアド・プロシージャを実行する前に再コンパイルするためのオプションです。

戻り値

SUCCEED または FAIL。

使用法

- アプリケーションは、2 通りの方法でストアド・プロシージャを呼び出すことができます。Transact-SQL の execute 文が含まれているコマンド・バッファを実行する方法と、リモート・プロシージャ・コール (RPC) を行う方法です。
- リモート・プロシージャ・コールには、execute 文と比較して、次のような利点があります。
  - RPC は、ネイティブのデータ型でストアド・プロシージャのパラメータを渡します。それに対し、execute 文は ASCII 文字でパラメータを渡します。したがって、RPC は execute 文より高速で、通常、execute 文より簡潔です。これは、アプリケーション・プログラムとサーバによる、ネイティブのデータ型とそれに対応する ASCII 文字間での変換を必要としないためです。
  - ストアド・プロシージャのリターン・パラメータを使用する場合は、RPC の方が、execute 文に比べて簡単かつ高速です。RPC を使用するときは、自動的にアプリケーションでリターン・パラメータを使用できるようになります (ただし、dbrpcparam ルーチンを使用してパラメータを RPC に追加するときに、リターン・パラメータであることを指定する必要があります)。一方、execute 文を使用してストアド・プロシージャを呼び出す場合は、その execute 文が含まれているコマンド・バッチが定数ではなくローカル変数をリターン・パラメータとして使用している場合にだけ、リターン・パラメータ値が使用可能となります。このとき、コマンド・バッチが実行されるたびに追加の解析処理が行われます。

- リモート・プロシージャ・コールを行うには、まず、dbrpcinit を呼び出して、呼び出すストアド・プロシージャを指定します。次に、ストアド・プロシージャの各パラメータに対して一度ずつ、dbrpcparam を呼び出します。最後に、dbrpcsend を呼び出して、パラメータ・リストの終了を示します。これで、サーバは指定されたプロシージャの実行を開始します。この後、dbsqlok、dbresults、dbnextrow を呼び出して、ストアド・プロシージャの結果を処理できます (ストアド・プロシージャに複数の select 文が含まれている場合は、dbresults を 2 回以上呼び出す必要があることに注意してください)。ストアド・プロシージャの結果がすべて処理された後で、dbretdata や dbretstatus などの、リターン・パラメータやステータス番号を処理するルーチンを呼び出すことができます。
- 実行されているプロシージャが存在するサーバが、アプリケーションが直接接続しているものではない場合は、プロシージャ内で実行されるコマンドをロールバックすることはできません。
- リモート・プロシージャ・コールの使用例は、オンラインのサンプル・プログラムの *example8.c* を参照してください。

参照　dbnextrow、dbresults、dbretdata、dbretstatus、dbrpcparam、dbrpcsend、dbsqlok

dbrpcparam

説明　リモート・プロシージャ・コールにパラメータを追加します。

構文

```
RETCODE dbrpcparam(dbproc, paramname, status, type,
        maxlen, datalen, value)

DBPROCESS    *dbproc;
char         *paramname;
BYTE         status;
int          type;
DBINT        maxlen;
DBINT        datalen;
BYTE         *value;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

paramname
呼び出されるパラメータの名前を指すポインタです。この名前は「@」で始まるものでなければなりません。この記号は、すべてのストアド・プロシージャのパラメータ名の先頭にあります。Transact-SQL の execute 文での場合と同様に、この名前は省略可能です。名前を使用しない場合は、NULL を指定してください。この場合、dbrpcparam 呼び出しの順序が、それぞれが参照するパラメータを決定します。

status
RPC パラメータ・オプションの 1 バイトのビットマスクです。現時点では、使用可能なオプションは DBRPCRETURN だけです。これは、アプリケーション・プログラムがこのパラメータをリターン・パラメータとして使用することを示すものです。

type
パラメータのデータ型を示す記号定数です (たとえば、SYBINT1、SYBCHAR など)。パラメータ値がサーバに渡されるときのデータ型は、対応するストアド・プロシージャのパラメータが定義されている Adaptive Server Enterprise データ型と一致する必要があります。型定数と対応する Adaptive Server Enterprise のデータ型のリストについては、「データ型」(443 ページ) を参照してください。

maxlen
リターン・パラメータの場合は、この値はストアド・プロシージャから返される RPC パラメータ値の最大バイト長です。*maxlen* が関係するのは、データ長が固定でないデータ型、つまり char、text、binary、image の値のみです。このパラメータが適用されない場合 (つまり、*type* が SYBINT2 などの固定長データ型である場合)、またはリターン・パラメータの長さを制限しない場合は、*maxlen* に -1 を指定してください。また、リターン・パラメータとして指定しないパラメータの場合も *maxlen* に -1 を指定してください。

datalen

ストアド・プロシージャに渡す RPC パラメータのバイト単位での長さです。この長さには、NULL ターミネータは含めません。

type が SYBCHAR、SYBVARCHAR、SYBBINARY、SYBVARBINARY、SYBBOUNDARY、または SYBSENSITIVITY, である場合は、*datalen* を指定しなければなりません。これらのデータ型の場合に *datalen* を -1 として渡すと、*dbproc* が参照する DBPROCESS は「dead」とマーク付けされる、つまり使用不可能になります。

type が SYBINT2 などの固定長データ型の場合は、*datalen* は -1 として渡してください。

RPC パラメータの値が NULL の場合は、*type* が固定長データ型であっても、*datalen* を 0 として渡してください。

value

RPC パラメータ自体を指すポインタです。*datalen* が 0 の場合、このポインタは無視され、NULL として扱われます。アプリケーションが dbrpcsend を呼び出すまでは、**value* は DB-Library の内部バッファ領域にコピーされないことに注意してください。dbrpcsend を呼び出すまでは、アプリケーションで **value* を上書きしてはなりません。

type の値は **value* のデータ型を示します。「データ型」(443 ページ) を参照してください。SYBDATETIME、SYBMONEY、SYBNUMERIC、SYBDECIMAL のように、同じ C 言語データ型がないものについては、dbconvert_ps を使用して **value* を初期化してください。

注意　アプリケーションは、dbrpcsend を呼び出してリモート・プロシージャ・コールをサーバへ送信するまでは、**value* を上書きしてはなりません。これは、DB-Library の前のバージョンからの機能変更です。

戻り値

SUCCEED または FAIL。

使用法

- アプリケーションは、2 通りの方法でストアド・プロシージャを呼び出すことができます。Transact-SQL の execute 文が含まれているコマンド・バッファを実行する方法と、リモート・プロシージャ・コール (RPC) を行う方法です。この 2 つの方法の相違点については、dbrpcinit のリファレンス・ページを参照してください。

- リモート・プロシージャ・コールを行うには、まず、dbrpcinit を呼び出して、呼び出すストアド・プロシージャを指定します。次に、ストアド・プロシージャの各パラメータに対して一度ずつ、dbrpcparam を呼び出します。最後に、dbrpcsend を呼び出して、パラメータ・リストの終了を示します。これで、サーバは指定されたプロシージャの実行を開始します。この後、dbsqlok、dbresults、dbnextrow を呼び出して、ストアド・プロシージャの結果を処理できます (ストアド・プロシージャに複数の select 文が含まれている場合は、dbresults を 2 回以上呼び出す必要があることに注意してください)。ストアド・プロシージャの結果がすべて処理された後で、dbretdata や dbretstatus などの、リターン・パラメータやステータス番号を処理するルーチンを呼び出すことができます。
- *type* が SYBCHAR、SYBVARCHAR、SYBBINARY、SYBVARBINARY、SYBBOUNDARY、SYBSENSITIVITY である場合は、*datalen* を指定しなければなりません。これらのデータ型の場合に *datalen* を -1 として渡すと、*dbproc* が参照する DBPROCESS は「dead」とマーク付けされる、つまり使用不可能になります。
- *type* が SYBNUMERIC または SYBDECIMAL である場合は、dbconvert_ps を使用して ***value* 内の DBNUMERIC または DBDECIMAL の値を初期化し、精度と位取りを指定してください。
- 実行されているプロシージャが存在するサーバが、アプリケーションが直接接続しているものではない場合は、プロシージャ内で実行されるコマンドをロールバックすることはできません。
- リモート・プロシージャ・コールの使用例は、オンラインのサンプル・プログラムの *example8.c* を参照してください。

参照

dbnextrow、dbresults、dbretdata、dbretstatus、dbrpcinit、dbrpcsend、dbsqlok

dbrpcsend

説明

リモート・プロシージャ・コールの終わりを知らせます。

構文

```
RETCODE dbrpcsend(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

使用法

- アプリケーションは、2 通りの方法でストアド・プロシージャを呼び出すことができます。Transact-SQL の execute 文が含まれているコマンド・バッファを実行する方法と、リモート・プロシージャ・コール (RPC) を行う方法です。この 2 つの方法の相違点については、dbrpcinit のリファレンス・ページを参照してください。
- リモート・プロシージャ・コールを行うには、まず、dbrpcinit を呼び出して、呼び出すストアド・プロシージャを指定します。次に、ストアド・プロシージャの各パラメータに対して一度ずつ、dbrpcparam を呼び出します。最後に、dbrpcsend を呼び出して、パラメータ・リストの終了を示します。これで、サーバは指定されたプロシージャの実行を開始します。この後、dbsqlok、dbresults、dbnextrow を呼び出して、ストアド・プロシージャの結果を処理できます (ストアド・プロシージャに複数の select 文が含まれている場合は、dbresults を 2 回以上呼び出す必要があることに注意してください)。ストアド・プロシージャの結果がすべて処理された後で、dbretdata や dbretstatus などの、リターン・パラメータやステータス番号を処理するルーチンを呼び出すことができます。
- 実行されているプロシージャが存在するサーバが、アプリケーションが直接接続しているものではない場合は、プロシージャ内で実行されるコマンドをロールバックすることはできません。
- リモート・プロシージャ・コールの使用例は、オンラインのサンプル・プログラムの *example8.c* を参照してください。

参照

dbnextrow、dbresults、dbretdata、dbretstatus、dbrpcinit、dbrpcparam、dbsqlok

dbrpwclr

説明

LOGINREC 構造体からすべてのリモート・パスワードをクリアします。

構文

```
void dbrpwclr(loginrec)

LOGINREC     *loginrec;
```

パラメータ

loginrec

LOGINREC 構造体を指すポインタです。このポインタは、引数として dbopen に渡されます。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

戻り値

なし。

使用法

- 1 つのサーバ上で実行している Transact-SQL のコマンド・バッチ、またはストアド・プロシージャから、他のサーバ上にあるストアド・プロシージャを呼び出すことができます。このサーバ間の通信を実現するために、実際には、dbopen を介してアプリケーションに接続している最初のサーバが、リモート・サーバである 2 番目のサーバにログインします。

  dbrpwset を使用すると、最初のサーバがリモート・サーバ上のストアド・プロシージャを呼び出すときに使用するパスワードを指定できます。最初のサーバがログインする必要のあるサーバごとに 1 つずつ、複数のパスワードを指定することもできます。

- dbopen を連続して呼び出してそれぞれ異なるサーバに接続する場合に、1 つの LOGINREC を繰り返し使用できます。dbrpwclr を使用すると、現在 LOGINREC 内にあるリモート・パスワード情報を削除できます。これにより、dbopen を連続して呼び出すときに、そのつど異なるリモート・パスワード情報 (dbrpwset を使用して指定) を格納できます。

参照

dblogin、dbopen、dbrpwset、DBSETLAPP、DBSETLHOST、DBSETLPWD、DBSETLUSER

dbrpwset

説明

LOGINREC 構造体にリモート・パスワードを追加します。

構文

RETCODE dbrpwset(loginrec, srvname, password, pwlen)

```
LOGINREC  *loginrec;
char      *srvname;
char      *password;
int       pwlen;
```

パラメータ

loginrec

LOGINREC 構造体を指すポインタです。このポインタは、引数として dbopen に渡されます。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

srvname
サーバの名前です。サーバ名は、システム・テーブル sysservers の *srvname* カラム内に格納されています。最初のサーバは、*srvname* で指定されたサーバ上のストアド・プロシージャを呼び出すときに、指定されたパスワードを使用してログインします。*srvname* が NULL のときは、指定されたパスワードは「汎用」パスワードであるとみなされ、サーバのパスワードが明示的に指定されていない場合に使用されます。

password
最初のサーバが、指定されたサーバへログインするために使用するパスワードです。

pwlen
パスワードのバイト単位での長さです。

戻り値

SUCCEED または FAIL。

指定されたパスワードの追加によって LOGINREC のリモート・パスワード・バッファがオーバフローすると、このルーチンは失敗します (リモート・パスワード・バッファの大きさは 255 バイトです。バッファ内の各パスワードのエントリは、パスワードそのもの、対応するサーバ名、追加の 2 バイトで構成されています)。

使用法

- 1 つのサーバ上で実行している Transact-SQL のコマンド・バッチ、またはストアド・プロシージャから、他のサーバ上にあるストアド・プロシージャを呼び出すことができます。このサーバ間の通信を実現するために、実際には、dbopen を介してアプリケーションに接続している最初のサーバが、リモート・サーバである 2 番目のサーバにログインし、リモート・プロシージャ・コールを実行します。

  dbrpwset を使用すると、最初のサーバがリモート・サーバ上のストアド・プロシージャを呼び出すときに使用するパスワードを指定できます。最初のサーバがログインする必要のあるサーバごとに 1 つずつ、複数のパスワードを指定することもできます。
- アプリケーションが、特定のサーバのリモート・パスワードを指定していない場合、そのサーバのパスワードは、DBSETLPWD によって設定されるデフォルトのパスワードになります (DBSETLPWD が呼び出されていない場合は、NULL 値になります)。この動作は、アプリケーションのユーザが複数のサーバ上で同じパスワードを使用している場合に都合が良いことがあります。
- dbrpwclr は、LOGINREC からリモート・パスワードをすべてクリアします。

参照

dblogin、dbopen、dbrpwclr、DBSETLAPP、DBSETLHOST、DBSETLPWD、DBSETLUSER

dbsafestr

説明

文字列中の引用符を二重にします。

構文

```
RETCODE dbsafestr(dbproc, src, srclen, dest, destlen,
        quotetype)

DBPROCESS    *dbproc;
char         *src;
DBINT        srclen;
char         *dest;
DBINT        destlen;
int          quotetype;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

src

元の文字列を指すポインタです。

srclen

src のバイト単位の長さです。*srclen* が -1 の場合、*src* は NULL で終了するものとみなされます。

dest

結果文字列を格納するためにプログラマが用意したバッファを指すポインタです。*dest* は、結果文字列と NULL ターミネータを格納するのに十分な大きさにしてください。

destlen

結果文字列を格納するためにプログラマが用意したバッファの長さです。*destlen* が -1 の場合、*dest* は結果文字列を格納するのに十分な大きさであるとみなされます。

quotetype

二重にする引用符のタイプです。表 2-24 に、*quotetype* に対して可能な値をリストします。

表 2-24：quotetype の値

quotetype の値	dbsafestr
DBSINGLE	*src* の中のすべての一重引用符 (') を二重にします。
DBDOUBLE	*src* の中のすべての二重引用符 (") を二重にします。
DBBOTH	*src* の中のすべての一重引用符および二重引用符を二重にします。

戻り値

SUCCEED または FAIL。

dbsafestr は、結果文字列が *dest* に対して大きすぎる場合、または無効な *quotetype* が指定された場合に失敗します。

使用法

- dbsafestr は、文字列の中の一重引用符と二重引用符を二重にします。これは、文字列中にリテラルの引用符を指定する場合に便利です。

参照

dbcmd、dbfcmd

dbsechandle

説明

セキュア・ログインを処理するユーザ関数をインストールします。

構文

```
RETCODE *dbsechandle(type, handler)

DBINT        type;
INTFUNCPTR    (*handler)();
```

パラメータ

type

表 2-25 に示す記号値のいずれかを持つ整数変数です。

表 2-25：type の値 (dbsechandle)

type の値	dbsechandle
DBENCRYPT	パスワードの暗号化を処理する関数をインストールします。
DBLABELS	ログイン・セキュリティ・ラベルを処理する関数をインストールします。

handler

対応するタイプのセキュア・ログインを処理する必要が生じたときに DB-Library が呼び出すユーザ関数を指すポインタです。

handler が NULL で *type* が DBENCRYPT の場合、DB-Library はデフォルトの暗号化ハンドラを使用します。

handler が NULL で *type* が DBLABELS の場合、dbsechandle は現在のラベル・ハンドラのインストールを解除します。

戻り値

SUCCEED または FAIL。

使用法

- dbsechandle は、セキュア・ログインを処理するユーザ関数をインストールします。
- dbsechandle を使用すると、次の 2 つのタイプのセキュア・ログインを処理する関数をインストールできます。
  - 暗号化パスワード・セキュア・ログイン

    このタイプのセキュア・ログインでは、サーバがクライアントにキーを渡します。クライアントは、そのキーを使用して暗号化したパスワードをサーバに返します。
  - セキュリティ・ラベル・セキュア・ログイン

    このタイプのセキュア・ログインでは、サーバはクライアントの身元を表すセキュリティ・ラベルを要求し、クライアントはこのラベルを渡します。

暗号化パスワード・セキュア・ログイン

- *type* が DBENCRYPT である場合、dbsechandle は、ユーザ・パスワードの暗号化時に DB-Library が呼び出す関数をインストールします。
- DB-Library は、dbopen を呼び出す前に DBSETLENCRYPT が呼び出された場合にだけ、パスワードの暗号化を行います。
- ユーザ関数がインストールされていない場合、DB-Library はデフォルトの暗号化ハンドラを呼び出します。
- 一般に、パスワードを暗号化するためにユーザ関数をインストールする必要はありません。アプリケーションが Adaptive Server Enterprise に接続するとき、DB-Library のデフォルトの暗号化ハンドラによってパスワードを暗号化できるためです。
- ユーザ定義の暗号化ハンドラは、ゲートウェイとなるアプリケーションによってインストールされます。暗号化ハンドラには、リモート・サーバから返される暗号化キーを受け取り、そのキーをクライアントに渡し、クライアントから暗号化されたパスワードを読み込んで、暗号化されたパスワードを DB-Library に返すという役目があります。DB-Library は、この暗号化されたパスワードをリモート・サーバに渡します。
- 暗号化ハンドラは、次の例のように定義してください。Windows プラットフォームの暗号化ハンドラは CS_PUBLIC で宣言してください。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。次に宣言の例を示します。

```
RETCODE CS_PUBLIC encryption_handler(dbproc, pwd,
    pwdlen, enc_key, keylen, outbuf, buflen, outlen)
 DBPROCESS    *dbproc;
 BYTE         *pwd;
 DBINT        pwdlen;
 BYTE         *enc_key;
 DBINT        keylen;
 BYTE         *outbuf;
 DBINT        buflen;
 DBINT        *outlen;
```

各パラメータの意味は次のとおりです。

- *dbproc* は DBPROCESS です。
- *pwd* は、暗号化されるユーザ・パスワードです。
- *pwdlen* は、ユーザ・パスワードの長さです。
- *enc_key* は、暗号化に使われるキーです。
- *keylen* は、暗号化キーの長さです。
- *outbuf* は、暗号化されたパスワードがコールバックによって格納されるバッファです。このバッファの割り付けと解放は、DB-Library が行います。
- *buflen* は、出力バッファの長さです。
- *outlen* は、DBINT を指すポインタです。暗号化ハンドラは、**outlen* を暗号化されたパスワードの長さに設定します。

- 暗号化ハンドラは、パスワードが正常に暗号化されたことを示すために SUCCEED を返します。暗号化ハンドラが SUCCEED 以外の値を返すと、DB-Library は接続をアボートします。

セキュリティ・ラベル・セキュア・ログイン

- type が DBLABELS の場合、dbsechandle は DB-Library がログイン・セキュリティ・ラベルを取得するために呼び出す関数をインストールします。
- DB-Library は、dbopen を呼び出す前に DBSETLABELLED が呼び出された場合にだけ、ログイン・セキュリティ・ラベルを送信します。
- アプリケーションがセキュリティ・ラベルを定義するには、次の 2 通りの方法があります。
  - 定義するラベルごとに一度ずつ dbsetsecurity を呼び出す。ほとんどのアプリケーションはこの方法を使用する。

- dbsechandle を呼び出して、セキュリティ・ラベルを生成するためのユーザ提供関数をインストールする。通常は、ゲートウェイ・アプリケーションだけがこの方法を使用する。

1 つのアプリケーションでこの両方の方法を使用する場合は、dbsetsecurity で定義されたラベルとユーザ提供関数が生成したラベルは同時にサーバに送信されます。

- DB-Library は、サーバからのログイン・セキュリティ・ラベルが要求されたときに、接続処理中にアプリケーションのラベル・ハンドラを呼び出します。ラベル・ハンドラは、呼び出されるたびに 1 つのラベルを返します。このラベルは、それまでに dbsetsecurity で定義されたラベルとともにサーバに送信されます。
- DB-Library のデフォルトのラベル・ハンドラはありません。
- ユーザ定義のラベル・ハンドラは、ゲートウェイとなるアプリケーションによってインストールしてください。ラベル・ハンドラには、クライアントのログイン・セキュリティ・ラベルを読み込んで、そのラベルを DB-Library へ渡す役目があります。DB-Library は、このラベルをリモート・サーバに送信します。
- ラベル・ハンドラは、次の例のように定義してください。Windows プラットフォームのラベル・ハンドラは、CS_PUBLIC で宣言してください。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。次に宣言の例を示します。

```
RETCODE CS_PUBLIC label_handler(dbproc, namebuf,
nbuflen, valuebuf, vbuflen, namelen, valuelen)
 DBPROCESS    *dbproc;
 DBCHAR       *namebuf;
 DBINT        nbuflen;
 DBCHAR       *valuebuf;
 DBINT        vbuflen;
 DBINT        *namelen;
 DBINT        *valuelen;
```

各パラメータの意味は次のとおりです。

- *dbproc* は DBPROCESS です。
- *namebuf* は、ハンドラによってログイン・セキュリティ・ラベルの名前が格納されるバッファです。このバッファの割り付けと解放は DB-Library が行います。
- *nbuflen* は、*namebuf* バッファの長さです。

- *valuebuf* は、ハンドラによってログイン・セキュリティ・ラベルの値が格納されるバッファです。このバッファの割り付けと解放は DB-Library が行います。
- *vbuflen* は、*valuebuf* バッファの長さです。
- *namelen* は、DBINT を指すポインタです。ラベル・ハンドラは、**namelen* を、*namebuf* に格納されたラベルの名前の長さに設定します。
- *valuelen* は、DBINT を指すポインタです。ラベル・ハンドラは、**valuelen* を、*valuebuf* に格納されたラベルの値の長さに設定します。

- 表 2-26 は、セキュリティ・ラベル・ハンドラの有効な戻り値のリストです。セキュリティ・ラベル・ハンドラが返す値は、次のいずれかにしてください。

表 2-26：セキュリティ・ラベル・ハンドラの戻り値

ラベル・ハンドラ戻り値	意味
DBMORELABEL	ラベル・ハンドラがログイン・セキュリティ・ラベルの名前と値を設定した。 DB-Library は、ラベル・ハンドラを再度呼び出して追加のラベルを取得する。
DBENDLABEL	ラベル・ハンドラがログイン・セキュリティ・ラベルの名前と値を設定した。 DB-Library は、ラベル・ハンドラをもう一度呼び出してはならない。
DBERRLABEL	ラベル・ハンドラのエラーが発生した。DB-Library は、接続をアボートする。

参照　DBSETLENCRYPT、dbopen

dbsendpassthru

説明　サーバに TDS パケットを送信します。

構文　RETCODE dbsendpassthru(dbproc, send_bufp)

```
DBPROCESS    *dbproc;
DBVOIDPTR    send_bufp;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

send_bufp
サーバに送信される TDS パケットを含むバッファを指すポインタです。1 パケットのデフォルト・サイズは、512 バイトです。このサイズを変更するには、DBSETLPACKET を使用します。

戻り値

DB_PASSTHRU_MORE、DB_PASSTHRU_EOM、または FAIL。

使用法

- dbsendpassthru は、サーバに TDS (Tabular Data Stream) パケットを送信します。
- TDS は、クライアントとサーバの間で、要求と要求結果を転送するために使用されるアプリケーション・プロトコルです。通常は、DB-Library/C がデータ・ストリームを管理しているため、DB-Library/C アプリケーションが直接 TDS を取り扱う必要はありません。
- dbrecvpassthru および dbsendpassthru は、ゲートウェイ・アプリケーションに役立ちます。2 つのサーバを仲介する役割を持つアプリケーションでこれらのルーチンを使用すると、情報を解釈して再エンコードすることなく 1 つのサーバから他のサーバへ TDS ストリームを転送できます。
- dbsendpassthru は、*send_bufp* が指すバッファから 1 パケット分のバイトを送信します。一般的には、*send_bufp* には dbrecvpassthru から返された **recv_bufp* を指定します。*send_bufp* に、送信するパケットが格納されている、ユーザ割り付けのバッファのアドレスを指定することもあります。
- 1 パケットのデフォルト・サイズは、512 バイトです。アプリケーションでこのパケット・サイズを変更するには、DBSETLPACKET を使用します。詳細については、dbgetpacket および DBSETLPACKET のリファレンス・ページを参照してください。
- バッファ内の TDS パケットに EOM (End Of Message) マークが付いている場合は、dbsendpassthru は DB_PASSTHRU_EOM を返します。TDS パケットがストリームの最後でない場合は、dbsendpassthru は DB_PASSTHRU_MORE を返します。
- dbsendpassthru オペレーションで使用される DBPROCESS 接続は、DB_PASSTHRU_EOM が受信されるまでは、他の DB-Library/C 関数で使用することはできません。

- 次のコードは、dbsendpassthru の例です。

```
/*
** The following code fragment illustrates the
** use of dbsendpassthru() in an Open Server
** gateway application. It will continually get
** packets from a client, and pass them through
** to the remote server.
**
** The routine srv_recvpassthru() is the Open
** Server counterpart required to complete this
** passthru operation.
*/

DBPROCESS     *dbproc;
SRV_PROC      *srvproc;
int           ret;
BYTE          *packet;
while (1)
 {
      ret = srv_recvpassthru(srvproc, &packet,
           (int *)NULL);

     if( ret == SRV_S_PASSTHRU_FAIL )
      {
           fprintf(stderr, "ERROR - ¥
                srv_recvpassthru failed in ¥
                lang_execute.¥n");
           exit();
      }
     /*
      ** Now send the packet to the remote server
      */
      if( dbsendpassthru(dbproc, packet) == FAIL )
      {
          fprintf(stderr, "ERROR - dbsendpassthru¥
                failed in lang_execute.¥n");
           exit();
      }
     /*
      ** We've sent the packet, so let's see if
      ** there's any more.
      */
      if( ret == SRV_S_PASSTHRU_MORE )
           continue;
      else
           break;
 }
```

参照 dbrecvpassthru

dbservcharset

説明 サーバの文字セット名を取得します。

構文 char *dbservcharset(dbproc)

DBPROCESS *dbproc;

パラメータ dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library/C がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 NULL で終了するサーバ文字セット名を指すポインタです。エラーの場合は NULL です。

使用法

- dbservcharset は、サーバの文字セット名を返します。
- DB-Library/C クライアントは、接続しているサーバとは異なる文字セットを使用できます。クライアントとサーバが異なる文字セットを使用している場合に、クライアントの文字セットに対する文字変換をサーバがサポートしていれば、サーバはクライアントと通信するときにサーバ文字セットとの間の変換を行います。
- アプリケーションが使用している文字セットをサーバに通知するには、DBSETLCHARSET を使用します。
- サーバが文字セット変換を実行するかどうかをアプリケーションで調べるには、dbcharsetconv を呼び出します。
- クライアントの文字セット名をアプリケーションで取得するには、dbgetcharset を呼び出します。

参照 dbcharsetconv、dbgetcharset、DBSETLCHARSET

dbsetavail

説明　DBPROCESS に通常の使用可能というマーク (being available) を付ける。

構文

```
void dbsetavail(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　なし。

使用法　このルーチンは、DBPROCESS に通常の使用可能マークを付けます。その後で DBISAVAIL を呼び出すと、この DBPROCESS がまだ使用されていなければ、「TRUE」が返されます。多くの DB-Library ルーチンは、自動的に DBPROCESS を「使用不可能 (not available)」に設定します。この機能は、1 つのプログラム内の各部分で 1 つの DBPROCESS を共用する場合に役立ちます。

参照　DBISAVAIL

dbsetbusy

説明　DB-Library がサーバから読み込みを行っているときに、ユーザ提供の関数を呼び出します。

構文

```
void dbsetbusy(dbproc, busyfunc)

DBPROCESS    *dbproc;
int          *(*busyfunc())();
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

busyfunc

DB-Library がサーバにアクセスするときに呼び出すユーザ提供の関数です。DB-Library が *busyfunc*() を呼び出すときのパラメータは 1 つだけです。このパラメータとは、dbsetbusy 呼び出しからの DBPROCESS を指すポインタです。

busyfunc() は、整数を返す関数を指すポインタを返します。

戻り値

なし。

使用法

- このルーチンは、ユーザ提供の関数を指定の *dbproc* に関連付けます。ユーザ提供関数は、DB-Library がサーバからの出力の読み込みを行っているとき、または読み込みを待っているときに自動的に呼び出されます。たとえば、あるアプリケーションでは、サーバにアクセスしたときに必ずメッセージを出力するとします。この場合、dbsetbusy によって、ユーザ提供の関数 *busyfunc*() が呼び出されます。
- 同様に、dbsetidle も、ユーザ提供の関数 *idlefunc*() を *dbproc* に関連付けるために使用できます。*idlefunc*() は、DB-Library がサーバからの出力の読み込みを完了したときに自動的に呼び出されます。
- サーバからの結果データは、512 バイトのパケットの形でアプリケーションに送られます (結果セットの最後のパケットは 512 バイト未満の場合もあります)。DB-Library は、各パケットの開始時に *busyfunc*() を呼び出し、各パケットの終了時に *idlefunc*() を呼び出します。サーバからの出力が複数のパケットにわたる場合は、*busyfunc*() と *idlefunc*() は複数回呼び出されます。
- 次に、*busyfunc*() と *idlefunc*() の定義とインストールの例を示します。

注意 アプリケーション関数 *busyfunc*() および *idlefunc*() は、コールバック・イベント・ハンドラであり、Windows プラットフォームでは CS_PUBLIC として宣言する必要があります。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。

```
/*
** busyfunc returns a pointer to a function that
** returns an integer.
*/
int      (*busyfunc())();
void     idlefunc();

int      counterfunc();
...
```

```
main()
 {
     DBPROCESS     *dbproc;
     ...

   dbproc = dbopen(login, NULL);

    /*
     ** Now that we have a DBPROCESS, install the
     ** busy-function and the idle-function.
     */
     dbsetbusy(dbproc, busyfunc);
     dbsetidle(dbproc, idlefunc);

   dbcmd(dbproc, "select * from sysdatabases");
     dbcmd(dbproc, " select * from sysobjects");
     dbsqlexec(dbproc);

   /*
     ** DB-Library calls busyfunc() for the first time
    ** during dbsqlexec(). Depending on the size of the
     ** results, it may call busyfunc() again during
     ** processing of the results.
     */

   while (dbresults(dbproc) != NO_MORE_RESULTS)
        dbprrow(dbproc);

   /*
     ** DB-Library calls idlefunc() each time a packet
     ** of results has been received. Depending on the
     ** size of the results, it may call idlefunc()
    ** multiple times during processing of the results.
     */
     ...
}

int CS_PUBLIC (*busyfunc(dbproc))()
  DBPROCESS  dbproc;
  {
       printf("Waiting for data...¥n");

    return(counterfunc);
  }

void CS_PUBLIC idlefunc(procptr, dbproc)

/*
  ** idlefunc's first parameter is a pointer to a
  ** routine that returns an integer. This is the same
  ** pointer that busyfunc returns.
```

```
  */
  int           (*procptr)();

DBPROCESS     *dbproc;
  {
      int       count;

    printf("Data is ready.¥n");
      count = (*procptr)();

    printf ("Counterfunc has been called %d %s.¥n",
      count, (count == 1 ?"time" :"times"));

}

 int counterfunc()
  {
       static     int     counter = 0;

     return(++counter);
  }
```

参照 dbsetidle

dbsetconnect

説明 ディレクトリ・サービスの代わりに使用するサーバ接続情報を指定します。

構文 RETCODE dbsetconnect(service_type, net_type, net_name, machine_name, port)

```
char      *service_type;
char      *net_type;
char      *net_name;
char      *machine_name;
char      *port;
```

パラメータ service_type

接続のタイプ。デフォルト値は次のとおりです。

- 「master」は master 行を指定します。これはサーバ・アプリケーションがクライアント・クエリを受信するときに使用します。
- 「query」は query 行を指定します。これはクライアント・アプリケーションがサーバを探すときに使用します。

net_type
ネットワーク・プロトコル名。有効な値：

- TCP/IP の場合は「tcp」－ すべての UNIX プラットフォーム
- DECnet の場合は「decnet」

net_name
ネットワークの記述子。Open Client/Open Server は、現時点では net_name を使用していません。これはプレースホルダであり、Sybase は今後この情報を定義します。TCP/IP ネットワークでは、*net_name* は「ether」に設定されます。

machine_name
サーバが稼働しているノードやマシンのネットワーク名です。*machine_name* に指定できる最大文字数はエントリで指定されるプロトコルによって異なります。

- TCP/IP での最大文字数は 32 文字です。
- DECnet での最大文字数は 6 文字です。

UNIX プラットフォームで /bin/hostname コマンドを使用して、ログインするマシンのネットワーク名を調べます。

port
クエリを受け取るためにサーバが使用するポートです。TCP/IP と DECnet プロトコルはそれぞれ次のようにこの要素を指定します。

- TCP/IP：有効なポート番号の範囲は 1024 から 49151 までです。この範囲内にあるポート番号を使用することをおすすめします。
- DECnet：有効なオブジェクト番号は 128 から 253 までの範囲です。オブジェクト名も有効です。

netstat コマンドを使用して、どのポート番号が使用されているかを確認してください。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンを使用すると、アプリケーションはサーバに接続するために必要となるサービス・タイプ、ネットワーク・プロトコル・タイプ、サーバのネットワーク名、サーバ名、ポート番号などの接続情報を指定できます。この接続情報は、dbopen への今後の呼び出しすべてにおいて使用されます。
- dbsetconnect が使用される場合は、サーバ・エントリ用の DSQUERY と通常のディレクトリ・サービス・ルックアップは迂回されます。

- dbsetconnect が呼び出されなかった場合、接続情報はディレクトリ・サービスを使用することで見つけることができます。デフォルト・ディレクトリ・サービスは、UNIX の場合は *interfaces* ファイル、Windows の場合は *sql.ini* ファイルです。その他のディレクトリ・サービスは、構成ファイル *libtcl.cfg* によって設定できます。
- 『Open Client/Server 設定ガイド』を参照してください

参照 dbopen

dbsetdefcharset

説明 アプリケーションのデフォルト文字セットを設定します。

構文

```
RETCODE dbsetdefcharset(charset)

char        *charset;
```

パラメータ

charset
使用する文字セットの名前です。*charset* は null で終了する文字列でなければなりません。

戻り値 SUCCEED または FAIL。

使用法

- dbsetdefcharset は、アプリケーションのデフォルト文字セットを設定します。
- デフォルト文字セットは、使用可能な DBPROCESS 構造体がない場合、または DBPROCESS 構造体の文字セットに対するローカライゼーション情報が見つからない場合に使用されます。
- アプリケーションが dbsetdefcharset を呼び出さない場合のデフォルト文字セットは、最初にオープンされた DBPROCESS 接続の文字セットです。DBPROCESS がまったくオープンされていない場合は、iso_1 となります。
- アプリケーションで dbsetdefcharset と dbsetdeflang の両方を呼び出す場合は、dbsetdefcharset を先に呼び出す必要があります。

参照 dbsetdeflang、dbsetdefcharset、dblogin、dbopen

dbsetdeflang

説明　アプリケーションのデフォルト言語名を設定します。

構文

```
RETCODE dbsetdeflang(language)

char      *language;
```

パラメータ

language
　使用する言語の名前です。*language* は、null で終了する文字列でなければなりません。

戻り値　SUCCEED または FAIL。

使用法

- dbsetdeflang は、アプリケーションのデフォルトの各国言語を設定します。
- デフォルト言語は、使用可能な DBPROCESS 構造体がない場合、または DBPROCESS 構造体の言語に対するローカライゼーション情報が見つからない場合に使用されます。
- アプリケーションが dbsetdeflang を呼び出さない場合のデフォルト言語は、最初にオープンした DBPROCESS 接続の言語です。DBPROCESS がまったくオープンされていない場合は us_english です。

参照　DBSETLNATLANG

dbsetidle

説明　DB-Library がサーバからの読み込みを終了したときに、ユーザ提供関数を呼び出します。

構文

```
void dbsetidle(dbproc, idlefunc)

DBPROCESS     *dbproc;
void          (*idlefunc)();
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

idlefunc
サーバがホストへのデータ送信を終了したときに DB-Library が呼び出す、ユーザ提供の関数です。DB-Library は、2 つのパラメータを指定して *idlefunc*() を呼び出します。このパラメータは、*busyfunc*() (整数を返す関数を指すポインタ) からの戻り値と、dbsetidle 呼び出しからの DBPROCESS を指すポインタです。

idlefunc() は、void を返します。

戻り値

なし。

使用法

- このルーチンは、ユーザ提供の関数を指定の *dbproc* に関連付けます。ユーザ提供関数は、DB-Library がサーバからの 1 パケット分の出力の読み込みを終了したとき、または、読み込みを待っているときに自動的に呼び出されます。たとえば、あるアプリケーションでは、サーバがホストへのデータ送信を終了したときにメッセージを出力するとします。この場合、dbsetidle によって、ユーザ提供の関数 *idlefunc*() が呼び出されます。
- 同様に、dbsetbusy も、ユーザ提供の関数 *busyfunc*() を *dbproc* に関連付けるために使用できます。*busyfunc*() は、DB-Library がサーバからの出力パケットの読み込みを行っているとき、または読み込みを待っているときに自動的に呼び出されます。
- サーバからの結果データは、512 バイトのパケットの形でアプリケーションに送られます (結果セットの最後のパケットは 512 バイト未満の場合もあります)。DB-Library は、各パケットの開始時に *busyfunc*() を呼び出し、各パケットの終了時に *idlefunc*() を呼び出します。サーバからの出力が複数のパケットにわたる場合は、*busyfunc*() と *idlefunc*() は複数回呼び出されます。
- *busyfunc*() と *idlefunc*() の定義とインストールの例については、dbsetbusy のリファレンス・ページを参照してください。

参照

dbsetbusy

dbsetifile

説明

Sybase interfaces ファイルの名前とロケーションを指定します。

構文

```
void dbsetifile(filename)

char        *filename;
```

パラメータ

filename

後続のすべての dbopen の呼び出しで検索される interfaces ファイルの名前です。このパラメータが NULL の場合、DB-Library はデフォルトのファイル名を使用します。

戻り値

なし。

使用法

- アプリケーションでこのルーチンを使用することにより、後続のすべての dbopen の呼び出しで検索する interfaces ファイルの名前とロケーションを指定できます。interfaces ファイルには、ネットワーク上で使用可能なすべてのサーバの名前とネットワーク・アドレスが記録されています。
- dbsetifile が呼び出されていない場合は、dbopen を呼び出すとデフォルトの動作が実行されます。DB-Library は、SYBASE 環境変数または論理名で指定されたディレクトリにある *interfaces* という名前のファイルを使用します。SYBASE 環境変数が設定されていない場合、DB-Library は、「sybase」というユーザのホーム・ディレクトリにある *interfaces* という名前のファイルを使用します。
- 『Open Client/Server 設定ガイド』を参照してください)。

注意　UNIX 以外のプラットフォームでは、クライアント・アプリケーションがサーバのアドレス情報を検索する方法は UNIX の *interfaces* ファイルのときとは異なります。クライアントがサーバに接続する方法については、『Open Client/Server 設定ガイド』を参照してください。

参照

dbopen

dbsetinterrupt

説明

サーバからの読み込みを待っている間、割り込みを処理するために、ユーザが提供する関数を呼び出す。

構文

void dbsetinterrupt(dbproc, chkintr, hndlintr)

```
DBPROCESS    *dbproc;
int          (*chkintr)();
int          (*hndlintr)();
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

chkintr

保留中の割り込みがあるかどうかをチェックするために DB-Library が呼び出すユーザ関数を指すポインタです。DB-Library は、サーバからの読み込みを待っている間、定期的にこの関数を呼び出します。DB-Library が chkintr() を呼び出すときのパラメータは 1 つだけです。このパラメータとは、dbsetinterrupt 呼び出しからの DBPROCESS を指すポインタです。

chkintr() は、「TRUE」または「FALSE」を返します。

hndlintr

割り込みが返された場合に DB-Library が呼び出すユーザ関数を指すポインタです。DB-Library が hndlintr() を呼び出すときのパラメータは 1 つだけです。このパラメータとは、dbsetinterrupt 呼び出しからの DBPROCESS を指すポインタです。

表 2-27 lists は、hndlintr の有効な戻り値です。

表 2-27：hndlintr() 関数の戻り値

戻り値	意味
INT_EXIT	プログラムをアボートする (UNIX プログラマの方へ：DB-Library はコア・ファイルを残しません)。
INT_CANCEL	現在のコマンド・バッチをアボートする。結果は DBPROCESS 接続からフラッシュされない。
INT_CONTINUE	サーバの応答を待ち続ける。

戻り値

なし。

使用法

- DB-Library は、サーバからの非ブロッキング読み込みを行います。サーバからの読み込みを待っている間、DB-Library は chkintr() 関数を呼び出して、保留中の割り込みがあるかどうかを調べます。chkintr() が「TRUE」を返したとき、dbsetinterrupt の hndlintr() としてインストールされているハンドラがある場合は、hndlintr() が呼び出されます。dbsetinterrupt は、ホスト・プログラムがサーバからの読み込みを待っている間、代替の割り込み処理を行うための関数です。
- hndlintr() からの戻り値に応じて、DB-Library は次の動作のいずれかを行います。

- サーバにアテンションを送信して、サーバの処理を中止する (INT_CANCEL)。詳細については、「割り込みハンドラからのキャンセル」(337 ページ) を参照。
- サーバからの読み込みを継続する (INT_CONTINUE)。
- プログラムを終了する (INT_EXIT)。

割り込みハンドラからのキャンセル

- hndlintr() から INT_CANCEL が返されると、DB-Library はサーバにアテンション・トークンを送信します。この結果、サーバはコマンド処理を中止します。サーバからは、すでに計算が完了した分の結果が送信される場合があります。メインライン・コードに制御が戻ったときに、メインライン・コードで次の処理のいずれかを実行する必要があります。
  - dbcancel を使用して結果をフラッシュする
  - 結果を通常どおりに処理する
- 割り込みハンドラ内で dbcancel を呼び出すことはできません。これは、サーバから DB-Library への出力の同期が取れなくなるためです。割り込みハンドラからのキャンセルの正しい方法は次のとおりです。
  - *int_canceled* フラグを DBPROCESS 構造に関連付ける。DBPROCESS 内のフラグを指すポインタをインストールするには dbsetuserdata を使用し、フラグのアドレスを取得するには dbgetuserdata を使用する。
  - INT_CANCEL を返すかどうかを示す *int_canceled* フラグを設定するように hndlintr() をコーディングする。
  - メインライン・コード内で、dbresults または dbnextrow に対する各呼び出しの前にフラグを検査する。*hndlintr*() がサーバ・コマンドをアボートしたことを int_canceled フラグが示している場合は、dbcancel を呼び出し、フラグをクリアする。

例

- 次に、chkintr() ルーチンと *hndlintr*() ルーチンの例を示します。

注意 アプリケーション *chkintr*() ルーチンおよび hndlintr() ルーチンはコールバック関数であり、Windows プラットフォームでは CS_PUBLIC として宣言する必要があります。移植性を維持するために、他のプラットフォームでも同様にコールバック・ハンドラを CS_PUBLIC で宣言してください。

```
int CS_PUBLIC  chkintr(dbproc)
DBPROCESS     *dbproc;
 {
   /*
   ** This routine assumes that the application
   ** sets the global variable
   ** "OS_interrupt_happened" upon catching
   ** an interrupt using some operating system
   ** facility.
   */
   if (OS_interrupt_happened)
   {
     /*
     ** Clear the interrupt flag, for
     ** future use.
     */
     OS_interrupt_happened = FALSE;
     return(TRUE);
   }
   else
     return(FALSE);
 }


int CS_PUBLIC  hndlintr(dbproc)
 DBPROCESS     *dbproc;
 {
   char    response[10];
   DBBOOL *int_canceled;
   /*
   ** We assume that a DBBOOL flag has been
   ** attached to dbproc with dbsetuserdata.
```

```
        */
        int_canceled = (DBBOOL *) dbgetuserdata(dbproc);
        if (int_canceled == (DBBOOL *)NULL)
        {
          printf("Fatal Error:no int_cancel flag ¥
                in the DBPROCESS¥n");
          return (INT_EXIT);
        }
        *int_canceled = FALSE;
        printf("¥nAn interrupt has occurred. Do you ¥
              want to:¥n¥n");
        printf("¥t1) Abort the program¥n");
        printf("¥t2) Cancel the current query¥n");
        printf("¥t3) Continue processing the current¥
              query's results¥n¥n");
        printf("Press 1, 2, or 3, followed by the ¥
              return key:");
        gets(response);
        switch(response[0])
        {
          case '1':
            return (INT_EXIT);
            break;
          case '2':
            *int_canceled = TRUE;
            return(INT_CANCEL);
            break;
          case '3':
            return(INT_CONTINUE);
            break;
          default:
            printf("Response not understood.¥
                  Aborting program.¥n");
            return (INT_EXIT);
            break;
        }
      }
```

参照　dbcancel、dbgetuserdata、dbsetuserdata、dbsetbusy、dbsetidle

DBSETLAPP

説明

LOGINREC 構造体内のアプリケーション名を設定する。

構文

RETCODE DBSETLAPP(loginrec, application)

```
LOGINREC        *loginrec;
char            *application;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

application

サーバへ送られるアプリケーション名です。これは NULL で終了する文字列でなくてはなりません。文字列の最大長は、NULL ターミネータを除いて 30 文字です。

戻り値

SUCCEED または FAIL。

使用法

- このマクロは、LOGINREC 構造体のアプリケーション・フィールドを設定します。これを有効にするには、dbopen の前に呼び出してください。
- 必ずしもこのルーチンを呼び出す必要はありません。デフォルトでは、アプリケーション名は NULL 値になります。
- アプリケーション名は、サーバの sysprocesses テーブル内でユーザのプロセスを識別しやすくするために使用されます。アプリケーション名を設定すると、master データベース内の sysprocesses テーブルに対する問い合わせを実行したときにこの名前が表示されます。

参照

dblogin、dbopen、DBSETLHOST、DBSETLPWD、DBSETLUSER

DBSETLCHARSET

説明

LOGINREC 構造体内の文字セットを設定します。

構文

RETCODE DBSETLCHARSET(loginrec, char_set)

```
LOGINREC   *loginrec;
DBCHAR     *char_set;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を取得するには、dblogin を使用します。

char_set

クライアントが使用する文字セットの名前です。*char_set* は NULL で終了する文字列でなくてはなりません。*char_set* のデフォルト値には、ISO-8859-1 (ほとんどのプラットフォーム) を表す「iso_1」、Code Page 850 (IBM RS/6000) を表す「cp850」、Roman8 文字セット (HP プラットフォーム) を表す「roman8」などがあります。

文字セット変換が必要ないことを示すには、*char_set* を NULL として渡してください。

戻り値

SUCCEED または FAIL。

使用法

- DBSETLCHARSET は、LOGINREC 構造体内のクライアント文字セットを設定します。
- DB-Library/C クライアントは、接続しているサーバとは異なる文字セットを使用できます。DBSETLCHARSET は、クライアントが使用している文字セットをサーバに知らせるために使用されます。
- LOGINREC は、クライアントとサーバとの接続を確立する dbopen 呼び出しのパラメータとして渡されるため、DBSETLCHARSET を有効にするには、dbopen の前に呼び出す必要があります。
- クライアントが使用する文字セットがサーバの文字セットと異なる場合は、クライアントとの通信時の文字セットの変換はすべてサーバが行います。
- 変換が不要な場合は、*char_set* を NULL として DBSETLCHARSET を呼び出してください。

参照

dbgetcharset、dblogin、dbopen

DBSETLENCRYPT

説明

Adaptive Server Enterprise にログインするときにネットワーク・パスワードの暗号化を使用するかどうかを指定します。

構文

```
RETCODE DBSETLENCRYPT(loginrec, enable)

LOGINREC    *loginrec;
DBBOOL      enable;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

enable

ログイン時に暗号化されたパスワードをサーバが要求するかどうかを指定するブール値 (「true」または「false」) です。

戻り値

SUCCEED または FAIL。

使用法

- DBSETLENCRYPT は Adaptive Server Enterprise にログインするときにネットワーク・パスワードの暗号化を使用するかどうかを指定する。アプリケーションが DBSETLENCRYPT を呼び出さない場合は、パスワードの暗号化は使用されません。
- ネットワーク・パスワードの暗号化は、ユーザの ID の認証を保護するメカニズムです。
- ネットワーク・パスワードの暗号化を使用することをアプリケーションが指定した場合は、アプリケーションが接続をオープンしようとすると、次の処理が行われます。
  - 最初の接続要求では、パスワードは送信されない。この時点では、クライアントは、サーバに対して暗号化が必要であることを示す。
  - サーバは、接続要求への応答として暗号化キーを返す。
  - DB-Library は、このキーを使用してユーザ・パスワードとリモート・パスワード (ある場合) を暗号化し、暗号化したパスワードをサーバに返す。
  - サーバは、暗号化されたパスワードを、キーを使用して復号化し、ログインを受け入れるか、または拒否する。
- パスワードの暗号化が指定されていない場合は、アプリケーションが接続をオープンしようとすると、次の処理が行われます。
  - パスワードは接続要求とともに送信される。
  - サーバは、ログインを受け入れるか、または拒否する。

参照

dbsechandle

DBSETLHOST

説明　LOGINREC 構造体内のホスト名を設定します。

構文　RETCODE DBSETLHOST(loginrec, hostname)

```
LOGINREC        *loginrec;
char            *hostname;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

hostname

サーバへ送られるホスト名です。これは NULL で終了する文字列でなくてはなりません。文字列の最大長は、NULL ターミネータを除いて 30 文字です。

戻り値　SUCCEED または FAIL。

使用法

- このマクロは、LOGINREC 構造体内のホスト名を設定します。これを有効にするには、dbopen の前に呼び出してください。
- ホスト名は、master データベース内の sysprocesses テーブルで使用されます。
- 必ずしもこのルーチンを呼び出す必要はありません。呼び出さない場合は、DB-Library によってホスト名にはデフォルト値が設定されます。このデフォルト値は、一般に、オペレーティング・システムによって設定されるホスト・マシン名です。

参照　dblogin、dbopen、DBSETLAPP、DBSETLPWD、DBSETLUSER

DBSETLMUTUALAUTH

説明　接続のセキュリティ・メカニズムの相互認証を有効または無効にする。

構文　RETCODE DBSETLMUTUALAUTH(loginrec, enable)

```
LOGINREC        *loginrec;
DBBOOL          *enable;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

enable
サーバが相互認証を有効にするかどうかを指定するブール値 (「true」または「false」) です。

戻り値 SUCCEED または FAIL。

使用法

- DBSETLMUTUALAUTH を有効にするには、dbopen() の前に呼び出し、DBSETLNETWORKAUTH を有効にする必要があります。
- DBSETLMUTUALAUTH を呼び出さない場合、相互認証はデフォルトで無効です。

参照 dblogin、DBSETLNETWORKAUTH、DBSETLSERVERPRINCIPAL

DBSETLNATLANG

説明 LOGINREC 構造体内の言語名を設定します。

構文 RETCODE DBSETLNATLANG(loginrec, language)

```
LOGINREC        *loginrec;
char            *language;
```

パラメータ

loginrec
dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を取得するには、dblogin を使用します。

language
使用する言語の名前です。*language* は、null で終了する文字列でなければなりません。

戻り値 SUCCEED または FAIL。

使用法

- このマクロは、LOGINREC 構造体内のユーザ言語を設定します。特定のユーザ言語を設定する場合は、dbopen の前に DBSETLNATLANG を呼び出してください。
- DBSETLNATLANG は、サーバのデフォルトの各国言語を使用しない場合にだけ呼び出してください。

参照 dblogin、dbopen、dbsetdeflang

DBSETLNETWORKAUTH

説明　ネットワーク・ベースの認証を有効または無効にする。

構文　RETCODE DBSETLNETWORKAUTH(loginrec, enable)

```
LOGINREC        *loginrec;
DBBOOL          *enable;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

enable

サーバがネットワーク認証を有効にするかどうかを指定するブール値 (「true」または「false」) です。

戻り値　SUCCEED または FAIL。

使用法　DBSETLNETWORKAUTH を呼び出さない場合、ネットワーク認証はデフォルトで無効です。

参照　dblogin、DBSETLMUTUALAUTH、DBSETLSERVERPRINCIPAL

dbsetloginfo

説明　TDS ログイン情報を DBLOGINFO 構造体から LOGINREC 構造体へ転送する。

構文　RETCODE dbsetloginfo(loginrec, loginfo)

```
LOGINREC        *login;
DBLOGINFO       *loginfo;
```

パラメータ

login

LOGINREC 構造体を指すポインタです。このポインタは、引数として dbopen へ渡されます。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

loginfo

ログイン・パラメータ情報が含まれている DBLOGINFO 構造体を指すポインタです。

戻り値　SUCCEED または FAIL。

使用法

- dbsetloginfo は TDS ログイン情報を DBLOGINFO 構造体から LOGINREC 構造体に転送します。情報が転送されると、dbsetloginfo は DBLOGINFO 構造体を解放します。
- アプリケーションは、1) Open Server のゲートウェイ・アプリケーションであり、2) TDS パススルーを使用している場合にだけ、dbsetloginfo を呼び出す必要があります。
- TDS (Tabular Data Stream) は、クライアントとサーバ間の要求と要求結果の転送に使用されるアプリケーション・プロトコルです。
- クライアントがサーバと直接接続している場合、2 つのプログラムは、データの送受信に使用する TDS のフォーマットをネゴシエートします。ゲートウェイ・アプリケーションが TDS パススルーを使用する場合は、TDS パケットは検査も処理も行われず、そのままクライアントとリモート・サーバの間で転送されます。このため、リモート・サーバとクライアントは、使用する TDS フォーマットについて合意しておく必要があります。
- dbsetloginfo は、クライアントとリモート・サーバが TDS フォーマットをネゴシエートするための 4 つの呼び出しの 2 番目の呼び出しです。この 4 つのうち 2 つは Server Library 呼び出しです。SRV_CONNECT イベント・ハンドラの中でだけ実行できるこれら 4 つの呼び出しは、次のとおりです。
  - srv_getloginfo － DBLOGINFO 構造体を割り付け、この構造体にクライアント SRV_PROC からの TDS 情報を格納します。
  - dbsetloginfo － srv_getloginfo によって取得した TDS 情報を、DBLOGINFO 構造体から DB-Library/C の LOGINREC 構造体に転送してから、DBLOGINFO 構造体を解放します。情報が転送されると、アプリケーションで dbopen を呼び出してリモート・サーバとの接続を確立するときに、この LOGINREC 構造体を使用できます。
  - dbgetloginfo － クライアントの TDS 情報に対するリモート・サーバの応答を、DBPROCESS 構造体から、新しく割り付けられた DBLOGINFO 構造体に転送します。
  - srv_setloginfo － dbgetloginfo によって取得されたリモート・サーバの応答をクライアントに送信してから、DBLOGINFO 構造体を解放します。
- 次に、TDS パススルーを行うリモート接続を用意する SRV_CONNECT ハンドラの例を示します。

```
RETCODE connect_handler(srvproc)
SRVPROC       *srvproc;
```

```
{
    SYBLOGINFO    *loginfo;
    LOGINREC      *loginrec;
    DBPROCESS     *dbproc;

  /*
    ** Get the TDS login information from the
    ** client SRV_PROC.
    */
    srv_getloginfo(srvproc, &loginfo);

  /* Get a LOGINREC structure */
    loginrec = dblogin();

  /*
    ** Initialize the LOGINREC with the logininfo
    ** from the SRV_PROC.
    */
    dbsetloginfo(loginrec, loginfo);

  /* Connect to the remote server */
    dbproc = dbopen(loginrec, REMOTE_SERVER_NAME)

  /*
    ** Get the TDS login response informationfrom
    ** the remote connection.
    */
    dbgetloginfo(dbproc, &loginfo);

  /*
    ** Return the login response information to
    ** the SRV_PROC.
    */
    srv_setloginfo(srvproc, loginfo);

   /* Accept the connection and return */
    srv_senddone(srvproc, 0, 0, 0);
    return(SRV_CONTINUE);
}
```

参照　dbgetloginfo、dbrecvpassthru、dbsendpassthru

dbsetlogintime

説明

DB-Library が DBPROCESS 接続の要求に対するサーバの応答を待つ時間を秒単位で設定します。

構文

```
RETCODE dbsetlogintime(seconds)

int     seconds;
```

パラメータ

seconds

タイムアウト値です。DB-Library がログインの応答を待つ時間 (秒) がこの値に達するとタイムアウトになります。タイムアウト値 0 は、無期限のタイムアウト時間を示します。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、dbopen を呼び出した後に DB-Library がログインの応答を待つ時間を秒単位で設定します。デフォルトのタイムアウト値は 60 秒です。
- クライアントとサーバの間で接続を行うとき、接続が失敗する状況には次の 2 つがあります (システムは正しく構成されているものとします)。
  - サーバが存在するマシンが正しく動作しており、ネットワークも正しく動作している場合。

    この場合に、指定のポートで受信するサーバがないときは、サーバが存在するマシンは、接続が形成できないことをネットワーク・エラーによってクライアントに通知します。この場合、dbsetlogintime に関係なく、接続は実行できません。
  - サーバが存在するマシンがダウンしている場合。

    この場合は、サーバが存在するマシンが応答しません。「応答なし」はエラーとはみなされないので、エラーが発生したという通知がネットワークからクライアントに送られることはありません。ただし、dbsetlogintime の呼び出しによってタイムアウト時間が設定されている場合は、設定された時間内にクライアントが応答を受信しなければタイムアウト・エラーが発生します。

参照

dberrhandle、dbsettime

DBSETLPACKET

説明

アプリケーションの LOGINREC 構造体内の TDS パケット・サイズを設定します。

構文

RETCODE DBSETLPACKET(login, packet_size)

```
LOGINREC    *login;
short       packet_size;
```

パラメータ

login

サーバにログインするときに、dbopen に引数として渡される LOGINREC 構造体を指すポインタです。アプリケーションで LOGINREC 構造体を取得するには、dblogin を使用します。

packet_size

要求するパケット・サイズをバイト単位で指定します。サーバは、実際のパケット・サイズを、ここで要求したサイズ以下に設定します。

戻り値

SUCCEED または FAIL。

使用法

- DBSETLPACKET は、アプリケーションの LOGINREC 構造体内のパケット・サイズ・フィールドを設定します。アプリケーションがサーバにログインすると、サーバは、この DBPROCESS 接続の TDS パケット・サイズを、このフィールドの値以下に設定します。設定されるパケット・サイズがこのフィールドの値より小さくなるのは、サーバに領域の制約がある場合です。それ以外の場合は、パケット・サイズはこのフィールドの値と等しくなります。
- アプリケーションで大量の text データまたは image データを送受信する場合は、パケット・サイズをデフォルトの 512 バイトより大きくすれば、ネットワークの読み込みおよび書き込みが少なくなるので、効率的です。
- サーバが設定したパケット・サイズをアプリケーション側で調べるには、dbgetpacket を呼び出します。
- TDS (Tabular Data Stream) は、クライアントとサーバ間の要求と要求結果の転送に使用されるアプリケーション・プロトコルです。
- TDS データは、パケットと呼ばれる固定サイズのまとまりに分けて送信されます。TDS パケットのデフォルト・サイズは、512 バイトです。アプリケーションで TDS パケット・サイズを変更するには、DBSETLPACKET を使用する以外の方法はありません。DBSETLPACKET を呼び出さない場合は、アプリケーション内のすべての DBPROCESS 接続にデフォルトのサイズが使用されます。

- 1 つのアプリケーション内の DBPROCESS 接続ごとに異なるパケット・サイズを使用できます。アプリケーションで DBPROCESS 接続ごとに異なるパケット・サイズを設定するには、次のいずれかを行います。
  - LOGINREC を 1 つだけ使用し、DBPROCESS 接続を生成する dbopen 呼び出しと次の呼び出しの間にパケット・サイズを変更する。
  - 複数の LOGINREC 構造体を使用し、それぞれ異なるパケット・サイズを設定して、DBPROCESS 接続を生成するときにこれらの LOGINREC 構造体を使用する。
- DBPROCESS 接続の実際のパケット・サイズは DBPROCESS が生成されるときに設定されるため、DBSETLPACKET を呼び出しても、それより前に dbopen によって割り付けられた DBPROCESS のパケット・サイズには影響しません。

参照 dblogin、dbopen、dbgetpacket

DBSETLPWD

説明 LOGINREC 構造体内のユーザのサーバ・パスワードを設定する。

構文

```
RETCODE DBSETLPWD(loginrec, password)

LOGINREC        *loginrec;
char            *password;
```

パラメータ

loginrec
dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

password
サーバへ送られるパスワードです。これは NULL で終了する文字列でなくてはなりません。文字列の最大長は、NULL ターミネータを除いて 30 文字です。

戻り値 SUCCEED または FAIL。

使用法

- このマクロは、LOGINREC 構造体のユーザのサーバ・パスワードを設定します。これを有効にするには、dbopen の前に呼び出してください。

- LOGINREC のパスワード・フィールドのデフォルト値は NULL です。したがって、パスワードの値が NULL の場合は、このルーチンを呼び出す必要はありません。
- dbopen を呼び出した後に、*loginrec* のパスワードが DB-Library によって自動的に消去されることはありません。このため、DB-Library プログラム内に読み込み可能なパスワードが存在することの危険性を最小限に留めるには、dblogin を呼び出した後で *password* を他の値に設定してください。

参照　dblogin、dbopen、DBSETLAPP、DBSETLHOST、DBSETLUSER

DBSETLSERVERPRINCIPAL

説明　必要に応じて、LOGINREC 構造体内のサーバのプリンシパル名を設定します。

構文

```
DBSETLSERVERPRINCIPAL(loginrec, name)

LOGINREC        *loginrec;
char            *name;
```

パラメータ

loginrec
dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

name
サーバのプリンシパル名。文字列の最大長は、NULL ターミネータを除いて 255 文字です。

戻り値　SUCCEED または FAIL。

使用法

- DBSETLSERVERPRINCIPAL を有効にするには、dbopen() の前に呼び出し、DBSETLNETWORKAUTH を有効にする必要があります。
- DBSETLSERVERPRINCIPAL を呼び出さない場合、サーバ名がプリンシパル名として設定されます。

参照　dblogin、DBSETLMUTUALAUTH、DBSETLNETWORKAUTH

DBSETLUSER

説明

LOGINREC 構造体内のユーザ名を設定します。

構文

RETCODE DBSETLUSER(loginrec, username)

```
LOGINREC        *loginrec;
char            *username;
```

パラメータ

loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を割り付けるには、dblogin を呼び出します。

username

サーバへ送られるユーザ名です。これは NULL で終了する文字列でなくてはなりません。文字列の最大長は、NULL ターミネータを除いて 30 文字です。サーバは、*username* を使用して、接続を試行するユーザを識別します。サーバのユーザ名は、master データベース内の syslogins テーブル内に定義されます。

戻り値

SUCCEED または FAIL。

使用法

- このマクロは、LOGINREC 構造体内のユーザ名を設定します。これを有効にするには、dbopen の前に呼び出してください。
- ほとんどの環境で、このマクロは省略可能です。このマクロを呼び出さない場合、通常は DB-Library によってユーザ名にデフォルト値が設定されます。

注意 *UNIX* の場合：ユーザ名のデフォルトは UNIX のログイン名です。

MPE/XL の場合：ユーザ名のデフォルトはシステム環境変数 HPUSER の値です。

参照

dblogin、dbopen、DBSETLHOST、DBSETLPWD、DBSETLAPP

dbsetmaxprocs

説明

同時にオープンできる DBPROCESS 構造体の最大数を設定します。

構文

RETCODE dbsetmaxprocs(maxprocs)

```
int         maxprocs;
```

パラメータ

maxprocs
この特定のプログラムに対して同時にオープンできる DBPROCESS 構造体の数の新しい制限値です。

戻り値

SUCCEED または FAIL。

使用法

- 1 つの DB-Library プログラムで同時にオープンできる DBPROCESS 構造体の数には上限があります。デフォルトでは、この数は 25 です。プログラムで dbsetmaxprocs を呼び出すことにより、この制限値を変更できます。
- プログラムで現在の制限値を調べるには、dbgetmaxprocs を呼び出します。

参照

dbgetmaxprocs、dbopen

dbsetnull

説明

NULL 値をバインドするときに使用される代入値を定義します。

構文

```
RETCODE dbsetnull(dbproc, bindtype, bindlen, bindval)

DBPROCESS    *dbproc;
int          bindtype;
int          bindlen;
BYTE         *bindval;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

bindtype
代入値を適用する変数バインドの型を指定する記号値です (dbbind のリファレンス・ページを参照してください)。

bindlen
代入値のバイト単位の長さです。DB-Library は、CHARBIND と BINARYBIND の場合を除いて、この設定を無視します。他の型はすべて、固定長であるか、または特殊なターミネータあるいは埋め込みバイト・カウントによってデータ長を表すようになっています。

bindval

NULL の代入値として使用する値を指す、汎用の BYTE ポインタです。dbsetnull によってこの値がコピーされるので、dbsetnull を呼び出した後はこのポインタを解放できます。

戻り値

SUCCEED または FAIL。

dbsetnull は、認識できない *bindtype* が指定された場合に、FAIL を返します。指定された DBPROCESS のステータスが dead の場合にも、FAIL を返します。

使用法

- dbbind ルーチンと dbaltbind ルーチンは、結果カラムの値をプログラム変数にバインドします。アプリケーションでこれらのルーチンを呼び出した後は、dbnextrow や dbgetrow を呼び出すと、結果の値は、バインドされている変数へ自動的にコピーされます。サーバから結果カラムのいずれかに NULL 値が返されたときは、DB-Library によって自動的に代入値が結果の変数に格納されます。
- DBPROCESS のそれぞれに、各バインド型に対応する代入値のリストがあります。表 2-28 は、デフォルトの代入値のリストです。

表 2-28：デフォルトの null 代入値

バインド型	null 代入値
TINYBIND	0
SMALLBIND	0
INTBIND	0
CHARBIND	空文字列 (ブランクが埋め込まれる)
STRINGBIND	空文字列 (ブランクが埋め込まれ、NULL で終了する)
NTBSTRINGBIND	空文字列 (NULL で終了する)
VARYCHARBIND	空の文字列
BINARYBIND	空配列 (ゼロが埋め込まれる)
VARYBINBIND	空の配列
DATETIMEBIND	8 バイトのゼロ
SMALLDATETIMEBIND	8 バイトのゼロ
MONEYBIND	$0.00
SMALLMONEYBIND	$0.00
FLT8BIND	0.0
REALBIND	0.0
DECIMALBIND	0.0 (デフォルトの位取りおよび精度)
NUMERICBIND	0.0 (デフォルトの位取りおよび精度)
BOUNDARYBIND	空文字列 (NULL で終了する)
SENSITIVITYBIND	空文字列 (NULL で終了する)

- dbsetnull を使用すると、独自の NULL 代入値を指定できます。dbsetnull を呼び出して特定の NULL 代入値を変更した場合は、もう一度 dbsetnull を呼び出して変更するまで、指定した DBPROCESS に対して新しい値が適用されます。
- dbconvert ルーチンも、変換先の変数を NULL に設定する必要があるときに、現在の NULL 代入値を使用します。
- dbnullbind ルーチンを使用すると、バインドされたカラムにインジケータ変数を関連付けることができます。DB-Library は、null データ値または変換エラーを示すようにインジケータ値を設定します。

参照　dbaltbind、dbbind、dbconvert、dbnullbind、「データ型」(443 ページ)

dbsetopt

説明

サーバまたは DB-Library のオプションを設定します。

構文

```
RETCODE dbsetopt(dbproc, option, char_param,
        int_param)

DBPROCESS    *dbproc;
int          option;
char         *char_param;
int          int_param;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。*dbproc* が NULL の場合は、すべてのアクティブ DBPROCESS 構造体について、オプションが適用されます。

option

設定するオプションです。オプションのリストについては、「オプション」(436 ページ) を参照してください。

char_param

オプションにはパラメータを指定できるものがあります。たとえば、DBOFFSET オプションは構成体をパラメータとして受け取り、そのオフセットを返します。

```
dbsetopt(dbproc, DBOFFSET, "compute", -1)
```

DBBUFFER オプションは、バッファされるローの数をパラメータとして受け取ります。

```
dbsetopt(dbproc, DBBUFFER, "500", -1)
```

DBBUFFER の例のように、*char_param* は、数値であっても必ず引用符で囲まれた文字列にしてください。サーバ・オプションに不正なパラメータが指定されている場合に、このことが検出されるのは、次にコマンド・バッファがサーバに送信されるときです。その結果 dbsqlexec または dbsqlsend の呼び出しは失敗し、DB-Library は、ユーザがインストールしたメッセージ・ハンドラを呼び出します。DB-Library オプション (DBBUFFER または DBTEXTLIMIT) のいずれかに正しくないパラメータが指定されている場合には、dbsetopt 呼び出し自体が失敗します。

オプションがパラメータを必要としない場合、*char_param* は、NULL にしてください。

int_param
オプションによっては、追加のパラメータ *int_param* を必要とするものがあります。このパラメータは、*char_param* として渡される文字列の長さです。現時点では、DBPRCOLSEP、DBPRLINESEP、DBPRPAD だけがこのパラメータを必要とします。

int_param が必要ない場合は、-1 として渡してください。

戻り値

SUCCEED または FAIL。

char_param が DB-Library オプションに対して正しくない場合は、dbsetopt は失敗します。ただし、*char_param* がサーバ・オプションに対するものである場合は、正しくなくても dbsetopt は失敗しません。これは、コマンド・バッファがサーバへ送られるまで、このパラメータが正しいかどうかが検証されないためです。

使用法

- このルーチンは、サーバ・オプションと DB-Library オプションを設定します。サーバ・オプションは SQL を介して直接設定したりクリアしたりできますが、アプリケーションでは、オプションの設定とクリアには dbsetopt と dbclropt を使用してください。このようにすれば、サーバ・オプションと DB-Library オプションを設定するインタフェースを統一できます。また、アプリケーションで dbisopt 関数を使用してオプションのステータスを調べることもできます。
- dbsetopt は、オプションをすぐには設定しません。オプションは、dbsqlexec または dbsqlsend の呼び出しによって次にコマンド・バッファがサーバへ送信されるときに設定されます。
- 各オプションとそのデフォルト・ステータスのリストについては、「オプション」(436 ページ) を参照してください。

参照

dbclropt、dbisopt、「オプション」(436 ページ)

dbsetrow

説明

バッファされたローの 1 つを「現在のロー」に設定します。

構文

STATUS dbsetrow(dbproc, row)

```
DBPROCESS    *dbproc;
DBINT        row;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

row

現在のローに設定するローの番号を表す整数です。サーバから最初に返されたローの番号が 1 です。これは、ロー・バッファ内の最初のローとはかぎりません。

戻り値

MORE_ROWS、NO_MORE_ROWS、または FAIL。

dbsetrow は、次の値を返します。

- ロー・バッファ内に *row* がある場合、MORE_ROWS を返します。
- ロー・バッファ内に *row* がない場合、またはロー・バッファリングが使用不可能な場合、NO_MORE_ROWS を返します。
- *dbproc* で指定した DBPROCESS のステータスが dead または not enabled の場合、FAIL を返します。

使用法

- dbsetrow は、バッファされたローの 1 つを「現在のロー」に設定します。アプリケーションで dbsetrow を呼び出すと、次に dbnextrow を呼び出したときは、このローが読み込まれます。
- 別の DB-Library/C ルーチンである dbgetrow も、ロー・バッファ内の指定のローを「現在のロー」に設定します。ただし、dbsetrow とは異なり、dbgetrow はそのローを読み込みます。dbbind または dbaltbind によってロー・データからプログラム変数へのバインドが指定されている場合は、そのバインドが有効になります。
- DB-Library/C オプション DBBUFFER がオンでなければ、dbsetrow の効果はありません。

- ロー・バッファリングを利用すると、指定した数のサーバ結果ローをプログラムのメモリ内に保持できます。ロー・バッファリングを行わない場合は、新しい dbnextrow 呼び出しによって生成される結果ローは、直前の結果ローの内容を上書きします。したがって、プログラムでランダムな順序で結果ローにアクセスする場合は、ロー・バッファリングが便利です。ただし、バッファ内のローごとに割り付けおよび解放を行う必要があるので、メモリおよびパフォーマンスへの影響があります。したがって、この機能は必要な場合にだけ使用してください。特に、dbgetrow または dbsetrow を呼び出す場合にだけ DBBUFFER オプションをオンにしてください。ロー・バッファリングはネットワーク・バッファリングとは無関係であり、まったく別の問題である点に注意してください。
- アプリケーションでロー・バッファリングを使用できない場合は、NO_MORE_ROWS が返されるまで dbnextrow を繰り返し呼び出すことにより、サーバからローを読み込んだ時点でそのローを処理します。ロー・バッファリングが使用可能な場合は、dbsetrow を使用してこれまでにサーバから読み込んだ任意のローに dbnextrow を使用して移動できます。その後で dbnextrow を呼び出すと、アプリケーションはバッファ内の *row* パラメータで指定されたローから順に読み込むことになります。dbnextrow がバッファの最後のローに達したとき、さらにサーバからのローがある場合は、再度サーバからローを読み込みます。バッファが一杯のときは、dbclrbuf を使用して一部のローをバッファからクリアしなければ、dbnextrow を実行してもサーバからのローの読み込みは行われません。
- ロー・バッファ内の最初のローの番号を返すマクロ DBFIRSTROW は、dbsetrow と一緒に使用すると便利です。次に例を示します。

  ```
  dbsetrow(dbproc, DBFIRSTROW(dbproc))
  ```

  次の dbnextrow の呼び出しがバッファ内の最初のローを読み込むように現在のローを設定します。

参照　dbclrbuf、DBCURROW、DBFIRSTROW、dbgetrow、DBLASTROW、dbnextrow、「オプション」(436 ページ)

dbsettime

説明

DB-Library が SQL コマンドに対するサーバの応答を待つ時間を秒単位で設定します。

構文

```
RETCODE dbsettime(seconds)

int         seconds;
```

パラメータ

seconds

タイムアウト値です。つまり、DB-Library がサーバの応答を待つ時間 (秒) がこの値に達するとタイムアウトになります。タイムアウト値 0 は、無期限のタイムアウト時間を示します。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、dbsqlexec、dbsqlok、dbresults、dbnextrow の呼び出し中に、DB-Library がサーバの応答を待つ時間を秒単位で設定します。デフォルトのタイムアウト値は 0 で、無期限のタイムアウト時間を意味します。
- dbsettime は、dbopen の呼び出しの前でも後でも、アプリケーション内の任意の場所で呼び出すことができます。呼び出すと、設定はすぐに反映されます。
- dbopen 呼び出しのタイムアウト値を設定するには、dbsetlogintime を使用してください。
- dbsqlexec は、クエリをサーバへ送った後、応答を受け取るか、タイムアウト時間が経過するまで待つことに注意してください。DB-Library がサーバからの応答を待つ時間を最小にするためには、代わりに dbsqlsend を呼び出し、続いて dbsqlok を呼び出します。
- プログラムで現在のタイムアウト値を調べるには、DBGETTIME を呼び出します。
- タイムアウトが発生すると、DB-Library エラー「SYBETIME」が生成されます。

参照

dberrhandle、DBGETTIME、dbsetlogintime、dbsqlexec、dbsqlok、dbsqlsend

dbsetuserdata

説明

DBPROCESS 構造体を使用して、ユーザが割り付けたデータへのポインタを保存します。

構文

```
void dbsetuserdata(dbproc, ptr)

DBPROCESS     *dbproc;
BYTE          *ptr;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

ptr

ユーザのプライベートなデータ領域を指す、汎用の BYTE ポインタです。

戻り値

なし。

使用法

- DBPROCESS 構造体に、ユーザ割り付けのデータを指すポインタを保存します。アプリケーションは、dbgetuserdata ルーチンを使用して、後でそのデータにアクセスできます。
- dbsetuserdata により、アプリケーションはユーザ・データを特定の DBPROCESS と対応させることができます。これにより、グローバル変数を使用して対応付ける必要がなくなります。このルーチンの使い方の 1 つとして、次の例で示すようなデッドロックの処理があります。このルーチンは、アプリケーションに複数の DBPROCESS 構造体があるときに特に役立ちます。
- アプリケーションは、*ptr* が指すデータを割り付けなければなりません。DB-Library がこのデータを操作することはありません。DB-Library は、後でアプリケーションが使用できるように、このデータを指すポインタを保存します。
- 次に、このルーチンを使用してデッドロックを処理する例を示します。デッドロックは、大規模なアプリケーションで時々発生する状態です。詳細については、Adaptive Server Enterprise の『システム管理ガイド』を参照してください。このプログラムは、サーバに更新を送ります。メッセージ・ハンドラによってデッドロックが検出されたときは、トランザクションを再実行します。

```
...
/*
** Deadlock detection:
**      In the DBPROCESS structure, we save a pointer to
```

```
        **      a DBBOOL variable. The message handler sets the
        **      variable when deadlock occurs. The result
        **      processing logic checks the variable and resends
        **      the transaction in case of deadlock.
        */

         /*
         ** Allocate the space for the DBBOOL variable
         ** and save it in the DBPROCESS structure.
         */
                  dbsetuserdata(dbproc, malloc(sizeof(DBBOOL)));

                  /* Initialize the variable to FALSE */
                  *((DBBOOL *) dbgetuserdata(dbproc)) = FALSE;
                   ...
                  /* Run queries and check for deadlock */
deadlock:
       /*
       ** Did we get here using deadlock?
       ** If so, the server has already aborted the
       ** transaction. We'll just start it again. In a
       ** real application, the deadlock handling may need
       ** to be somewhat more sophisticated. For
       ** instance, you may want to keep a counter and
       ** retry the transaction just a fixed number
       ** of times.
       */
       if (*((DBBOOL *) dbgetuserdata(dbproc)) == TRUE)
       {
            /* Reset the variable to FALSE */
            *((DBBOOL *) dbgetuserdata(dbproc)) = FALSE;
       }
       /* Start the transaction */
       dbcmd(dbproc, "begin transaction ");
       /* Run the first update command */
       dbcmd(dbproc, "update ......");
       dbsqlexec(dbproc);
       while (dbresults(dbproc) != NO_MORE_RESULTS)
       {
            /* application code */
       }
       /*  Did we deadlock?*/
       if (*((DBBOOL *) dbgetuserdata(dbproc)) == TRUE)
            goto deadlock;
       /*  Run the second update command.*/
       dbcmd(dbproc, "update ......");
```

```
        dbsqlexec(dbproc);
        while (dbresults(dbproc) != NO_MORE_RESULTS)
        {
             /* application code */
        }
        /*  Did we deadlock?*/
        if (*((DBBOOL *) dbgetuserdata(dbproc)) == TRUE)
             goto deadlock;
        /*  No deadlock -- Commit the transaction */
        dbcmd(dbproc, "commit transaction");
        dbsqlexec(dbproc);
        dbresults(dbproc);
...


/*
** SERVERMSGS
** This is the server message handler. Assume that
** the dbmsghandle() routine installed it earlier in
** the program.
*/
servermsgs(dbproc, msgno, msgstate, severity, msgtext,
   srvname, procname, line)
DBPROCESS      *dbproc;
DBINT          msgno;
int            msgstate;
int            severity;
char           *msgtext;
char           *srvname;
char           *procname;
DBUSMALLINT    line;
{
        /*  Is this a deadlock message?*/
        if (msgno == 1205)
        {
             /* Set the deadlock indicator */
             *((DBBOOL *) dbgetuserdata(dbproc)) = TRUE;
             return (0);
        }
        /*  Normal message handling code here */
}
```

参照　dbgetuserdata

dbsetversion

説明

DB-Library のバージョン・レベルを指定します。

構文

```
RETCODE dbsetversion(version)

DBINT       version;
```

パラメータ

version

アプリケーションが想定する DB-Library の動作のバージョンです。表 2-29 は、*version* に指定できる記号値のリストです。

表 2-29 : version の値 (dbsetversion)

version の値	意味	サポートされる機能
DBVERSION_46	4.6 の動作	RPC、レジスタード・プロシージャ、リモート・プロシージャ・コール、text データ型、image データ型。 これは、DB-Library のデフォルト・バージョンです。
DBVERSION_100	10.0 の動作	numeric データ型および decimal データ型。

戻り値

SUCCEED または FAIL。

使用法

- dbsetversion は、アプリケーションが想定する DB-Library の動作のバージョンを設定します。DB-Library は、実際に使用されている DB-Library のバージョンに関係なく、要求された動作をします。
- アプリケーションでの dbsetversion の呼び出しは必須ではありません。ただし、dbsetversion を呼び出さなかった場合の DB-Library の動作レベルはリリース 4.6 となります。
- アプリケーションで dbsetversion を呼び出す場合は、どの DB-Library ルーチンの呼び出しよりも先に呼び出さなければなりません。ただし dbinit は例外です。

- dbsetversion の呼び出しを 2 回以上行うとエラーになります。

注意

- 環境変数 SYBOCS_DBVERSION を使用し、実行時の DB-Library バージョン・レベルを設定します。これを設定すると、この変数にバージョン・レベルとして保存された DB-Library 値を使用するようアプリケーション・コードが変更されます。
- この環境変数が設定されていなければ、DB-Library はリリース 4.6 の動作レベルを提供するか、明示的な dbsetversion 呼び出しによって要求されたバージョン・レベルを使用します。環境変数が設定され、dbsetversion が呼び出された場合、dbsetversion は環境変数を無効にします。

参照　dbinit

dbspid

説明　指定された DBPROCESS のサーバ・プロセス ID を取得します。

構文

```
int dbspid(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　*dbproc* のサーバ・プロセス ID。

使用法

- dbspid は、指定された DBPROCESS のサーバ・プロセス ID を取得します。このプロセス ID は、サーバの sysprocesses テーブル内に存在します。
- このサーバ・プロセス ID を使用して、sysprocesses テーブルに対するクエリを実行できます。

参照　dbopen

dbspr1row

説明

サーバのクエリ結果のローを 1 つバッファに入れます。

構文

RETCODE dbspr1row(dbproc, buffer, buf_len)

```
DBPROCESS    *dbproc;
char         *buffer;
DBINT        buf_len;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

buffer

dbspr1row の結果が格納される文字バッファを指すポインタです。

buf_len

null ターミネータを含む *buffer* の長さです。

戻り値

SUCCEED または FAIL。

注意 エラーが発生した場合は、**buffer* の内容は不定です。

使用法

- dbspr1row は、プログラマが用意したバッファに、サーバのクエリ結果のローの 1 つを、NULL で終了する文字列として格納します。
- dbspr1row は、デバッグ用のデータを表示したり、データ表示がスクロールするアプリケーションを作成したりするときに役立ちます。
- dbspr1row を使用すると、プログラマは、dbprrow を使用する場合よりも幅広くデータ表示を制御できます。dbprrow は常に表示デバイスに出力を書き込みますが、dbspr1row は、出力をバッファに書き込みます。これにより、プログラマは、いつでも任意の場所からデータを表示できます。
- 結果データが変換後の最大長になるように文字を埋め込むには、DB-Library の DBPRPAD オプションを使用して埋め込み文字を指定します。埋め込み文字は各カラムのデータに付加されます。変換後の最大カラム長は、そのカラムで可能な最長の文字列の長さです。これは、カラムに表示できるデータの長さとカラム名の長さのいずれか長い方です。DBPRPAD オプションの詳細については、「オプション」(436 ページ) を参照してください。

- カラム・セパレータ文字列を指定するには、DB-Library の DBPRCOLSEP オプションを使用します。カラム・セパレータは、変換された各カラム・データの後ろに追加されます。ただし、最後のデータには追加されません。デフォルトのセパレータは ASCII 0x20 (スペース) です。DBPRCOLSEP オプションの詳細については、「オプション」(436 ページ) を参照してください。
- 1 行の最大文字数を指定するには、DB-Library の DBPRLINELEN オプションを使用します。
- ライン・セパレータ文字列を指定するには、DB-Library の DBPRLINESEP オプションを使用します。デフォルトのライン・セパレータは、改行です (ASCII 0x0a または 0x0d で、ホスト・システムによって異なる)。DBPRLINELEN と DBPRLINESEP の詳細については、「オプション」(436 ページ) を参照してください。
- dbspr1row に必要なバッファの大きさを調べるには、dbspr1rowlen を呼び出します。
- dbspr1row が返す結果ローのフォーマットは、SQL クエリによって決まります。dbspr1row は、データを印刷可能文字に変換し、必要に応じてカラムに文字を埋め込み、カラム・セパレータとライン・セパレータを追加しますが、それ以上のフォーマットは行いません。
- dbspr1row を最大限に活用するには、dbnextrow の呼び出しが成功するたびに一度ずつ dbspr1row を呼び出すようにしてください。
- 次のコードは、dbspr1row の例です。

```
char     mybuffer[2000];

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
    dbspr1row(dbproc, mybuffer, sizeof(mybuffer));
    fprintf( stdout, "¥n%s", mybuffer);
  }
```

- 次のコードは、DBPRPAD オプションと DBPRCOLSEP オプションの例です。

```
    char     mybuffer[2000];

    /*
    ** Specify the pad and column separator
    ** characters */

    /* Pad = 0x2A */
    dbsetopt(dbproc, DBPRPAD, "*", DBPADON);
```

```
/* Col. sep. = 0x2C20 */
dbsetopt(dbproc, DBPRCOLSEP, ", ", 2);

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
     dbspr1row(dbproc, mybuffer,
          sizeof(mybuffer) );
     fprintf( stdout, "¥n%s", mybuffer);
}

/* Turn padding off */
dbsetopt(dbproc, DBPRPAD, SS, DBPADOFF );
/* Revert to default */
dbsetopt(dbproc, DBPRCOLSEP, RS, -1 );
```

参照 dbclropt、dbisopt、dbprhead、dbprrow、dbspr1rowlen、dbsprhead、dbsprline、「オプション」(436 ページ)

dbspr1rowlen

説明 dbsprhead、dbsprline、dbspr1row から返される結果を保持するために割り付けるバッファの大きさを決定します。

構文 DBINT dbspr1rowlen(dbproc)

DBPROCESS *dbproc;

パラメータ dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 正常に終了した場合は、dbsprhead、dbsprline、dbspr1row に必要なバッファのサイズをバイト単位で返します。エラーの場合は負の整数を返します。

使用法

- dbspr1rowlen は、dbsprhead、dbsprline、dbspr1row に必要なバッファのサイズを、バイト単位で計算します。このサイズには、null ターミネータも含まれます。
- dbspr1rowlen は、デバッグ用のデータを出力したり、データ表示をスクロールしたりするときに役立ちます。

- dbspr1rowlen を最大限に活用するには、dbresults の呼び出しが成功するたびに一度ずつ dbspr1rowlen を呼び出すようにしてください。
- 次のコードは、dbspr1rowlen の例です。

```
dbcmd(dbproc, "select * from sysdatabases");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(crdate) by name");

dbsqlexec(dbproc);
dbresults(dbproc);
printf("Maximum row length will be %ld ¥
      characters.¥n", dbspr1rowlen(dbproc));
```

参照　dbprhead、dbprrow、dbspr1row、dbsprhead、dbsprline、「オプション」(436 ページ)

dbsprhead

説明　サーバのクエリ結果のヘッダをバッファに入れます。

構文　RETCODE dbsprhead(dbproc, buffer, buf_len)

```
DBPROCESS    *dbproc;
char         *buffer;
DBINT        buf_len;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

バッファ (buffer)
　クエリ結果のヘッダが格納される文字バッファを指すポインタです。

buf_len
　null ターミネータを含む *buffer* の長さです。

戻り値　SUCCEED または FAIL。

注意　エラーが発生した場合は、**buffer* の内容は不定です。

使用法

- dbsprhead は、プログラマが用意したバッファに、現在のクエリ結果セットのヘッダを、null で終了する文字列として格納します。ヘッダはカラム名で構成されます。カラム名の順序は、dbspr1row の出力順序と一致します。
- dbsprhead は、デバッグ用のデータを出力したり、データ表示をスクロールしたりするときに役立ちます。
- カラム名が変換後の最大長になるように文字を埋め込むには、DB-Library の DBPRPAD オプションを使用して埋め込み文字を指定します。埋め込み文字は、各カラム名に付加されます。変換後の最大カラム長は、そのカラムで可能な最長の文字列の長さです。これは、カラムに表示できるデータの長さとカラム名の長さのいずれか長い方です。DBPRPAD オプションの詳細については、「オプション」(436 ページ) を参照してください。
- カラム・セパレータ文字列を指定するには、DB-Library の DBPRCOLSEP オプションを使用します。カラム・セパレータは、各カラム名の後ろに追加されます。ただし、最後のカラム名には追加されません。デフォルトのセパレータは ASCII 0x20 (スペース) です。DBPRCOLSEP オプションの詳細については、「オプション」(436 ページ) を参照してください。
- 1 行の最大文字数を指定するには、DB-Library の DBPRLINELEN オプションを使用します。

  ライン・セパレータ文字列を指定するには、DB-Library の DBPRLINESEP オプションを使用します。デフォルトのライン・セパレータは、改行です (ASCII 0x0a または 0x0d で、ホスト・システムによって異なる)。DBPRLINELEN と DBPRLINESEP の詳細については、「オプション」(436 ページ) を参照してください。
- dbsprhead に必要なバッファの大きさを調べるには、dbspr1rowlen を呼び出します。
- dbsprhead を最大限に活用するには、dbresults の呼び出しが成功するたびに一度ずつ dbsprhead を呼び出すようにしてください。
- 次のコードは、dbsprhead の例です。

```
dbcmd(dbproc, "select * from sysdatabases");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(crdate) by name");

dbsqlexec(dbproc);
dbresults(dbproc);

dbsprhead(dbproc, buffer, sizeof(buffer));
printf("%s¥n", buffer);
```

参照　dbprhead、dbprrow、dbsetopt、dbspr1row、dbspr1rowlen、dbsprline、「オプション」(436 ページ)

dbsprline

説明　dbsprhead が生成したカラム名に対応するアンダーラインが含まれる、フォーマットされた文字列を取得します。

構文　RETCODE dbsprline(dbproc, buffer, buf_len, linechar)

```
DBPROCESS    *dbproc;
char         *buffer;
DBINT        buf_len;
DBCHAR       linechar;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

バッファ (buffer)
　dbsprline の結果を格納する文字バッファを指すポインタです。

buf_len
　null ターミネータを含む *buffer* の長さです。

linechar
　dbsprhead によって生成されたカラム名に「アンダーライン」を付けるための文字です。

戻り値　SUCCEED または FAIL。

注意　エラーが発生した場合は、**buffer* の内容は不定です。

使用法

- dbsprline は、dbsprhead によって生成されたカラム名に「アンダーライン」を付けるために使用します。dbsprline は、現在のクエリ結果セットのカラムごとに、*linechar* で指定された文字のグループを生成し、この文字列に null ターミネータを付けたものを、プログラマが用意したバッファに格納します。この行のフォーマットは、dbsprhead の出力のフォーマットと一致します。
- dbsprline に必要なバッファの大きさを調べるには、dbspr1rowlen を呼び出します。

- dbsprhead を最大限に活用するには、dbresults の呼び出しが成功するたびに一度ずつ dbsprhead を呼び出すようにしてください。
- dbsprline は、デバッグ用のデータを出力したり、データ表示をスクロールしたりするときに役立ちます。
- 次のコードは、dbsprline の例です。

```
dbcmd(dbproc, "select * from sysdatabases");
dbcmd(dbproc, " order by name");
dbcmd(dbproc, " compute max(crdate) by name");

dbsqlexec(dbproc);
dbresults(dbproc);

/*
 ** Display the column headings, underline them
 ** with "*"
 */
 dbsprhead(dbproc, buffer, sizeof(buffer));
 printf("%s¥n", buffer);

dbsprline(dbproc, buffer, sizeof(buffer), '*');
printf("%s¥n", buffer);

/* Process returned rows as usual */
```

参照 dbprhead、dbprrow、dbspr1row、dbspr1rowlen、dbsprhead、「オプション」(436 ページ)

dbsqlexec

説明 コマンド・バッチをサーバに送信します。

構文 RETCODE dbsqlexec(dbproc)

DBPROCESS *dbproc;

パラメータ dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値 SUCCEED または FAIL。

失敗の主な原因は、SQL 構文エラーです。また、カラム名やテーブル名が正しくないなどの、セマンティック・エラーがある場合も dbsqlexec は失敗します。失敗は、バッチ内のコマンドに 1 つでもセマンティック・エラーや構文エラーがあると発生します。以前の結果がまだ処理されていないときや、コマンド・バッファが空の場合にも dbsqlexec は失敗します。

さらに、データベース保護違反のような実行時エラーも、dbsqlexec が失敗する原因となります。実行時エラーによって dbsqlexec が失敗するのは次の場合です。

- エラーを引き起こしたコマンドが、コマンド・バッファにある唯一のコマンドの場合
- エラーを引き起こしたコマンドが、複数コマンド・バッファにある最初のコマンドの場合

コマンド・バッファに複数のコマンドがあるときに、最初のコマンドに問題がなければ、実行時エラーによって dbsqlexec が失敗することはありません。代わりに、実行時エラーを引き起こしたコマンドを処理する dbresults 呼び出しが失敗します。

実行時エラーとストアド・プロシージャについては、状況がやや複雑になります。前述のルールにより、execute コマンドの実行時エラーが起こると、dbsqlexec は失敗します。ただし、ストアド・プロシージャ内の文で実行時エラーが発生しても、dbsqlexec は失敗しません。たとえば、ストアド・プロシージャに insert 文が含まれているけれども、データベース・テーブルへの挿入のパーミッションがユーザに付与されていない場合は、insert 文は失敗しますが、dbsqlexec はこの場合も SUCCEED を返します。ストアド・プロシージャ内で実行時エラーが発生したかどうかを調べるには、dbretstatus ルーチンを使用してプロシージャのリターン・ステータスを確認し、関連するサーバ・メッセージをメッセージ・ハンドラ内でトラップしてください。

使用法

- このルーチンは、DBPROCESS のコマンド・バッファに格納されている SQL コマンドをサーバに送信します。コマンドを DBPROCESS 構造体に追加するには、dbcmd または dbfcmd を呼び出します。
- dbsqlexec が SUCCEED を返したら、アプリケーションは、結果を処理するために dbresults を呼び出さなければなりません。
- 標準的な呼び出しのシーケンスを次に示します。

```
DBINT     xvariable;
DBCHAR    yvariable[10];

/* Read the query into the command buffer */
```

```
dbcmd(dbproc, "select x = 100, y = 'hello'");

/* Send the query to Adaptive Server Enterprise */
dbsqlexec(dbproc);

/* Get ready to process the query results */
dbresults(dbproc);

/* Bind column data to program variables */
dbbind(dbproc, 1, INTBIND, (DBINT) 0,
     (BYTE *) &xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0,
     yvariable);

/* Now process each row */
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
     C-code to print or process row data
}
```

- dbsqlexec は、dbsqlsend と dbsqlok を続けて実行することと同じです。ただし、dbsqlexec は、クエリをサーバに送信した後、応答を受け取るかタイムアウト時間が経過するまで待ちます。dbsqlexec の代わりに dbsqlok と dbsqlsend を使用すると、アプリケーションで複数の入力および出力ストリームに効率よく応答できます。dbsqlsend と dbsqlok のリファレンス・ページを参照してください。
- アプリケーションが dbsqlexec を呼び出すとき、コマンド・バッファに複数のコマンドが存在する場合があります。これらのコマンドは 1 つの単位としてサーバに送信され、1 つのコマンド・バッチとみなされます。

参照

dbcmd、dbfcmd、dbnextrow、dbresults、dbretstatus、dbsettime、dbsqlok、dbsqlsend

dbsqlok

説明

サーバからの結果を待ち、サーバが応答している命令の正当性を検証します。

構文

RETCODE dbsqlok(dbproc)

DBPROCESS *dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

失敗の主な原因は、SQL 構文エラーです。また、カラムやテーブル名が正しくないなどの、セマンティック・エラーがある場合も dbsqlok は失敗します。失敗は、バッチの中のコマンドに 1 つでもセマンティック・エラーや構文エラーがあると発生します。

さらに、データベース保護違反のような実行時エラーも、コマンド・バッファにコマンドが 1 つしかない場合には、dbsqlok が失敗する原因となります。コマンド・バッファに複数のコマンドがある場合は、実行時エラーが発生しても dbsqlok が失敗することはありません。代わりに、実行時エラーを引き起こしたコマンドを処理する dbresults 呼び出しが失敗します。

実行時エラーとストアド・プロシージャについては、状況がやや複雑になります。前述のルールにより、execute コマンドの実行時エラーが起こると、dbsqlok は失敗します。ただし、ストアド・プロシージャ内の文で実行時エラーが発生しても、dbsqlok は失敗しません。たとえば、ストアド・プロシージャに insert 文が含まれているけれども、データベース・テーブルへの挿入のパーミッションがユーザに付与されていない場合は、insert 文は失敗しますが、dbsqlok はこの場合も SUCCEED を返します。ストアド・プロシージャ内で実行時エラーが発生したかどうかを調べるには、dbretstatus ルーチンを使用してプロシージャのリターン・ステータスを確認し、関連するサーバ・メッセージをメッセージ・ハンドラ内でトラップしてください。

使用法

- dbsqlok は、サーバ・コマンドが成功したか失敗したかをレポートし、成功したコマンドの結果処理を開始します。
- dbsqlok の呼び出しが成功したら、続けて、結果を処理するために dbresults を呼び出さなければなりません。
- dbsqlok は次の場合に便利です。
    - dbsqlsend 呼び出しの後

      dbsqlsend を使用して Transact-SQL コマンドのバッチをサーバに送信した後に、dbsqlok を呼び出してください。
    - dbrpcsend 呼び出しの後

dbrpcinit、dbrpcparam、dbrpcsend を使用して RPC コマンドを送信した後に、dbsqlok を呼び出してください。

- dbwritetext または dbmoretext の呼び出しの後

  dbwritetext または dbmoretext を呼び出してテキスト更新コマンドをサーバに送信した後に、dbsqlok を呼び出してください。

dbsqlok を dbsqlsend と共に使用する

- dbsqlsend の呼び出し後、dbsqlok は結果処理を開始します。
- dbsqlok と dbsqlsend は、dbsqlexec の代わりに使用できます。dbsqlexec は、コマンド・バッチを送信してサーバからの最初の結果を待ちます。アプリケーションはブロックされ、結果を受け取るまでは他の処理は実行できません。dbsqlsend と dbsqlok を dbpoll と共に使用すると、アプリケーションをブロックしないで同様の処理を実行できます。この場合の代表的な制御シーケンスを次に示します。
  - dbsqlsend を呼び出して、コマンドをサーバに送信します。
  - ループ内で dbpoll を呼び出して、サーバからの結果が到着しているかどうかを調べます。ループの反復の間に、直接関係のない処理を実行することができます。結果が到着したことを dbpoll が示したときは、ループを終了します。
  - dbsqlok を呼び出すと、成功か失敗かがレポートされ、成功の場合は結果の処理が開始されます。

  注意 サーバからの応答の最初の数バイトしか存在していなくても、dbsqlok によって読み込むデータの準備ができていると dbpoll が報告することがあります。この場合、dbsqlexec と同じように、dbsqlok は残りの応答を待つか、またはタイムアウト時間が経過するまで待ちます。しかし、実際には、ほとんどの場合に応答全体が一度に使用可能になります。

- 次の例は、dbsqlok および dbpoll の使用方法を示したものです。この例では、アプリケーション関数 busy_wait, を呼び出して dbpoll ループを実行します。busy_wait を呼び出すメインライン・コードは次のとおりです。

```
/*
 ** This is a query that will take some time.
 */
 dbcmd(dbproc, "waitfor delay '00:00:05' select its = 'over'");
```

```
    /*
    ** Send the query with dbsqlsend. dbsqlsend does not
    ** wait for a server response.
    */
    retcode = dbsqlsend(dbproc);
    if (retcode != SUCCEED)
    {
      fprintf(stdout, "dbsqlsend failed. Exiting.¥n");
      dbexit();
      exit(ERREXIT);
    }

    /*
    ** If we call dbsqlok() now, it might block. But, we can use
    ** a dbpoll() loop to get some other work done while
    ** we are waiting for the results.
    */
    busy_wait(dbproc);

    /*
    ** Now there should be some results waiting to be read, so
    ** call dbsqlok().
    */
    retcode = dbsqlok(dbproc);
    if (retcode != SUCCEED)
    {
      fprintf(stdout, "Query failed.¥n");
    }
    else
    {
      ... dbresults() loop goes here ...
    }
```

busy_wait は dbpoll ループを実行します。ループの各反復では、dbpoll を呼び出して、結果が到着したかどうかを調べます。結果が到着している場合は、busy_wait は呼び出し元に戻ります。結果が到着していない場合は、wait_work 関数を呼び出します。wait_work は、直接関係のない作業を実行してから、呼び出し元に戻ります。関数 wait_work_init と wait_work_cleanup はそれぞれ wait_work の初期化処理とクリーンアップ処理を実行します。次にこれらの関数のコード例を示します。

```
void busy_wait(dbproc)
 DBPROCESS     *dbproc;
 {
   RETCODE retcode;
```

```
  DBPROCESS *ready_dbproc;
  int poll_ret_reason;

  wait_work_init();
  while (1)
  {
    retcode = dbpoll(dbproc, 0, &ready_dbproc, &poll_ret_reason);
    if (retcode != SUCCEED)
    {
      fprintf(stdout, "dbpoll() failed!Exiting.¥n");
      dbexit();
      exit(ERREXIT);
    }
    if (poll_ret_reason == DBRESULT)
    {
      /*
      ** Query results have arrived. Now we break out of
      ** the loop and return. Our caller can then call dbsqlok().
      */
      break; /* while */
    }
    else
    {
      /*
      ** Here's where we can do some non-related work while we
      ** are waiting.
      */
      wait_work();
    }
  } /* while */
  wait_work_cleanup();
} /* busy_wait */

/* These globals are used by the wait functions.*/
static int wait_pos;
static char wait_char;
void wait_work()
{

  /*
  ** "work", as defined here, consists of drawing a 'w' or 'W' to
  ** the terminal. We output one character each time we are called.
  ** When we reach the 65th character position, we switch from
  ** 'w' to 'W' (or vice-versa) and start over.
  */
  fputc(wait_char, stdout);
```

```
  ++wait_pos;
  if (wait_pos >= 65)
  {
     /*
     ** Go back to the beginning of the line, then switch from
     ** 'W' to 'w' or vice versa.
     */
     fputc('¥r', stdout);
     wait_pos = 0;
     wait_char = (wait_char == 'w' ?'W' :'w');
  }
}
void wait_work_init()
{
  wait_pos = 0;
  wait_char = 'w';
}
void wait_work_cleanup()
{
  fputc('¥n', stdout);
}
```

dbsqlok を dbrpcsend と共に使用する

- dbsqlok は、RPC コマンドの後に呼び出されると、結果の処理を開始します。RPC コマンドを作成して送信するには、dbrpcinit、dbrpcparam、dbrpcsend を使用します。プログラムでは、dbrpcsend の後に dbsqlok を呼び出さなければなりません。
- dbpoll をループ内で呼び出すことによって、dbrpcsend と dbsqlok との間のサーバの応答をポーリングすることができます。
- dbrpcinit、dbrpcparam、dbrpcsend のリファレンス・ページを参照してください。RPC コマンドの例については、オンラインのサンプル・プログラムの *example8.c* を参照してください。

dbsqlok を dbwritetext と dbmoretext と共に使用する

- dbsqlok は、テキスト更新コマンドの後で呼び出されると、結果の処理を開始します。テキストを更新する場合は、dbwritetext と dbmoretext を使用して、テキストをいくつかのまとまりに分けてサーバに送信します。これらを呼び出した後に、dbsqlok を呼び出さなければなりません。

- dbwritetext と dbmoretext のリファレンス・ページを参照してください。dbwritetext には例があります。

参照

dbcmd、dbfcmd、DBIORDESC、DBIOWDESC、dbmoretext、dbnextrow、dbpoll、DBRBUF、dbresults、dbretstatus、dbrpcsend、dbsettime、dbsqlexec、dbsqlsend、dbwritetext

dbsqlsend

説明

コマンド・バッチをサーバに送信します。応答は待ちません。

構文

```
RETCODE dbsqlsend(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

以前の結果がまだ処理されていない場合、またはコマンド・バッファが空である場合は、dbsqlsend は失敗します。

使用法

- このルーチンは、コマンド・バッファに格納されている SQL コマンドをサーバに送信します。アプリケーションでコマンドをコマンド・バッファに追加するには、dbcmd または dbfcmd を呼び出します。
- dbsqlsend から SUCCEED が返されたときは、コマンド・バッチの正当性を確認するために dbsqlok を呼び出す必要があります。その後で、dbresults を呼び出して結果を処理します。
- dbsqlexec は、dbsqlsend と dbsqlok を続けて実行することと同じです。
- dbsqlsend と dbsqlok を一緒に使用することは、UNIX のアプリケーションで特に有効です。クエリをサーバに送信した後、dbsqlexec は、応答を受け取るか、またはタイムアウト時間が経過するまで待ちます。dbsqlexec の代わりに dbsqlsend、dbpoll、および dbsqlok を使用すると、アプリケーションで複数の入力および出力ストリームに効率よく応答できます。詳細については、dbsqlok のリファレンス・ページを参照してください。

参照　dbcmd、dbfcmd、DBIORDESC、DBIOWDESC、dbnextrow、dbpoll、dbresults、dbsettime、dbsqlexec、dbsqlok

dbstrbuild

説明　変数のプレースホルダが含まれているテキストから、印刷可能な文字列を構築します。

構文

```
int dbstrbuild(dbproc, charbuf, bufsize,
        text [, formats [, arg] ...])

DBPROCESS    *dbproc;
char         *charbuf;
int          bufsize;
char         *text;
char         *formats;
???          args...;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信およびデータの管理に使用するすべての情報が含まれます。dbstrbuild が dbproc を使用する目的は、エラーが起きた場合に、プログラマがインストールしたエラー・ハンドラがあれば、そのパラメータとして渡すためだけです。

charbuf

dbstrbuild によって構築されたメッセージが格納されるバッファを指すポインタです。

bufsize

出力先バッファのバイト単位のサイズです。このサイズには、結果文字列の null ターミネータの 1 バイトも含まれます。

text

メッセージ・テキストと変数のプレースホルダを含む、null で終了する文字列を指すポインタです。プレースホルダは、パーセント記号と整数と感嘆符で構成されています。整数は、特定のプレースホルダに対してどの引数を置き換えるかを示します。引数とフォーマット文字列の番号は、左から右の順に付けられます。たとえば、引数 1 は、「%1!」のプレースホルダを置き換えます。

フォーマット
null で終了する文字列を指すポインタです。この文字列は、*text* 文字列のプレースホルダごとに 1 つの sprintf スタイルのフォーマット指定子で構成されます。

args
formats 文字列の内容に従って変換される値です。*formats* 文字列のフォーマットごとに 1 つずつの引数がなければなりません。最初の値は「%1!」パラメータに対応し、2 番目の値は「%2!」に対応します。フォーマットよりも引数が少ない場合は、その結果は不定です。フォーマットを最後まで置き換えても引数が残る場合は、残った引数は無視されます。

戻り値

成功した場合は、結果のメッセージ文字列の長さを返します。NULL ターミネータは含まれません。エラーの場合は、負の整数を返します。

使用法

- エラー・メッセージ内のパラメータの出現順序は、言語によって異なります。dbstrbuild を使用すると、C 標準ライブラリの sprintf ルーチンと同様の方法でエラー・メッセージを構成できます。dbstrbuild を使用することによって、エラー・メッセージをある言語から別の言語に簡単に変換できます。
- dbstrbuild は、エラー・テキストから、印刷可能な文字列を構築します。そのエラー・テキストには、変数のプレースホルダ、変数の型や書式についての情報を含むフォーマット文字列、変数に実際の値を代入するための可変数の引数が含まれます。
- 変数のプレースホルダは、パーセント記号と整数と感嘆符で構成されています。整数は、特定のプレースホルダに対してどの引数を置き換えるかを示します。引数とフォーマット文字列の番号は、左から右の順に付けられます。たとえば、引数 1 は、「%1!」のプレースホルダを置き換えます。

  たとえば、ストアド・プロシージャのキーワードの使い方が正しくないことを伝えるエラー・メッセージがあるとします。このメッセージには、3 つの引数が必要です。使い方に誤りのあるキーワード、そのキーワードのある行、およびストアド・プロシージャの名前です。英語のローカライゼーション・ファイルでは、メッセージ・テキストは次のようになります。

  ```
  The keyword '%1!' is misused in line %2! of stored
  procedure '%3!'.
  ```

  日本語のローカライゼーション・ファイルでは、同じメッセージが次のようになります。

  ```
  In line '%2!' of stored procedure '%3!', the keyword
  ```

```
'%1!' misused is.
```

上のどちらのメッセージの場合も、dbstrbuild の行は次のようになります。

```
dbstrbuild(dbproc, charbuf, BUFSIZE, <get the
message somehow>, "%s %d %s", keyword,
linenum, sp_name)
```

keyword はプレースホルダ「%1!」に代入され、*linenum* はプレースホルダ「%2!」に代入され、*sp_name* はプレースホルダ「%3!」に代入されます。

- 次のコードは、メッセージを構築する dbstrbuild の例です。簡単にするために、メッセージのテキストはハードコードされています。実際には、dbstrbuild のメッセージ・テキストは、ローカライゼーション・ファイルにあります。

```
char    charbuf[BUFSIZE];
int     linenum = 15;
char    *filename = "myfile";
char    *dirname = "mydir";

dbstrbuild (dbproc, charbuf, BUFSIZE,
"Unable to read line %1! of file %2! in ¥
directory %3!.", "%d %s %s", linenum,
filename, dirname);
printf(charbuf);
```

- dbstrbuild のフォーマット指定子は、他の文字で区切られていても、区切られずに隣接していてもかまいません。このため、既存の英語のメッセージの文字列を、dbstrbuild のフォーマット・パラメータとして使うことができます。最初のフォーマット指定子は「%1!」パラメータを表し、2 番目のフォーマット指定子は「%2!」パラメータを表すように指定します。

参照　dbconvert、dbdatename、dbdatepart

dbstrcmp

説明　指定されたソート順を使用して、2 つの文字列を比較する。

構文

```
int dbstrcmp(dbproc, str1, len1, str2, len2,
        sortorder)

DBPROCESS       *dbproc;
char            *str1;
int             len1;
char            *str2;
int             len2;
DBSORTORDER     *sortorder;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

str1

比較する最初の文字列を指すポインタです。*str1* には NULL を指定できます。

len1

str1 のバイト単位の長さです。*len1* が -1 のとき、*str1* は null で終了するとみなされます。

str2

比較する 2 番目の文字列を指すポインタです。*str2* には NULL を指定できます。

len2

str2 のバイト単位の長さです。*len2* が -1 のとき、*str1* は null で終了するとみなされます。

sortorder

dbloadsort によって割り付けられた DBSORTORDER 構造体を指すポインタです。*sortorder* が NULL の場合、dbstrcmp は strcmp と同じように、バイナリ値を使用して *str1* と *str2* を比較します。

戻り値

- *str1* が *str2* より辞書順で大きい場合は、1。
- *str1* と *str2* が辞書順で等しい場合は、0。
- *str1* が *str2* より辞書順で小さい場合は、-1。

使用法

- dbstrcmp は、*str1* と *str2* を比較して、*str1* が *str2* より辞書順で大きいか、等しいか、小さいかによって、0 より大きいか、等しいか、小さい整数を返します。

- dbstrcmp は、dbloadsort によってサーバから取得されたソート順を使用します。このため、DB-Library アプリケーション・プログラムは、サーバと同じソート順を使用して文字列を比較できます。
- 言語によっては、2 つの文字列を構成する文字が異なっていても、あるソート順に従うと、辞書順では等しいと判定されることがあります。これらの文字列は「等しい」かもしれませんが、順序リストに並べるときは標準の順序に従ってどちらを先にするかを決める必要があります。このような 2 つの文字列を比較するとき、dbstrcmp は 0 (2 つの文字列が等しいことを示す) を返しますが、dbstrsort は 0 以外の値を返し、ソートされたリストでどちらの文字列を先に配置するかを示します。

  次にこの例を示します。2 つの英語の文字列があり、大文字と小文字を区別しないソート順を使用します。このソート順では、大文字を小文字よりも前に置くように指定されています。

```
/* This call returns 0:*/
 dbstrcmp(dbproc, "ABC", 3, "abc", 3, mysort);

 /* This call returns a negative value:*/
 dbstrsort(dbproc, "ABC", 3, "abc", 3, mysort);
```

参照　dbfreesort、dbloadsort、dbstrsort

dbstrcpy

説明　コマンド・バッファのすべてまたは一部をコピーします。

構文　RETCODE dbstrcpy(dbproc, start, numbytes, dest)

```
DBPROCESS    *dbproc;
int          start;
int          numbytes;
char         *dest;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

start
コマンド・バッファ内のコピーを開始する文字の位置です。最初の文字の位置は、0 です。*start* がコマンド・バッファの長さよりも大きい場合、dbstrcpy は *dest[0]* に NULL ターミネータを挿入します。

numbytes
コピーする文字数です。*numbytes* が -1 のとき、dbstrcpy は、*dest* が指している領域の大きさが十分かどうかに関係なく、コマンド・バッファ全体をコピーします。コピーする長さが 0 バイトでもエラーにはなりません。この場合、dbstrcpy は *dest[0]* に NULL ターミネータを挿入します。コピーできる文字数が *numbytes* で指定された文字数より少なかった場合、dbstrcpy は、可能な数のバイトをコピーし、SUCCEED を返します。

dest
コピー元の文字列をコピーするためのコピー先バッファを指すポインタです。呼び出し元は、コピー先バッファがコピーした文字を格納するのに十分な大きさであることを確認してから dbstrcpy を呼び出す必要があります。関数 dbstrlen は、コマンド・バッファ全体のサイズを返します。

戻り値

SUCCEED または FAIL。

start が負のとき、dbstrcpy は FAIL を返します。

使用法

- dbstrcpy は、アプリケーションが用意した文字列バッファにコマンド・バッファの一部をコピーします。このコピーは、NULL で終了します。
- 内部的には、コマンド・バッファは NULL で終了しないテキスト文字列を連結したリストです。dbgetchar、dbstrcpy、dbstrlen を組み合わせると、コマンド・バッファ内の位置を指定してその部分をコピーできます。
- dbstrcpy は、コピー先の大きさが、コピー元の文字列を受け取るのに十分であると想定します。大きさが十分でない場合は、セグメンテーション・フォールトが発生することがあります。
- *numbytes* が -1 として渡されると、dbstrcpy はコマンド・バッファ全体をコピーします。*dest* が指す領域の大きさがこの文字列に対して十分であることが確実でない場合は、*numbytes* を -1 として渡さないでください。関数 dbstrlen は現在のコマンド文字列の長さを返します。
- コマンド・バッファ全体をファイルに出力する方法を次に示します。

```
FILE      *outfile;
DBPROCESS *dbproc;
char      *prbuf; /* buffer for collecting the command buffer
                   ** contents as a null-terminated string
                   */
 RETCODE      return_code;

/*
 ** Allocate sufficient space.  dbstrlen() returns the number of
 ** characters currently in the command buffer. We need one
 ** more byte because dbstrcpy will append a null terminator.
 ** NOTE that memory allocation and disposal may be done
 ** differently on your platform.
 **/
 prbuf = (char *) malloc(dbstrlen(dbproc) + 1);
 if (prbuf == NULL)
 {
   fprintf(stderr, "Out of memory.");
   dbexit();
   exit(ERREXIT); /* ERREXIT is defined in the DB-lib headers */
 }
 /* Copy the command buffer into the allocated space:*/
 return_code = dbstrcpy(dbproc, 0, -1, prbuf);
 assert(return_code == SUCCEED);

 /* Print the contents:*/
 fprintf(outfile, "%s", prbuf);

 /* Free the buffer:*/
 free(prbuf);
```

参照　dbcmd、dbfcmd、dbfreebuf、dbgetchar、dbstrlen

dbstrlen

説明　コマンド・バッファの長さを、文字数で返します。

構文　int dbstrlen(dbproc)

DBPROCESS　*dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

コマンド・バッファの長さを、文字数で返します。

使用法

- dbstrlen は、コマンド・バッファ内の SQL コマンド・テキストの長さを文字数で返します。
- 内部的には、コマンド・バッファは NULL で終了しないテキスト文字列を連結したリストです。dbgetchar、dbstrcpy、dbstrlen を組み合わせると、コマンド・バッファ内の位置を指定してその部分をコピーできます。
- dbstrlen を使用してコピー先バッファの大きさが十分であることを確認してから、dbstrcpy を使用してコマンド・バッファをコピーしてください。
- dbstrlen が返す数には、null ターミネータのスペースは含まれません。

参照

dbcmd、dbfcmd、dbfreebuf、dbgetchar、dbstrcpy

dbstrsort

説明

ソートされたリストで、2 つの文字列のどちらが先に置かれるかを調べます。

構文

```
int dbstrsort(dbproc, str1, len1, str2, len2,
        sortorder)

DBPROCESS       *dbproc;
char            *str1;
int             len1;
char            *str2;
int             len2;
DBSORTORDER     *sortorder;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

str1
比較する最初の文字列を指すポインタです。*str1* には NULL を指定できます。

len1
str1 のバイト単位の長さです。*len1* が -1 のとき、*str1* は null で終了するとみなされます。

str2
比較する 2 番目の文字列を指すポインタです。*str2* には NULL を指定できます。

len2
str2 のバイト単位の長さです。*len2* が -1 のとき、*str1* は null で終了するとみなされます。

sortorder
dbloadsort によって割り付けられた DBSORTORDER 構造体を指すポインタです。*sortorder* が NULL の場合、dbstrsort は strcmp と同じように、バイナリ値を使用して *str1* と *str2* を比較します。

戻り値

- *str1* が *str2* より後に置かれる場合は、1。
- *str1* と *str2* が同一の場合は、0。
- *str1* が *str2* より前に置かれる場合は、-1。

使用法

- dbstrsort は、*str1* と *str2* を比較して、ソートされたリスト内で *str1* が *str2* より後に置かれるか、同じ場所 (文字列が同一であることを示す) に置かれるか、または *str2* の前に置かれるかに応じて、0 より大きい整数か、等しい整数か、小さい整数を返します。
- dbstrsort は、dbloadsort によってサーバから取得されたソート順を使用します。このため、DB-Library アプリケーション・プログラムは、サーバと同じソート順を使用して文字列を比較できます。
- 言語によっては、2 つの文字列を構成する文字が異なっていても、あるソート順に従うと、辞書順では等しいと判定されることがあります。これらの文字列は「等しい」かもしれませんが、順序リストに並べるときは標準の順序に従ってどちらを先にするかを決める必要があります。このような 2 つの文字列を比較するとき、dbstrcmp は 0 (2 つの文字列が等しいことを示す) を返しますが、dbstrsort は 0 以外の値を返し、ソートされたリストでどちらの文字列を先に配置するかを示します。

  次にこの例を示します。2 つの英語の文字列があり、大文字と小文字を区別しないソート順を使用します。このソート順では、大文字を小文字よりも前に置くように指定されています。

```
/* This call returns 0:*/
 dbstrcmp(dbproc, "ABC", 3, "abc", 3, mysort);

 /* This call returns a negative value:*/
 dbstrsort(dbproc, "ABC", 3, "abc", 3, mysort);
```

- dbstrsort は、dbstrcmp によって等しいと判定された 2 つの文字列を検査するときにだけ使用できます。dbstrcmp の判定の結果、これらの文字列が等しくない場合は、dbstrsort の動作は定義されていません。

参照 dbfreesort、dbloadsort、dbstrcmp

dbtabbrowse

説明 指定されたテーブルが DB-Library のブラウズ・モード機能を使用して更新可能かどうかを調べます。

構文 DBBOOL dbtabbrowse(dbproc, tabnum)

```
DBPROCESS    *dbproc;
int          tabnum;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

tabnum

select 文の from 句で指定されている、対象となるテーブルの番号です。テーブル番号は 1 から始まります。

戻り値 「TRUE」または「FALSE」。

使用法

- dbtabbrowse は、DB-Library のブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- dbtabbrowse を使用すると、ブラウズ可能なテーブルを識別できます。これは、アドホック・クエリに基づいたブラウズ・モード更新を実行する前に、アドホック・クエリを検査するときに便利です。クエリがプログラムにハードコードされている場合には、このルーチンは必要ありません。

- テーブルが「ブラウズ可能」となるには、ユニーク・インデックスと timestamp カラムが必要です。
- アプリケーションは、dbresults の後はいつでも dbtabbrowse を呼び出すことができます。
- オンラインのサンプル・プログラムの *example7.c* に、dbtabbrowse の呼び出しが含まれています。

参照　dbcolbrowse、dbcolsource、dbqual、dbtabcount、dbtabname、dbtabsource、dbtsnewlen、dbtsnewval、dbtsput

dbtabcount

説明　現在の select クエリに含まれるテーブルの数を返します。

構文

```
int dbtabcount(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　現在の結果ロー・セットに含まれるテーブルの数を返します。この数には、サーバのワーク・テーブルも含まれます。

dbtabcount は、エラーの場合に -1 を返します。

使用法

- dbtabcount は、DB-Library のブラウズ・モード・ルーチンの 1 つです。これは、ブラウズ・モードの select (つまり、for browse キーワードを含む select) の結果に対してのみ使用可能です。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。

- select クエリは、結果ロー・セットを 1 つ生成します。この結果ローのカラムは、複数のデータベース・テーブルから導出されています。クエリの select リストに含まれるカラムに対してブラウズ・モードの更新を行うには、クエリに関係するテーブルの数をアプリケーションが認識している必要があります。これは、テーブルごとに別の update 文が必要となるためです。dbtabcount を使用すると、アドホック・クエリに関するこの情報を得ることができます。クエリがプログラムにハードコードされている場合には、このルーチンは必要ありません。
- このルーチンが返す数には、クエリの処理に使用される、サーバの「ワーク・テーブル」が含まれます。サーバは、クエリを処理するために、テンポラリの内部ワーク・テーブルを作成することがあります。サーバは、文の処理が終わると、これらのワーク・テーブルを削除します。ワーク・テーブルを更新することはできず、アプリケーションで使用することもできません。このため、アプリケーションは、テーブル番号を使用する前にその番号がワーク・テーブルのものではないことを確認しなければなりません。dbtabname を使用すると、特定のテーブル番号がワーク・テーブルを参照しているかどうかを調べることができます。
- アプリケーションは、dbresults の後はいつでも dbtabcount を呼び出すことができます。
- オンラインのサンプル・プログラムの *example7.c* に、dbtabcount の呼び出しが含まれています。

参照

dbcolbrowse、dbcolsource、dbqual、dbtabbrowse、dbtabname、dbtabsource、dbtsnewlen、dbtsnewval、dbtsput

dbtabname

説明

番号に基づいて、テーブル名を返します。

構文

```
char *dbtabname(dbproc, tabnum)

DBPROCESS    *dbproc;
int          tabnum;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

tabnum
対象となるテーブルの番号です。テーブル番号は 1 から始まります。特定のクエリに含まれるテーブルの総数を求めるには、dbtabcount を使用してください。

戻り値

指定されたテーブルの、NULL で終了する名前を指すポインタを返します。このポインタは、指定されたテーブル番号が範囲外のとき、または指定されたテーブルがサーバのワーク・テーブルのときに NULL になります。ワーク・テーブルの説明については、dbtabcount のリファレンス・ページを参照してください。

使用法

- dbtabname は、DB-Library のブラウズ・モード・ルーチンの 1 つです。これは、ブラウズ・モードの select (つまり、for browse キーワードを含む select) の結果に対してのみ使用可能です。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- select クエリによって生成される結果ローのセットは、複数のデータベース・テーブルから導出されていることがあります。dbtabname を使用すると、アドホック・クエリに関係する各テーブルの名前を調べることができます。クエリがプログラムにハードコードされている場合は、このルーチンは必要ありません。
- アプリケーションは、dbresults の後はいつでも dbtabname を呼び出すことができます。
- オンラインのサンプル・プログラムの *example7.c* に、dbtabname の呼び出しが含まれています。

参照

dbcolbrowse、dbcolsource、dbqual、dbtabbrowse、dbtabcount、dbtabsource、dbtsnewlen、dbtsnewval、dbtsput

dbtabsource

説明

特定の結果カラムの導出元テーブルの名前と番号を返します。

構文

```
char *dbtabsource(dbproc, colnum, tabnum)

DBPROCESS    *dbproc;
int          colnum;
int          *tabnum;
```

パラメータ

dbproc
: 特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

colnum
: 対象となる結果カラムの番号です。カラム番号は 1 から始まります。

tabnum
: テーブル番号が格納される整数を指すポインタです。ブラウズ・モードを扱う多くの DB-Library ルーチンが、テーブル名またはテーブル番号を受け取ります。dbtabsource が NULL を返す場合 (下記の「戻り値」の項を参照)、**tabnum* は -1 に設定されます。

戻り値

この結果カラムの導出元テーブルの名前を指すポインタを返します。戻り値が NULL の場合は、次のいずれかを意味します。

- DBPROCESS のステータスが dead または not enabled です。このエラーが発生したときは、アプリケーションのエラー・ハンドラが呼び出されます。
- カラム番号が範囲外です。
- カラムは、max(colname) などの式から生成されたものです。

使用法

- dbtabsource は、DB-Library のブラウズ・モード・ルーチンの 1 つです。これは、ブラウズ・モードの select (つまり、for browse キーワードを含む select) の結果に対してのみ使用可能です。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- dbtabsource を使用すると、現在の結果ロー・セットのカラムがどのテーブルからのものかを調べることができます。この情報は、アドホック・クエリに基づいて update 文や delete 文の where 句を構成するために dbqual を使用するときに有効です。クエリがプログラムにハードコードされている場合は、このルーチンは必要ありません。

- アプリケーションは、dbresults の後はいつでも dbtabsource を呼び出すことができます。
- オンラインのサンプル・プログラムの *example7.c* に、dbtabsource の呼び出しが含まれています。

参照　dbcolbrowse、dbcolsource、dbqual、dbtabbrowse、dbtabcount、dbtabname、dbtsnewlen、dbtsnewval

DBTDS

説明　使用されている TDS (Tabular Data Stream プロトコル) のバージョンを調べます。

構文

```
int DBTDS(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値　*dbproc* がサーバと通信するために使用している TDS のバージョンを返します。現時点では、可能なバージョンは次のとおりです。

- DBTDS_2_0
- DBTDS_3_4
- DBTDS_4_0
- DBTDS_4_2
- DBTDS_4_6
- DBTDS_4_9_5
- DBTDS_5_0

DBTDS は、エラーの場合に負の整数を返します。

使用法

- DBTDS は、*dbproc* がサーバと通信するために使用している TDS (Tabular Data Stream プロトコル) のバージョンを返します。

参照　dbversion

dbtextsize

説明

現在のローの text データまたは image データのうち、まだ読み込まれていない残りのバイト数を返します。

構文

DBINT dbtextsize(dbproc)

DBPROCESS *dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

次の表は、dbtextsize の戻り値のリストです。

dbtextsize の戻り値	意味
>= 0	読み込まれずに残っているバイト数。0 は NO_MORE_ROWS. を示す。
-1	エラーが発生した。
-2	dbtextsize が、RPC データに対して呼び出された。

使用法

- dbtextsize は、カラムが 1 つしか存在せず、そのカラムが text データ型または image データ型であるとみなします。
- dbtextsize は、アプリケーションが text または image の値の大きさを認識していないときに役立ちます。
- dbtextsize は、RPC text データに対しては機能しません。

参照

dbreadtext

dbtsnewlen

説明

ブラウズ・モードの更新の後で、*timestamp* カラムの新しい値の長さを返します。

構文

int dbtsnewlen(dbproc)

DBPROCESS *dbproc;

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

更新されたローの新しいタイムスタンプ値の長さをバイト単位で返します。アプリケーションにタイムスタンプが返されていない場合 (更新が失敗した場合、または update 文に組み込み関数 tsequal が含まれていない場合など)、dbtsnewlen は -1 を返します。

使用法

- dbtsnewlen は、DB-Library のブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- dbtsnewlen は、*timestamp* カラムについての情報を返します。dbqual から返される where 句には、組み込み関数 tsequal の呼び出しが含まれています。この where 句を update 文の中で使用すると、tsequal 関数によって、更新されたローの *timestamp* カラムに新しい値が格納され、その新しいタイムスタンプ値がアプリケーションに返されます (更新が成功した場合)。dbtsnewlen 関数を使用すると、新しいタイムスタンプ値の長さを保存できます。この長さを、dbtsput で使用できます。

参照

dbcolbrowse、dbcolsource、dbqual、dbtabbrowse、dbtabcount、dbtabname、dbtabsource、dbtsnewval、dbtsput

dbtsnewval

説明

ブラウズ・モードの更新の後で、*timestamp* カラムの新しい値を返します。

構文

```
DBBINARY *dbtsnewval(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

更新されたローの新しいタイムスタンプ値を指すポインタを返します。アプリケーションにタイムスタンプが返されていない場合 (更新が失敗した場合、または update 文に組み込み関数 tsequal が含まれていない場合など)、ポインタは NULL になります。

使用法

- dbtsnewval は、DB-Library のブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- dbtsnewval は、*timestamp* カラムについての情報を返します。dbqual から返される where 句には、組み込み関数 tsequal の呼び出しが含まれています。この where 句を update 文の中で使用すると、tsequal 関数によって、更新されたローの *timestamp* カラムに新しい値が格納され、その新しいタイムスタンプ値がアプリケーションに返されます (更新が成功した場合)。このルーチンを使用すると、新しいタイムスタンプ値を保存できます。この値を、dbtsput で使用できます。

参照

dbtabbrowse、dbtabsource、dbqual、dbtabbrowse、dbtabcount、dbtabname、dbtabsource、dbtsnewlen、dbtsput

dbtsput

説明

timestamp カラムの新しい値を、DBPROCESS 内の指定のテーブルの現在のローに入れます。

構文

```
RETCODE dbtsput(dbproc, newts, newtslen, tabnum,
        tabname)

DBPROCESS    *dbproc;
DBBINARY     *newts;
int          newtslen;
int          tabnum;
char         *tabname;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

これは、元の select クエリを実行するのに使用した DBPROCESS にしてください。

newts
新しいタイムスタンプ値を指すポインタです。dbtsnewval によって返されます。

newtslen
新しいタイムスタンプ値の長さです。dbtsnewlen によって返されます。

tabnum
更新されたテーブルの番号です。テーブル番号は 1 から始まります。*tabnum* で指定するテーブルは、ブラウズ可能でなければなりません。テーブルがブラウズ可能かどうかを調べるには、dbtabbrowse を使用してください。

この値が -1 の場合は、*tabname* パラメータで指定したテーブルが使用されます。

tabname
NULL で終了するテーブル名を指すポインタです。*tabname* で参照するテーブルは、ブラウズ可能にします。このポインタが NULL の場合は、*tabnum* パラメータで指定したテーブルが使用されます。

戻り値

SUCCEED または FAIL。

このルーチンは、次の場合に FAIL を返します。

- アプリケーションが、存在しないローのタイムスタンプを更新しようとした場合。
- アプリケーションが、タイムスタンプ値 (*newts*) に NULL を使用して、タイムスタンプを更新しようとした場合。
- 指定されたテーブルが、ブラウズ可能でない場合。

使用法

- dbtsput は、DB-Library のブラウズ・モード・ルーチンの 1 つです。ブラウズ・モードの詳細については、「ブラウズ・モード」(29 ページ) を参照してください。
- dbtsput は、タイムスタンプ・カラムを操作します。dbqual から返される where 句には、組み込み関数 tsequal の呼び出しが含まれています。この where 句を update 文の中で使用すると、tsequal 関数によって、更新されたローのタイムスタンプ・カラムに新しい値が格納され、その新しいタイムスタンプ値がアプリケーションに返されます (更新が成功した場合)。同じローをもう一度更新する場合は、update 文の where 句には最新のタイムスタンプ値を使用する必要があります。

このルーチンは、現在ブラウズしているローに対する、DBPROCESS 内のタイムスタンプを更新します。アプリケーションでそのローをもう一度更新する必要が生じたときは、dbqual を呼び出して、新しいタイムスタンプを使用する新しい where 句を構成します。

参照

dbcolbrowse、dbcolsource、dbqual、dbtabbrowse、dbtabcount、dbtabname、dbtabsource、dbtsnewlen、dbtsnewval

dbtxptr

説明

現在のローのカラムに対するテキスト・ポインタの値を返します。

構文

```
DBBINARY *dbtxptr(dbproc, column)

DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column

対象となる select リスト・カラムの番号です。カラム番号は 1 から始まります。

戻り値

対象となるカラムのテキスト・ポインタを指す DBBINARY ポインタを返します。このポインタは NULL のこともあります。

使用法

- SYBTEXT 型または SYBIMAGE 型のデータベース・カラム・ローのそれぞれにテキスト・ポインタが関連付けられており、text 値または image 値をユニークに識別します。このテキスト・ポインタは、dbwritetext 関数が text 値および image 値を更新するために使用します。
- 指定した text カラムまたは image カラムのすべてのローに、有効なテキスト・ポインタが存在している必要があります。text カラムまたは image カラムのローにデータが格納されていれば、そのローには有効なテキスト・ポインタがあります。ただし、text カラムまたは image カラムのローの内容が NULL 値のときは、そのテキスト・ポインタが有効になるのは、その NULL 値が update 文で明示的に入力された場合だけです。

key と x のカラムを持つ textnull というテーブルがあるとします。x は NULL を受け付ける text カラムです。次の文を実行すると、この text カラムのローに有効なテキスト・ポインタが割り当てられます。

```
update textnull
set x = null
```

一方、text カラムに NULL 値を insert したときは、有効なテキスト・ポインタは割り当てられません。このことは、明示的な NULL の insert の場合も、次のような暗黙的な NULL の insert の場合も同様です。

```
insert textnull (key)
values (2)
```

NULL の text 値または image 値を扱う場合は、有効なテキスト・ポインタを取得するために必ず update 文を使用してください。

- text 値または image 値に関連付けられたテキスト・ポイントを取得するために dbtxptr を呼び出す前に、その値が含まれているローを選択 (select) する必要があります。この select によって、テキスト・ポインタのコピーがアプリケーションの DBPROCESS 内に格納されます。その後で、dbtxptr を使用してこのテキスト・ポインタを DBPROCESS から取り出します。

  dbtxptr を呼び出す前に select が実行されていない場合、呼び出しの結果として DB-Library のエラー・メッセージが返されます。

- dbtxptr の使用例については、dbwritetext のリファレンス・ページを参照してください。

参照　dbtxtimestamp、dbwritetext

dbtxtimestamp

説明　現在のローのカラムに対するテキスト・タイムスタンプの値を返します。

構文　DBBINARY *dbtxtimestamp(dbproc, column)

```
DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column

対象となる select リスト・カラムの番号です。カラム番号は 1 から始まります。

戻り値

対象となるカラムのテキスト・タイムスタンプを指す DBBINARY ポインタを返します。このポインタは NULL のこともあります。

使用法

- SYBTEXT 型または SYBIMAGE 型のデータベース・カラムのそれぞれに、テキスト・タイムスタンプが関連付けられています。このテキスト・タイムスタンプは、そのカラムの最終変更時刻を記録します。テキスト・タイムスタンプを dbwritetext 関数と組み合わせて使用すると、競合するアプリケーション・ユーザどうしがデータベースの同じ値に対する相手の変更を誤って消去するような事態を回避できます。テキスト・タイムスタンプは、SYBTEXT カラムまたは SYBIMAGE カラムに対して Transact-SQL の select を実行したときに DBPROCESS に返されます。
- NULL 以外のテキスト・タイムスタンプの長さは必ず DBTXTSLEN になります (現時点では 8 バイトと定義されています)。
- text 値または image 値に関連付けられているテキスト・タイムスタンプを取得するために dbtxtimestamp を呼び出す前に、その値が含まれるローを選択 (select) する必要があります。この select によって、テキスト・タイムスタンプのコピーがアプリケーションの DBPROCESS 内に格納されます。その後で、dbtxtimestamp を使用してこのテキスト・タイムスタンプを DBPROCESS から取り出します。

  dbtxtimestamp を呼び出す前に select が実行されていない場合、呼び出しの結果として DB-Library のエラー・メッセージが返されます。
- dbtxtimestamp の使用例については、dbwritetext のリファレンス・ページを参照してください。

参照

dbtxptr、dbwritetext

dbtxtsnewval

説明

dbwritetext 呼び出しの後で、テキスト・タイムスタンプの新しい値を返します。

構文

```
DBBINARY *dbtxtsnewval(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

戻り値

dbwritetext オペレーションによって変更された SYBTEXT 値または SYBIMAGE 値の、新しいテキスト・タイムスタンプの値を指すポインタを返します。このポインタは NULL のこともあります。

使用法

- SYBTEXT 型または SYBIMAGE 型のデータベース・カラムのそれぞれに、テキスト・タイムスタンプが関連付けられており、カラムの値が変更されるとテキスト・タイムスタンプが更新されます。テキスト・タイムスタンプを dbwritetext 関数と組み合わせて使用すると、競合するアプリケーション・ユーザどうしがデータベースの同じ値に対する相手の変更を誤って消去するような事態を回避できます。テキスト・タイムスタンプは、SYBTEXT または SYBIMAGE カラムに対して Transact-SQL の select を実行したときに、DBPROCESS に返され、dbtxtimestamp を呼び出すことによって調べることもできます。
- dbwritetext オペレーション (複数の dbmoretext 呼び出しを含む場合がある) が成功するたびに、サーバは、更新されたテキスト・タイムスタンプの値を DB-Library に送り返します。dbtxtsnewval を利用すると、アプリケーションでこの新しいタイムスタンプの値を取得できます。
- アプリケーションでの dbtxtsnewval の使用方法は 2 つあります。1 つは、dbtxtsnewval からの戻り値を dbwritetext 呼び出しの *timestamp* パラメータとして使用するものです。もう 1 つは、dbtxtsnewval と dbtxtsput を組み合わせて、新しいタイムスタンプ値を DBPROCESS のロー・バッファに格納するものです。このようにすれば、後で dbtxtimestamp を使用してこのタイムスタンプ値にアクセスできます。これは、アプリケーションが結果ローをバッファリングし、新しいタイムスタンプをすぐには必要としないときに特に便利です。

参照

dbmoretext、dbtxtimestamp、dbtxtsput、dbwritetext

dbtxtsput

説明

DBPROCESS の現在のローの指定のカラムに、新しいテキスト・タイムスタンプの値を入れます。

構文

RETCODE dbtxtsput(dbproc, newtxts, colnum)

```
DBPROCESS    *dbproc;
DBBINARY     *newtxts;
int          colnum;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

newtxts
新しいテキスト・タイムスタンプの値を指すポインタです。これは dbtxtsnewval によって返されます。

colnum
対象となる select リスト・カラムの番号です。カラム番号は 1 から始まります。

戻り値

SUCCEED または FAIL。

使用法

- SYBTEXT 型または SYBIMAGE 型のデータベース・カラムのそれぞれに、テキスト・タイムスタンプが関連付けられており、カラムの値が変更されるとテキスト・タイムスタンプが更新されます。テキスト・タイムスタンプを dbwritetext 関数と組み合わせて使用すると、競合するアプリケーション・ユーザどうしがデータベースの同じ値に対する相手の変更を誤って消去するような事態を回避できます。テキスト・タイムスタンプは、SYBTEXT または SYBIMAGE カラムに対して Transact-SQL の select を実行したときに、DBPROCESS に返され、dbtxtimestamp を呼び出すことによって調べることもできます。

- dbwritetext オペレーション (複数の dbmoretext 呼び出しを含む場合がある) が成功するたびに、サーバは、更新されたテキスト・タイムスタンプの値を DB-Library に送り返します。アプリケーションは、dbtxtsnewval を使用することによって、この新しいタイムスタンプの値を取得できます。次に、dbtxtsput を使用して、この新しいタイムスタンプを DBPROCESS のロー・バッファに格納します。このようにすれば、後で dbtxtimestamp を使用してアクセスできるようになります。これは、アプリケーションが結果ローをバッファリングし、新しいタイムスタンプをすぐには必要としないときに特に便利です。

参照　dbmoretext、dbtxtimestamp、dbtxtsnewval、dbwritetext

dbuse

説明　特定のデータベースを使用します。

構文

```
RETCODE dbuse(dbproc, dbname)

DBPROCESS    *dbproc;
char         *dbname;
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

dbname
　使用するデータベースの名前です。

戻り値　SUCCEED または FAIL。

使用法

- このルーチンは、特定の DBPROCESS の指定されたデータベースに対して、Transact-SQL の use コマンドを発行します。このルーチンはコマンドを設定し、dbsqlexec と dbresults を呼び出します。
- 要求したデータベースのリカバリ処理が終了していないために use コマンドが失敗した場合、dbuse は成功するまで、または他のエラーを検出するまで、1 秒間隔で use コマンドを送信し続けます。

- このルーチンは、呼び出し側が指定した *dbproc* を使用します。さらに、その dbproc のコマンド・バッファも使用します。dbuse は、バッファ内にすでにコマンドがある場合は上書きし、終了するとバッファをクリアします。

参照

dbchange、dbname

dbvarylen

説明

指定された通常の結果カラムのデータ長が可変かどうかを調べます。

構文

```
DBBOOL dbvarylen(dbproc, column)

DBPROCESS    *dbproc;
int          column;
```

パラメータ

dbproc
特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

column
対象となる通常の結果カラムの番号です。最初のカラムの番号は 1 です。

戻り値

カラムのデータが可変長かどうかを示す「TRUE」または「FALSE」を返します。カラム番号が範囲外の場合も、dbvarylen は「FALSE」を返します。

使用法

- このルーチンは、指定した通常の結果カラム (非計算カラム) のデータが可変長かどうかを示します。結果カラムの導出元であるデータベース・カラムの型が varchar、varbinary、text, image、boundary、または sensitivity の場合は、「TRUE」を返します。導出元のデータベース・カラムが NULL として定義されている、つまりそのカラムに null 値が含まれる可能性がある場合も「TRUE」を返します。
- このルーチンは、アドホック・クエリを処理するプログラムで、NULL または可変長データが返される可能性があるかどうかを検出する場合に便利です。

- カラムのデータ型を調べるには、dbcoltype を使用します。データ型のリストについては、「データ型」(443 ページ) を参照してください。

参照　dbcollen、dbcolname、dbcoltype、dbdata、dbdatlen、dbnumcols、dbprtype

dbversion

説明　使用している DB-Library のバージョンを調べます。

構文　char *dbversion()

パラメータ　なし。

戻り値　使用している DB-Library のバージョンが格納された文字列を指すポインタを返します。

使用法　dbversion は、現在使用している DB-Library のバージョン番号が格納された文字列を指すポインタを返します。

参照　DBTDS

dbwillconvert

説明　特定のデータ型変換が DB-Library で可能かどうかを調べます。

構文　DBBOOL dbwillconvert(srctype, desttype)

```
int     srctype;
int     desttype;
```

パラメータ

srctype
変換されるデータのデータ型です。このパラメータは、表 2-30 に示すサーバ・データ型のいずれかです。

desttype
データの変換先のデータ型です。このパラメータは、表 2-30 に示すサーバ・データ型のいずれかです。

戻り値　データ型の変換がサポートされている場合は「TRUE」、サポートされていない場合は、「FALSE」を返します。

使用法

- プログラムでこのルーチンを使用すると、dbconvert で特定のデータ型変換を実行できるかどうかを判断できます。dbconvert が、サポートしていない変換の実行を指定されたときは、ユーザ提供のエラー・ハンドラがある場合はそのハンドルを呼び出し、グローバル・エラー番号を設定し、FAIL を返します。
- dbconvert は、どのサーバ・データ型で保管されているデータでも変換できます (すべての変換が可能というわけではありません)。表 2-30 に、サーバと DB-Library のデータ型を示します。

表 2-30 : サーバと DB-Library のデータ型

サーバ・タイプ	プログラム変数タイプ
SYBCHAR	DBCHAR
SYBTEXT	DBCHAR
SYBBINARY	DBBINARY
SYBIMAGE	DBBINARY
SYBINT1	DBTINYINT
SYBINT2	DBSMALLINT
SYBINT4	DBINT
SYBFLT8	DBFLT8
SYBREAL	DBREAL
SYBNUMERIC	DBNUMERIC
SYBDECIMAL	DBDECIMAL
SYBBIT	DBBIT
SYBMONEY	DBMONEY
SYBMONEY4	DBMONEY4
SYBDATETIME	DBDATETIME
SYBDATETIME4	DBDATETIME4
SYBBOUNDARY	DBCHAR
SYBSENSITIVITY	DBCHAR

- 表 2-8 (117 ページ) に、dbconvert と dbconvert_ps がサポートするデータ型変換を示します。表の左端の列が変換元のデータ型のリストで、上端の行が変換先のデータ型のリストです (簡潔にするために、各データ型のプレフィクスの「SYB」は省略されています)。dbwillconvert が「TRUE」(T) を返した場合は、その変換はサポートされています。「FALSE」(F) を返した場合は、その変換はサポートされていません。
- dbconvert または dbconvert_ps のリファレンス・ページを参照してください。

参照　dbaltbind、dbbind、dbconvert、dbconvert_ps、「データ型」(443 ページ)

dbwritepage

説明　1 ページ分のバイナリ・データをサーバに書き込みます。

警告！このルーチンがどのような処理を行うかを十分理解したうえで、このルーチンを使用してください。

構文

```
RETCODE dbwritepage(dbproc, dbname, pageno, size, buf)

DBPROCESS    *dbproc;
char         *dbname;
DBINT        pageno;
DBINT        size;
BYTE         buf[];
```

パラメータ

dbproc
　特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

dbname
　対象となるデータベースの名前です。

pageno
　書き込み先のデータベース・ページのページ番号です。

size
　サーバに書き込むバイト数です。現在、Adaptive Server Enterprise のデータベース・ページの長さは、2048 バイトです。

buf
　書き込むデータを保持するバッファを指すポインタです。

戻り値　SUCCEED または FAIL。

使用法　dbwritepage は、1 ページ分のバイナリ・データをサーバに書き込みます。このルーチンは、主に損傷したデータベースのページを検査および修復するのに使用します。dbwritepage を呼び出した後は、DBPROCESS にサーバからのエラー・メッセージまたは情報メッセージが格納されていることがあります。これらのメッセージには、ユーザ提供のメッセージ・ハンドラによってアクセスできます。

参照 dbmsghandle、dbreadpage

dbwritetext

説明 text 値または image 値をサーバに送信します。

構文

```
RETCODE dbwritetext(dbproc, objname, textptr,
                    textptrlen, timestamp, log,
                    size, text)

DBPROCESS     *dbproc;
char          *objname;
DBBINARY      *textptr;
DBTINYINT     textptrlen;
DBBINARY      *timestamp;
DBBOOL        log;
DBINT         size;
BYTE          *text;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信とデータの管理に使用するすべての情報が含まれます。

objname

データベースのテーブルとカラムの名前を、ピリオド (.) で区切って指定します。

textptr

変更される text 値または image 値のテキスト・ポインタを指すポインタです。このポインタを取得するには、dbtxptr を呼び出します。テキスト・ポインタは有効なものでなければなりません。詳細については、dbtxptr のリファレンス・ページを参照してください。

textptrlen

このパラメータは、今後の互換性を保持するためのものです。現時点では、このパラメータの値は、定義された定数 DBTXPLEN にします。

timestamp
変更する text 値または image 値のテキスト・タイムスタンプを指すポインタです。このポインタを取得するには、dbtxtimestamp または dbtxtsnewval を使用します。この値は、text 値または image 値が変更されると変更されます。このパラメータは省略可能で、NULL の指定もできます。

log
この dbwritetext オペレーションをトランザクション・ログに記録するかどうかを指定するブール値です。

size
書き込む text 値または image 値の合計サイズをバイト単位で指定します。dbwritetext は、このパラメータのみに基づいて送信バイト数を決定するので、この *size* が実際の値のサイズを超えてはなりません。

text
書き込まれる text 値または image 値が格納されているバッファのアドレスです。このポインタが NULL の場合は、*size* で指定したバイト数のデータがすべてサーバに送信されるまで、dbmoretext を繰り返し呼び出してください。

戻り値

SUCCEED または FAIL。

失敗の一般的な原因として、*timestamp* パラメータが正しくない場合が挙げられます。これは、アプリケーションが text カラムを取り出した時点と、そのカラムを更新するために dbwritetext を呼び出した時点の間に、別のアプリケーションが割り込んで更新を行った場合に発生します。

使用法

- dbwritetext は、SYBTEXT 値と SYBIMAGE 値を更新します。アプリケーションでこの関数を使用すると、更新する値を Transact-SQL の update 文にコピーしなくても、サーバに長い値を送信できます。さらに、dbwritetext を利用すれば、アプリケーションからテキスト・タイムスタンプ・メカニズムにアクセスできます。これにより、競合する 2 人のアプリケーション・ユーザがデータベースの同じ値に加えた相手の変更を誤って上書きしてしまうことを防止できます。
- *timestamp* パラメータは省略可能です。

timestamp パラメータを指定した場合は、*timestamp* パラメータの値がデータベースの text カラムのタイムスタンプと一致している場合にのみ dbwritetext は成功します。一致している場合は、dbwritetext によって text カラムが更新され、同時にそのカラムのタイムスタンプが現在の時刻で更新されます。これには、競合するアプリケーションによる更新を統制するという効果があります。あるアプリケーションが text カラムを検索してから dbwritetext を呼び出すまでの間に、別のアプリケーションがそのカラムを更新すると、最初のアプリケーションによる dbwritetext は失敗します。

timestamp パラメータを指定しない場合は、カラムのタイムスタンプ値に関係なく、dbwritetext によって text カラムが更新されます。

- *timestamp* パラメータとして使用する値は、アプリケーションが text 値または image 値に対する select を実行したときに、そのアプリケーションの DBPROCESS の中に格納されます。この値を DBPROCESS から取り出すには、dbtxtimestamp を使用します。

  さらに、dbwritetext オペレーション (dbmoretext を何度も呼び出す場合もあります) が成功するたびに、Adaptive Server Enterprise は新しいテキスト・タイムスタンプ値を DB-Library に送り返します。dbtxtsnewval を使用すると、アプリケーションでこの新しい値を取り出すことができます。

- dbwritetext は、機能としては Transact-SQL の writetext コマンドに似ています。通常は、コマンド・バッファから writetext コマンドを送るよりも、dbwritetext を呼び出す方が効率的です。さらに、writetext で処理できるデータはおよそ 120K までですが、dbwritetext は最大 2GB の長さのカラムを扱うことができます。『ASE リファレンス・マニュアル』を参照してください。

- dbwritetext を呼び出したときに、ログに記録するかどうかは、*log* パラメータの値に応じて決まります。

  ログに記録しておくとメディアのリカバリ時に役立ちますが、テキスト・データをログに記録すると、トランザクション・ログのサイズが急速に増加します。dbwritetext オペレーションをログに記録する場合は、トランザクション・ログが別のデータベース・デバイス上にあることを確認してください。詳細については、『システム管理ガイド』の create database のリファレンス・ページ、『ASE リファレンス・マニュアル』の sp_logdevice のリファレンス・ページを参照してください。

  ログへの記録をオフにした状態で dbwritetext を使用するには、データベース・オプション select into/bulkcopy が「true」に設定されている必要があります。次の SQL コマンドでこの設定を行います。

```
sp_dboption 'mydb', 'select into/bulkcopy', 'true'
```

sp_dboption の詳細については、『ASE リファレンス・マニュアル』を参照してください。

- アプリケーションは、サーバに text 値または image 値を一括して送ることも、一定のまとまりごとに送ることもできます。dbwritetext を単独で使用して、text 値または image 値全体の送信を処理します。dbwritetext とともに dbmoretext を使用すると、大きな text 値または image 値を、多数の小さなまとまりに分けてアプリケーションからサーバに送信することができます。これは、大きなデータ・バッファを割り付けることができないオペレーティング・システムを使用する場合に、特に役立ちます。
- text 値または image 値全体を送信するには、*text* パラメータを NULL 以外に設定します。このようにすると、データ転送の最初から最後までが dbwritetext によって行われ、必要に応じて dbsqlok と dbresults が呼び出されます。次のコードは、dbwritetext の例です。

```
LOGINREC      *login;
DBPROCESS     *q_dbproc;
DBPROCESS     *u_dbproc;
DBCHAR        abstract_var[512];

 /* Initialize DB-Library.*/
 if (dbinit() == FAIL)
      exit(ERREXIT);
 /*
 ** Open separate DBPROCESSes for querying and updating.
 ** This is not strictly necessary in this example,
 ** which retrieves only one row. However, this
 ** approach becomes essential when performing updates
 ** on multiple rows of retrieved data.
 */
 login = dblogin();
 q_dbproc = dbopen(login, NULL);
 u_dbproc = dbopen(login, NULL);

 /* The database column "abstract" is a text column.
  ** Retrieve the value of one of its rows.
 */
 dbcmd(q_dbproc, "select abstract from articles where ¥
      article_id = 10");
 dbsqlexec(q_dbproc);
 dbresults(q_dbproc);
 dbbind(q_dbproc, 1, STRINGBIND, (DBINT) 0,
      abstract_var);
```

```
/*
** For simplicity, we'll assume that just one row is
** returned.
*/
dbnextrow(q_dbproc);

/* Here we can change the value of "abstract_var" */
/* For instance ...*/
strcpy(abstract_var, "A brand new value.");

/* Update the text column */
dbwritetext (u_dbproc, "articles.abstract",
     dbtxptr(q_dbproc, 1), DBTXPLEN,
     dbtxtimestamp(q_dbproc, 1), TRUE,
     (DBINT)strlen(abstract_var), abstract_var);
/* We're all done */
dbexit();
```

- text 値または image 値全体を一度に送信するのではなく、いくつかのまとまりに分けて送信するには、*text* パラメータを NULL に設定します。これにより、dbwritetext は、テキストの転送がまもなく開始されることをサーバに通知するとすぐに、アプリケーションに制御を返します。実際のテキストは、dbmoretext によって送信されます。この関数は、まとまりごとに 1 回ずつ、複数回呼び出されます。次のコードは、dbwritetext を dbmoretext とともに使用する方法を示します。

```
LOGINREC      *login;
DBPROCESS     *q_dbproc;
DBPROCESS     *u_dbproc;
DBCHAR        part1[512];
static DBCHAR   part2[512] = " This adds another ¥
      sentence to the text.";

if (dbinit() == FAIL)
      exit(ERREXIT);

login = dblogin();
 q_dbproc = dbopen(login, NULL);
 u_dbproc = dbopen(login, NULL);

dbcmd(q_dbproc, "select abstract from articles where ¥
      article_id = 10");
 dbsqlexec(q_dbproc);
 dbresults(q_dbproc);
 dbbind(q_dbproc, 1, STRINGBIND, (DBINT) 0, part1);
```

```
/*
 ** For simplicity, we'll assume that just one row is
 ** returned.
 */
 dbnextrow(q_dbproc);

/*
 ** Here we can change the value of part of the text
 ** column. In this example, we will merely add a
 ** sentence to the end of the existing text.
 */

/* Update the text column */
 dbwritetext (u_dbproc, "articles.abstract",
 dbtxptr(q_dbproc, 1), DBTXPLEN,
 dbtxtimestamp(q_dbproc, 1), TRUE,
 (DBINT)(strlen(part1) + strlen(part2)), NULL);

dbsqlok(u_dbproc);
dbresults(u_dbproc);

/* Send the update value in chunks */
dbmoretext(u_dbproc, (DBINT)strlen(part1), part1);
dbmoretext(u_dbproc, (DBINT)strlen(part2), part2);

dbsqlok(u_dbproc);
dbresults(u_dbproc);
dbexit();
```

dbwritetext 呼び出しと最初の dbmoretext の呼び出しの間、および最後の dbmoretext 呼び出しの後に、dbsqlok と dbresults の呼び出しが必要となることに注意してください。

- dbwritetext を dbmoretext とともに使用するときは、指定したデータベースの text カラムがロックされます。このロックは、最後の dbmoretext によるデータの送信が完了すると解放されます。これにより、あるアプリケーションが更新している間に、別のアプリケーションがその text カラムを読み込んだり更新したりしないことが保証されます。
- ビューの中の text カラムまたは image カラムに対して dbwritetext を使用することはできません。

- DB-Library/C の DBTEXTSIZE オプションは、サーバのグローバル変数 @@*textsize* の値に反映されます。この変数は、Adaptive Server Enterprise が返す text 値または image 値のサイズを制限するものです。@@*textsize* のデフォルト値は 32,768 バイトです。アプリケーションで 32,768 バイトを超える text 値または image 値を取得する場合は、dbsetopt を呼び出して @@*textsize* を大きくする必要があります。
- DB-Library/C の DBTEXTLIMIT オプションは、DB-Library/C が読み込む text 値または image 値のサイズを制限します。

参照 dbmoretext、dbtxptr、dbtxtimestamp、dbwritetext、dbtxtsput

dbxlate

説明 ある文字セットから別の文字セットに文字列を変換します。

構文

```
int dbxlate(dbproc, src, srclen, dest, destlen, xlt,
        srcbytes_used, srcend, status)

DBPROCESS       dbproc;
char            *src;
int             srclen;
char            *dest;
int             destlen;
DBXLATE         *xlt;
int             *srcbytes_used;
DBBOOL          srcend;
int             *status;
```

パラメータ

dbproc

特定のフロントエンド／サーバ・プロセスを結びつけるための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／サーバ間の通信およびデータの管理に使用するすべての情報が含まれます。

src

変換する文字列を指すポインタです。

srclen

src のバイト単位の長さです。*srclen* が -1 の場合、*src* は NULL で終了するものとみなされます。

dest
変換後の文字列と NULL ターミネータが格納されるバッファを指すポインタです。

destlen
変換後の文字列が格納されるバッファの、バイト単位のサイズです。*destlen* が -1 の場合は、*dest* は変換後の文字列と NULL ターミネータを保持するのに十分な大きさであるとみなされます。

xlt
ある文字セットから別の文字セットへ文字列を変換するために使用される変換構造体を指すポインタです。この変換構造体は、dbload_xlate を使用して割り付けます。

srcbytes_used
実際に変換されたバイト数です。文字列全体を変換すると *dest* をオーバフローする場合は、*src* のうち、*dest* に収まる dbxlate だけが変換されます。*destlen* が -1 の場合は、*srcbytes_used* は *srclen* になります。

srcend
データがこれ以上あるかどうかを示すブール値です。*srcend* が「true」の場合は、これ以上渡されるデータはありません。*srcend* が「false」の場合は、*src* は変換する大きな文字列データの一部ですが、その文字列の終わりではありません。

status
変換された文字列のステータスを示すコードを指すポインタです。表 2-31 に、*status* カラムの値をリストします。

表 2-31：ステータスの値

status の値	意味
DBXLATE_XOF	変換後の文字列が *dest* をオーバフローした。
DBXLATE_XOK	変換が成功した。
DBXLATE_XPAT	*src* の末尾の部分は、変換可能なパターンの最初の部分である。これらのバイトは変換されていない。

戻り値

成功した場合は、実際に *dest* に格納されたバイト数が返されます。エラーの場合は、負の整数が返されます。

使用法

- dbxlate は、ある文字セットから別の文字セットへ文字列を変換します。これは、サーバの文字セットがディスプレイ・デバイスの文字セットと異なるときなどに使用します。
- 次のコードは、dbxlate の例です。

```
char          destbuf[128];
```

```
int             srcbytes_used;
DBXLATE*        xlt_todisp;
DBXLATE         *xlt_tosrv;

dbload_xlate((DBPROCESS *)NULL, "iso_1",
      "trans.xlt", &xlt_tosrv, &xlt_todisp);
printf("Original string:¥n¥t%s¥n¥n",
      TEST_STRING);
dbxlate((DBPROCESS *)NULL, TEST_STRING,
      strlen(TEST_STRING), destbuf, -1, xlt_todisp,
      &srcbytes_used);
printf("Translated to display character set:¥
      ¥n¥t%s¥n¥n", destbuf);
dbfree_xlate((DBPROCESS *)NULL, xlt_tosrv,
      xlt_todisp);
```

参照 dbload_xlate、dbfree_xlate

エラー

説明 すべての DB-Library エラーとエラー重大度のリスト。

構文

```
#include <sybfront.h>

#include <sybdb.h>

#include <syberror.h>
```

使用法

- DB-Library で発生する可能性のあるすべてのエラーとそのエラー重大度のリストです。
- 表 2-32 (420 ページ) は、エラー値のアルファベット順のリストです。この表の 2 番目のカラムは、それぞれのエラーに対応するエラー重大度の記号値です。3 番目のカラムは、エラーに対応するテキストです。
- 表 2-33 (436 ページ) は、エラーの重大度および対応する数値とエラーのタイプのリストです。
- エラーまたは情報イベントが発生したとき、アプリケーションの現在のエラー・ハンドラがある場合は、これらの数値がエラー・ハンドラに渡されます。アプリケーションは dberrhandle を呼び出してエラー・ハンドラをインストールします。

- エラー値は、ヘッダ・ファイル *sybdb.h* で定義されています。エラー重大度の値は、ヘッダ・ファイル *syberror.h* で定義されています。プログラムで *syberror.h* をインクルードする必要があるのは、エラー重大度の記号値を参照する場合のみです。

エラー

表 2-32 には、すべての DB-Library エラーが示されています。

表 2-32：エラー

エラー名	エラー重大度	エラー・テキスト
SYBEAAMT	EXPROGRAM	User attempted a dbaltbind with mismatched column and variable types.
SYBEABMT	EXPROGRAM	User attempted a dbbind with mismatched column and variable types.
SYBEABNC	EXPROGRAM	Attempt to bind to a non-existent column.
SYBEABNP	EXPROGRAM	Attempt to bind using NULL pointers.
SYBEABNV	EXPROGRAM	Attempt to bind to a NULL program variable.
SYBEACNV	EXCONVERSION	Attempt to do data-conversion with NULL destination variable.
SYBEADST	EXCONSISTENCY	International Release:Error in attempting to determine the size of a pair of translation tables.
SYBEAICF	EXCONSISTENCY	International Release:Error in attempting to install custom format.
SYBEALTT	EXCONSISTENCY	International Release:Error in attempting to load a pair of translation tables.
SYBEAOLF	EXRESOURCE	International Release:Error in attempting to open a localization file.
SYBEAPCT	EXCONSISTENCY	International Release:Error in attempting to perform a character set translation.
SYBEAPUT	EXPROGRAM	Attempt to print unknown token.
SYBEARDI	EXRESOURCE	International Release:Error in attempting to read datetime information from a localization file.
SYBEARDL	EXRESOURCE	International Release:Error in attempting to read the *dblib.loc* localization file.
SYBEASEC	EXPROGRAM	Attempt to send an empty command buffer to the server.

エラー名	エラー重大度	エラー・テキスト
SYBEASNL	EXPROGRAM	Attempt to set fields in a null LOGINREC.
SYBEASTL	EXPROGRAM	Synchronous I/O attempted at AST level.
SYBEASUL	EXPROGRAM	Attempt to set unknown LOGINREC field.
SYBEAUTN	EXPROGRAM	Attempt to update the timestamp of a table that has no timestamp column.
SYBEBADPK	EXINFO	Packet size of %1 not supported-size of %2 used instead!
SYBEBBCI	EXINFO	Batch successfully bulk copied to the server.
SYBEBBL	EXPROGRAM	Bad *bindlen* parameter passed to dbsetnull.
SYBEBCBC	EXPROGRAM	bcp_columns must be called before bcp_colfmt and bcp_colfmt_ps.
SYBEBCBNPR	EXPROGRAM	bcp_bind:if *varaddr* is NULL、*prefixlen* must be 0 and no terminator should be specified.
SYBEBCBNTYP	EXPROGRAM	bcp_bind:if *varaddr* is NULL and *varlen* greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY.
SYBEBCBPREF	EXPROGRAM	Illegal prefix length. Legal values are 0, 1, 2 or 4.
SYBEBCFO	EXUSER	bcp host files must contain at least one column.
SYBEBCHLEN	EXPROGRAM	*host_collen* should be greater than or equal to -1.
SYBEBCIS	EXCONSISTENCY	Attempt to bulk copy an illegally-sized column value to the server.
SYBEBCIT	EXPROGRAM	It is illegal to use BCP terminators with program variables other than SYBCHAR, SYBBINARY, SYBTEXT, or SYBIMAGE.
SYBEBCITBLEN	EXPROGRAM	bcp_init:*tblname* parameter is too long.

エラー名	エラー重大度	エラー・テキスト
SYBEBCITBNM	EXPROGRAM	bcp_init:*tblname* parameter cannot be NULL.
SYBEBCMTXT	EXPROGRAM	bcp_moretext may be used only when there is at least one text or image column in the Server table.
SYBEBCNL	EXNONFATAL	Negative length-prefix found in BCP datafile.
SYBEBCNN	EXUSER	Attempt to bulk copy a NULL value into a Server column which does not accept null values.
SYBEBCNT	EXUSER	Attempt to use Bulk Copy with a non-existent Server table.
SYBEBCOR	EXCONSISTENCY	Attempt to bulk copy an oversized row to the server.
SYBEBCPB	EXPROGRAM	bcp_bind, bcp_moretext and bcp_sendrow may not be used after bcp_init has been passed a non-NULL input file name.
SYBEBCPCTYP	EXPROGRAM	bcp_colfmt:If *table_colnum* is 0, *host_type* cannot be 0.
SYBEBCPI	EXPROGRAM	bcp_init must be called before any other bcp routines.
SYBEBCPN	EXPROGRAM	bcp_bind, bcp_collen, bcp_colptr, bcp_moretext and bcp_sendrow may be used only after bcp_init has been called with the copy direction set to DB_IN.
SYBEBCPREC	EXNONFATAL	Column %1!:Illegal precision value encountered.
SYBEBCPREF	EXPROGRAM	Illegal prefix length. Legal values are -1, 0, 1, 2 or 4.
SYBEBCRE	EXNONFATAL	I/O error while reading bcp datafile.
SYBEBCRO	EXINFO	The BCP hostfile '%1!' contains only %2! rows. It was impossible to read the requested %3! rows.
SYBEBCSA	EXUSER	The BCP hostfile '%1!' contains only %2! rows.Skipping all of these rows is not allowed.

エラー名	エラー重大度	エラー・テキスト
SYBEBCSET	EXCONSISTENCY	Unknown character set encountered.
SYBEBCSI	EXPROGRAM	Host-file columns may be skipped only when copying into the Server.
SYBEBCSNDROW	EXPROGRAM	bcp_sendrow may not be called unless all text data for the previous row has been sent using bcp_moretext.
SYBEBCSNTYP	EXPROGRAM	column number %1!:If *varaddr* is NULL and *varlen* greater than 0, the table column type must be SYBTEXT or SYBIMAGE and the program variable type must be SYBTEXT, SYBCHAR, SYBIMAGE or SYBBINARY.
SYBEBCUC	EXRESOURCE	bcp:Unable to close host datafile.
SYBEBCUO	EXRESOURCE	bcp:Unable to open host datafile.
SYBEBCVH	EXPROGRAM	bcp_exec may be called only after bcp_init has been passed a valid host file.
SYBEBCVLEN	EXPROGRAM	*varlen* should be greater than or equal to -1.
SYBEBCWE	EXNONFATAL	I/O error while writing bcp datafile.
SYBEBDIO	EXPROGRAM	Bad bulk copy direction. Must be either IN or OUT.
SYBEBEOF	EXNONFATAL	Unexpected EOF encountered in bcp datafile.
SYBEBIHC	EXPROGRAM	Incorrect host-column number found in bcp format file.
SYBEBIVI	EXPROGRAM	bcp_columns, bcp_colfmt and bcp_colfmt_ps may be used only after bcp_init has been passed a valid input file.
SYBEBNCR	EXPROGRAM	Attempt to bind user variable to a non-existent compute row.
SYBEBNUM	EXPROGRAM	Bad *numbytes* parameter passed to dbstrcpy.
SYBEBPKS	EXPROGRAM	In DBSETLPACKET, the packet size parameter must be between 0 and 999999.

エラー名	エラー重大度	エラー・テキスト
SYBEBPREC	EXPROGRAM	Illegal precision specified.
SYBEBPROBADDEF	EXCONSISTENCY	bcp protocol error:Illegal default column ID received.
SYBEBPROCOL	EXCONSISTENCY	bcp protocol error:Returned column count differs from the actual number of columns received.
SYBEBPRODEF	EXCONSISTENCY	bcp protocol error:Expected default information and got none.
SYBEBPRODEFID	EXCONSISTENCY	bcp protocol error:Default column ID and actual column ID are not same.
SYBEBPRODEFTYP	EXCONSISTENCY	bcp protocol error:Default value datatype differs from column datatype.
SYBEBPROEXTDEF	EXCONSISTENCY	bcp protocol error:More than one row of default information received.
SYBEBPROEXTRES	EXCONSISTENCY	bcp protocol error:Unexpected set of results received.
SYBEBPRONODEF	EXCONSISTENCY	bcp protocol error:Default value received for column that does not have default.
SYBEBPRONUMDEF	EXCONSISTENCY	bcp protocol error:Expected number of defaults differs from the actual number of defaults received.
SYBEBRFF	EXRESOURCE	I/O error while reading bcp format file.
SYBEBSCALE	EXPROGRAM	Illegal scale specified.
SYBEBTMT	EXPROGRAM	Attempt to send too much text data using the bcp_moretext call.
SYBEBTOK	EXCOMM	Bad token from the server:Datastream processing out of sync.
SYBEBTYP	EXPROGRAM	Unknown bind type passed to DB-Library function.
SYBEBTYPSRV	EXPROGRAM	Datatype is not supported by the server.
SYBEBUCE	EXRESOURCE	bcp:Unable to close error file.
SYBEBUCF	EXPROGRAM	bcp:Unable to close format file.

エラー名	エラー重大度	エラー・テキスト
SYBEBUDF	EXPROGRAM	bcp:Unrecognized datatype found in format file.
SYBEBUFF	EXPROGRAM	bcp:Unable to create format file.
SYBEBUFL	EXCONSISTENCY	DB-Library internal error-send buffer length corrupted.
SYBEBUOE	EXRESOURCE	bcp:Unable to open error file.
SYBEBUOF	EXPROGRAM	bcp:Unable to open format file.
SYBEBWEF	EXNONFATAL	I/O error while writing bcp error file.
SYBEBWFF	EXRESOURCE	I/O error while writing bcp format file.
SYBECAP	EXCOMM	DB-Library capabilities not accepted by the Server.
SYBECAPTYP	EXCOMM	Unexpected capability type in CAPABILITY datastream.
SYBECDNS	EXCONSISTENCY	Datastream indicates that a compute column is derived from a non-existent select list member.
SYBECDOMAIN	EXCONVERSION	Source field value is not within the domain of legal values.
SYBECINTERNAL	EXCONVERSION	Internal Conversion error.
SYBECLOS	EXCOMM	Error in closing network connection.
SYBECLPR	EXCONVERSION	Data conversion resulted in loss of precision.
SYBECNOR	EXPROGRAM	Column number out of range.
SYBECNOV	EXCONVERSION	Attempt to set variable to NULL resulted in overflow.
SYBECOFL	EXCONVERSION	Data conversion resulted in overflow.
SYBECONN	EXCOMM	Unable to connect: Adaptive Server Enterprise is unavailable or does not exist.
SYBECRNC	EXPROGRAM	The current row is not a result of compute clause %1!, so it is illegal to attempt to extract that data from this row.
SYBECRSAGR	EXPROGRAM	Aggregate functions are not allowed in a cursor statement.

エラー名	エラー重大度	エラー・テキスト
SYBECRSBROL	EXPROGRAM	Backward scrolling cannot be used in a forward scrolling cursor.
SYBECRSBSKEY	EXPROGRAM	Keyset cannot be scrolled backward in mixed cursors with a previous fetch type.
SYBECRSBUFR	EXPROGRAM	Row buffering should not be turned on when using cursor APIs.
SYBECRSDIS	EXPROGRAM	Cursor statement contains one of the disallowed phrases compute, union, for browse, or select into.
SYBECRSFLAST	EXPROGRAM	Fetch type LAST requires fully keyset driven cursors.
SYBECRSFRAND	EXPROGRAM	Fetch types RANDOM and RELATIVE can only be used within the keyset of keyset driven cursors.
SYBECRSFROWN	EXPROGRAM	Row number to be fetched is outside valid range.
SYBECRSFTYPE	EXRESOURCE	Unknown fetch type.
SYBECRSINV	EXPROGRAM	Invalid cursor statement.
SYBECRSINVALID	EXRESOURCE	The cursor handle is invalid.
SYBECRSMROWS	EXRESOURCE	Multiple rows are returned, only one is expected while retrieving dbname.
SYBECRSNOBIND	EXPROGRAM	Cursor bind must be called prior to dbcursor invocation.
SYBECRSNOCOUNT	EXPROGRAM	The DBNOCOUNT option should not be turned on when doing updates or deletes with dbcursor.
SYBECRSNOFREE	EXPROGRAM	The DBNOAUTOFREE option should not be turned on when using cursor APIs.
SYBECRSNOIND	EXPROGRAM	One of the tables involved in the cursor statement does not have a unique index.
SYBECRSNOKEYS	EXRESOURCE	The entire keyset must be defined for KEYSET type cursors.
SYBECRSNOLEN	EXRESOURCE	No unique index found.
SYBECRSNOPTCC	EXRESOURCE	No OPTCC was found.
SYBECRSNORDER	EXRESOURCE	The order of clauses must be from, where, and order by.

エラー名	エラー重大度	エラー・テキスト
SYBECRSNORES	EXPROGRAM	Cursor statement generated no results.
SYBECRSNROWS	EXRESOURCE	No rows returned, at least one is expected.
SYBECRSNOTABLE	EXRESOURCE	Table name is NULL.
SYBECRSNOUPD	EXPROGRAM	Update or delete operation did not affect any rows.
SYBECRSNOWHERE	EXPROGRAM	A where clause is not allowed in a cursor update or insert.
SYBECRSNUNIQUE	EXRESOURCE	No unique keys associated with this view.
SYBECRSORD	EXPROGRAM	Only fully keyset driven cursors can have order by, group by, or having phrases.
SYBECRSRO	EXPROGRAM	Data locking or modifications cannot be made in a read-only cursor.
SYBECRSSET	EXPROGRAM	A set clause is required for a cursor update or insert.
SYBECRSTAB	EXPROGRAM	Table name must be determined in operations involving data locking or modifications.
SYBECRSVAR	EXRESOURCE	There is no valid address associated with this bind.
SYBECRSVIEW	EXPROGRAM	A view cannot be joined with another table or a view in a cursor statement.
SYBECRSVIIND	EXPROGRAM	The view used in the cursor statement does not include all the unique index columns of the underlying tables.
SYBECRSUPDNB	EXPROGRAM	Update or insert operations cannot use bind variables when binding type is NOBIND.
SYBECRSUPDTAB	EXPROGRAM	Update or insert operations using bind variables require single table cursors.
SYBECSYN	EXCONVERSION	Attempt to convert data stopped by syntax error in source field.
SYBECUFL	EXCONVERSION	Data conversion resulted in underflow.

エラー名	エラー重大度	エラー・テキスト
SYBEDBPS	EXRESOURCE	Maximum number of DBPROCESS already allocated.
SYBEDDNE	EXINFO	DBPROCESS is dead or not enabled.
SYBEDIVZ	EXUSER	Attempt to divide by $0.00 in function %1!.
SYBEDNTI	EXPROGRAM	Attempt to use dbtxtsput to put a new text timestamp into a column whose datatype is neither SYBTEXT nor SYBIMAGE.
SYBEDPOR	EXPROGRAM	Out-of-range *datepart* constant.
SYBEDVOR	EXPROGRAM	Day values must be between 1 and 7.
SYBEECAN	EXINFO	Attempted to cancel unrequested event notification.
SYBEEINI	EXINFO	Must call dbreginit before dbregexec.
SYBEETD	EXPROGRAM	Failure to send the expected amount of text or image data using dbmoretext.
SYBEEUNR	EXCOMM	Unsolicited event notification received.
SYBEEVOP	EXINFO	Called dbregwatch with a bad options parameter.
SYBEEVST	EXINFO	Must initiate a transaction before calling dbregparam.
SYBEFCON	EXCOMM	Adaptive Server Enterprise connection failed.
SYBEFRES	EXFATAL	Challenge-Response function failed.
SYBEFSHD	EXRESOURCE	Error in attempting to find the Sybase home directory.
SYBEFUNC	EXPROGRAM	Functionality not supported at the specified version level.
SYBEICN	EXPROGRAM	Invalid *computeid* or compute column number.
SYBEIDCL	EXCONSISTENCY	Illegal datetime column length returned by Adaptive Server Enterprise. Legal datetime lengths are 4 and 8 bytes.
SYBEIDECCL	EXCONSISTENCY	Invalid decimal column length returned by the server.

エラー名	エラー重大度	エラー・テキスト
SYBEIFCL	EXCONSISTENCY	Illegal floating-point column length returned by Adaptive Server Enterprise. Legal floating-point lengths are 4 and 8 bytes.
SYBEIFNB	EXPROGRAM	Illegal field number passed to bcp_control.
SYBEIICL	EXCONSISTENCY	Illegal integer column length returned by Adaptive Server Enterprise. Legal integer lengths are 1, 2, and 4 bytes.
SYBEIMCL	EXCONSISTENCY	Illegal money column length returned by Adaptive Server Enterprise. Legal money lengths are 4 and 8 bytes.
SYBEINLN	EXUSER	Interface file:unexpected end-of-line.
SYBEINTF	EXUSER	Server name not found in interface file.
SYBEINUMCL	EXCONSISTENCY	Invalid numeric column length returned by the server.
SYBEIPV	EXINFO	%1! is an illegal value for the %2! parameter of %3!.
SYBEISOI	EXCONSISTENCY	International Release:Invalid sort-order information found.
SYBEISRVPREC	EXCONSISTENCY	Illegal precision value returned by the server.
SYBEISRVSCL	EXCONSISTENCY	Illegal scale value returned by the server.
SYBEITIM	EXPROGRAM	Illegal timeout value specified.
SYBEIVERS	EXPROGRAM	Illegal version level specified.
SYBEKBCI	EXINFO	1000 rows sent to the server.
SYBEKBCO	EXINFO	1000 rows successfully bulk copied to host file.
SYBEMEM	EXRESOURCE	Unable to allocate sufficient memory.
SYBEMOV	EXUSER	Money arithmetic resulted in overflow in function %1!.
SYBEMPLL	EXUSER	Attempt to set maximum number of DBPROCESSes lower than 1.
SYBEMVOR	EXPROGRAM	Month values must be between 1 and 12.

エラー名	エラー重大度	エラー・テキスト
SYBENBUF	EXINFO	Called dbsendpassthru with a NULL *buf* parameter.
SYBENBVP	EXPROGRAM	Cannot pass dbsetnull a NULL *bindval* pointer.
SYBENDC	EXPROGRAM	Cannot have negative component in date in numeric form.
SYBENDTP	EXPROGRAM	Called dbdatecrack with NULL datetime parameter.
SYBENEG	EXCOMM	Negotiated login attempt failed.
SYBENHAN	EXINFO	Called dbrecvpassthru with a NULL handle parameter.
SYBENMOB	EXPROGRAM	No such member of order by clause.
SYBENOEV	EXINFO	DBPOLL can not be called when registered procedure notifications have been disabled.
SYBENPRM	EXPROGRAM	NULL parameter not allowed for this dboption.
SYBENSIP	EXPROGRAM	Negative starting index passed to dbstrcpy.
SYBENTLL	EXUSER	Name too long for LOGINREC field.
SYBENTTN	EXPROGRAM	Attempt to use dbtxtsput to put a new text timestamp into a non-existent data row.
SYBENULL	EXINFO	NULL DBPROCESS pointer passed to DB-Library.
SYBENULP	EXPROGRAM	Called %s with a NULL %s parameter.
SYBENXID	EXNONFATAL	The Server did not grant us a distributed-transaction ID.
SYBEONCE	EXPROGRAM	Function can be called only once.
SYBEOOB	EXCOMM	Error in sending out-of-band data to the server.
SYBEOPIN	EXNONFATAL	Could not open interface file.
SYBEOPNA	EXNONFATAL	Option is not available with current server.

エラー名	エラー重大度	エラー・テキスト
SYBEOREN	EXINFO	International Release:Warning: an out-of-range error-number was encountered in *dblib.loc*. The maximum permissible error-number is defined as DBERRCOUNT in *sybdb.h*.
SYBEORPF	EXUSER	Attempt to set remote password would overflow the login record's remote password field.
SYBEPOLL	EXINFO	There is already an active dbpoll.
SYBEPRTF	EXINFO	dbtracestring may only be called from a *printfunc*.
SYBEPWD	EXUSER	Login incorrect.
SYBERDCN	EXCONVERSION	Requested data conversion does not exist.
SYBERDNR	EXPROGRAM	Attempt to retrieve data from a non-existent row.
SYBEREAD	EXCOMM	Read from the server failed.
SYBERESP	EXPROGRAM	Response function address passed to dbresponse must be non-NULL.
SYBERPCS	EXINFO	Must call dbrpcinit before dbrpcparam or dbrpcsend.
SYBERPIL	EXPROGRAM	It is illegal to pass -1 to dbrpcparam for the *datalen* of parameters which are of type SYBCHAR, SYBVARCHAR, SYBBINARY, or SYBVARBINARY.
SYBERPNA	EXNONFATAL	The RPC facility is available only when using a server whose version number is 4.0 or later.
SYBERPND	EXPROGRAM	Attempt to initiate a new Adaptive Server Enterprise operation with results pending.
SYBERPNULL	EXPROGRAM	*value* parameter for dbrpcparam can be NULL, only if the *datalen* parameter is 0.
SYBERPTXTIM	EXPROGRAM	RPC parameters cannot be of type text or image.

エラー名	エラー重大度	エラー・テキスト
SYBERPUL	EXPROGRAM	When passing a SYBINTN, SYBDATETIMN, SYBMONEYN, or SYBFLTN parameter using dbrpcparam, it is necessary to specify the parameter’s maximum or actual length so that DB-Library can recognize it as a SYINT1, SYBINT2, SYBINT4, SYBMONEY, SYBMONEY4, and so on.
SYBERTCC	EXPROGRAM	dbreadtext may not be used to receive the results of a query that contains a COMPUTE clause.
SYBERTSC	EXPROGRAM	dbreadtext may be used only to receive the results of a query that contains a single result column.
SYBERXID	EXNONFATAL	The Server did not recognize our distributed-transaction ID.
SYBESECURE	EXPROGRAM	Secure Adaptive Server Enterprise function not supported in this version.
SYBESEFA	EXPROGRAM	DBSETNOTIFS cannot be called if connections are present.
SYBESEOF	EXCOMM	Unexpected EOF from the server.
SYBESFOV	EXPROGRAM	International Release:dbsafestr overflowed its destination buffer.
SYBESMSG	EXSERVER	General Adaptive Server Enterprise error:Check messages from the server.
SYBESOCK	EXCOMM	Unable to open socket.
SYBESPID	EXPROGRAM	Called dbspid with a NULL dbproc.
SYBESYNC	EXCOMM	Read attempted while out of synchronization with Adaptive Server Enterprise.
SYBETEXS	EXINFO	Called dbmoretext with a bad size parameter.
SYBETIME	EXTIME	Adaptive Server Enterprise connection timed out.
SYBETMCF	EXPROGRAM	Attempt to install too many custom formats using dbfmtinstall.

エラー名	エラー重大度	エラー・テキスト
SYBETMTD	EXPROGRAM	Attempt to send too much TEXT data using the dbmoretext call.
SYBETPAR	EXPROGRAM	No SYBTEXT or SYBIMAGE parameters were defined.
SYBETPTN	EXUSER	Syntax error:Only two periods are permitted in table names.
SYBETRAC	EXINFO	Attempted to turn off a trace flag that was not on.
SYBETRAN	EXINFO	DBPROCESS is being used for another transaction.
SYBETRAS	EXINFO	DB-Library internal error:Trace structure not found.
SYBETRSN	EXINFO	Bad *numbytes* parameter passed to dbtracestring.
SYBETSIT	EXINFO	Attempt to call dbtsput with an invalid timestamp.
SYBETTS	EXUSER	The table which bulk copy is attempting to copy to a host file is shorter than the number of rows which bulk copy was instructed to skip.
SYBETYPE	EXINFO	Invalid argument type given to Hyper/DB-Library.
SYBEUCPT	EXUSER	Unrecognized custom-format parameter-type encountered in dbstrbuild.
SYBEUCRR	EXCONSISTENCY	Internal software error:Unknown connection result reported by dbpasswd.
SYBEUDTY	EXCONSISTENCY	Unknown datatype encountered.
SYBEUFDS	EXUSER	Unrecognized format encountered in dbstrbuild.
SYBEUFDT	EXCONSISTENCY	Unknown fixed-length datatype encountered.
SYBEUHST	EXUSER	Unknown host machine name.
SYBEUMSG	EXCOMM	Unknown message-id in MSG datastream.
SYBEUNAM	EXFATAL	Unable to get current user name from operating system.

エラー名	エラー重大度	エラー・テキスト
SYBEUNOP	EXNONFATAL	Unknown option passed to dbsetopt.
SYBEUNT	EXUSER	Unknown network type found in interface file.
SYBEURCI	EXRESOURCE	International Release:Unable to read copyright information from the DB-Library localization file.
SYBEUREI	EXRESOURCE	International Release:Unable to read error information from the DB-Library localization file.
SYBEUREM	EXRESOURCE	International Release:Unable to read error mnemonic from the DB-Library localization file.
SYBEURES	EXRESOURCE	International Release:Unable to read error string from the DB-Library localization file.
SYBEURMI	EXRESOURCE	International Release:Unable to read money-format information from the DB-Library localization file.
SYBEUSCT	EXCOMM	Unable to set communications timer.
SYBEUTDS	EXCOMM	Unrecognized TDS version received from the server.
SYBEUVBF	EXPROGRAM	Attempt to read an unknown version of bcp format file.
SYBEUVDT	EXCONSISTENCY	Unknown variable-length datatype encountered.
SYBEVDPT	EXUSER	For bulk copy, all variable-length data must have either a length-prefix or a terminator specified.
SYBEWAID	EXCONSISTENCY	DB-Library internal error:ALTFMT following ALTNAME has wrong id.
SYBEWRIT	EXCOMM	Write to the server failed.
SYBEXOCI	EXNONFATAL	International Release:A character-set translation overflowed its destination buffer while using bcp to copy data from a host-file to the server.

エラー名	エラー重大度	エラー・テキスト
SYBEXTDN	EXPROGRAM	Warning: The *xlt_todisp* parameter to dbfree_xlate was NULL. The space associated with the xlt_tosrv parameter has been freed.
SYBEXTN	EXPROGRAM	The *xlt_tosrv* and *xlt_todisp* parameters to dbfree_xlate were NULL.
SYBEXTSN	EXPROGRAM	Warning: The *xlt_tosrv* parameter to dbfree_xlate was NULL. The space associated with the *xlt_todisp* parameter has been freed.
SYBEZTXT	EXINFO	Attempt to send zero length text or image to dataserver using dbwritetext.
UNUSED	EXINFO	This error number is unused.

エラー重大度

表 2-33 は、各エラー重大度の記号値の意味を示します。

表 2-33：エラー重大度

エラー重大度	数値	説明
EXINFO	1	情報。エラーではない。
EXUSER	2	ユーザ・エラー。
EXNONFATAL	3	致命的なエラーではない。
EXCONVERSION	4	DB-Library のデータ変換エラー。
EXSERVER	5	サーバがエラー・フラグを返した。
EXTIME	6	サーバからの応答を待っている間にタイムアウトになった。DBPROCESS はまだ有効。
EXPROGRAM	7	ユーザ・プログラムの中にコード・エラーがあった。
EXRESOURCE	8	リソースを使い果たした。DBPROCESS のステータスが dead の可能性がある。
EXCOMM	9	サーバとの通信で障害が発生した。DBPROCESS のステータスが dead の可能性がある。
EXFATAL	10	致命的なエラー。DBPROCESS のステータスは dead である。
EXCONSISTENCY	11	内部ソフトウェア・エラー。Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせる。

参照

DBDEAD、dberrhandle

オプション

説明

すべての DB-Library オプションのリスト。

構文

```
#include <sybfront.h>

#include <sybdb.h>
```

使用法

- dbsetopt と dbclropt を使用してオプションの設定やクリアを行うときは、次に示す、*sybdb.h* で定義されている定数を使用します。すべてのオプションは、デフォルトではオフになっています。次に示すオプションが使用可能です。
- DBARITHABORT ― このオプションが設定されている場合は、クエリ実行中に算術例外が発生すると、サーバはそのクエリをアボートします。

- DBARITHIGNORE ― このオプションが設定されている場合は、クエリ実行中に算術例外が起きたときに、サーバは選択された値または更新された値を NULL 値に置き換えます。Adaptive Server Enterprise は警告メッセージを返しません。DBARITHABORT または DBARITHIGNORE のどちらも設定されていない場合は、Adaptive Server Enterprise は NULL 値による置き換えを行い、クエリの実行完了後に警告メッセージを出力します。
- DBAUTH ― このオプションは、システムの管理権限レベルを設定します。レベルには、「sa」、「sso」、「oper」、「dbcc_edit」があります。これらのレベルの詳細については、『ASE リファレンス・マニュアル』を参照してください。
- DBBUFFER ― このオプションを使用すると、結果ローをバッファに格納し、dbgetrow 関数を使用してランダムな順序で結果ローにアクセスできます。このオプションは DB-Library がローカルに扱うもので、サーバ・オプションではありません。このオプションを設定するときは、バッファに格納するローの数をパラメータとして指定してください。バッファに格納するローの数として 0 を指定すると、バッファはデフォルト・サイズ (現時点では 1000 ロー) に設定されます。

  アプリケーションが dbclropt を呼び出して DBBUFFER オプションをクリアすると、DB-Library はそのロー・バッファに関連付けられているメモリを解放します。
- DBCHAINXACTS ― このオプションは、連鎖トランザクションまたは非連鎖トランザクションの動作を選択するときに使用します。

  連鎖トランザクションとは、データを変更または検索する個々の SQL 文が、複数文トランザクションを暗黙的に開始するという意味です。delete 文、insert 文、open 文、fetch 文、select 文、update 文はいずれも暗黙的にトランザクションを開始します。トランザクションを終了するには、commit 文または rollback 文を明示的に実行する必要があります。連鎖モードを使用すると、ANSI SQL 互換となります。

  非連鎖トランザクションとは、データを変更または検索する個々の SQL 文が暗黙的に別のトランザクションになるという意味です。複数文のトランザクションを定義するには、begin transaction 文と、commit 文または rollback 文を明示的に指定する必要があります。

  このオプションはデフォルトではオフ (非連鎖) です。このオプションはすべてのクエリの動作に影響するので、連鎖モードで動作するアプリケーションは、接続がオープンされた直後にこのオプションをオンにする必要があります。

- DBDATEFIRST ― 週の最初の曜日を 1 ～ 7 の範囲の数に設定します。us_english のデフォルトは 1 (日曜日) です。
- DBDATEFORMAT ― datetime または smalldatetime の入力に使用する日付要素 (月／日／年) の順序を設定します。有効な引数は、「mdy」、「dmy」、「ymd」、「ydm」、「myd」、「dym」 です。us_english のデフォルトは「mdy」です。

  ロー・バッファリングを利用すると、指定した数のサーバ結果ローをプログラムのメモリ内に保持できます。ロー・バッファリングを行わない場合は、新しい dbnextrow 呼び出しによって生成される結果ローは、直前の結果ローの内容を上書きします。したがって、ロー・バッファリングは、プログラムでランダムな順序で結果ローにアクセスする場合に便利です。ただし、バッファ内のローごとに割り付けおよび解放を行う必要があるので、メモリおよびパフォーマンスへの影響があります。したがって、この機能は必要な場合にだけ使用してください。特に、dbgetrow または dbsetrow を呼び出す場合にだけ DBBUFFER オプションをオンにしてください。ロー・バッファリングはネットワーク・バッファリングとは無関係であり、まったく別の問題である点に注意してください。(dbgetrow、dbnextrow、dbclrbuf のリファレンス・ページを参照してください)。
- DBFIPSFLAG ― このオプションを指定すると、標準でない SQL コマンドに対してサーバ側でフラグが設定されます。このオプションは、デフォルトではオフになっています。
- DBISOLATION ― このオプションは、トランザクションの独立性レベルを指定するときに使用します。指定できるレベルは 1 と 3 です。デフォルトのレベルは 1 です。レベルを 3 に設定すると、トランザクション内の select クエリで指定されたテーブルのすべてのページが、そのトランザクションが終了するまでロックされます。
- DBNATLANG ― DB-Library の国際化オプションです。指定された DBPROCESS (DBPROCESS が指定されていない場合は、オープンしているすべての DBPROCESS) に、特定の各国言語を関連付けます。DBPROCESS に対する各国言語が設定されていない場合は、デフォルトでは英語が使用されます。

アプリケーションのユーザがアドホック・クエリを行うことができるプログラムでは、DBNATLANG に優先する設定を、Transact-SQL の set language コマンドで指定できます。

注意　特定の LOGINREC を使用してオープンした DBPROCESS はいずれも、その LOGINREC に関連付けられている各国言語を使用します。各国言語を LOGINREC に関連付けるには、DBSETLNATLANG マクロを使用してください。

- DBNOAUTOFREE － このオプションを設定すると、明示的に dbfreebuf が呼び出されたときだけコマンド・バッファがクリアされるようになります。DBNOAUTOFREE が設定されていなければ、dbsqlexec または dbsqlsend を呼び出した後の dbcmd または dbfcmd の最初の呼び出し時に、新しいテキストが入力される前にコマンド・バッファが自動的にクリアされます。
- DBNOCOUNT － このオプションを設定すると、個々の SQL 文の影響を受けたローの数に関する情報がサーバから送られなくなります。アプリケーションでこの情報を取得するには、DBCOUNT を呼び出すという方法もあります。
- DBNOEXEC － このオプションが設定されている場合は、サーバはコンパイル段階までクエリを処理しますが、クエリの実行はしません。このオプションは、DBSHOWPLAN とともに使用できます。
- DBOFFSET － このオプションが設定されている場合は、クエリ内の特定の構成体のオフセットがサーバから返されます。DBOFFSET は、特定の構成体を指定するパラメータを受け取ります。このオプションに指定できるパラメータは、「select」、「from」、「table」、「order」、「compute」、「statement」、「procedure」、「execute」、「param」です (「param」は、ストアド・プロシージャのパラメータを指していることに注意してください)。dbsetopt などのルーチンを呼び出すとき、これらのオプション・パラメータは大文字と小文字のどちらで指定してもかまいません。オフセットは、バッチに構文エラーがない場合にだけ返されます。
- DBPARSEONLY － このオプションが設定されている場合は、サーバはクエリの構文のみをチェックし、ホストにエラー・メッセージを返します。DBOFFSET オプションが設定されている場合にエラーがなければ、オフセットが返されます。

- DBPRCOLSEP － カラム・セパレータ文字を変更します。dbprhead、dbprrow、dbsprhead、dbsprline、dbspr1row によってフォーマットされたクエリ結果ローでは、指定された文字列でカラムが区切られます。デフォルトのセパレータは ASCII 0x20 (スペース) です。3 番目のパラメータである文字列には、null ターミネータは必要ありません。dbsetopt を呼び出すとき、使用する文字列の長さを 4 番目のパラメータとして指定します。デフォルト・セパレータに戻すには、長さとして -1 を指定してください。この場合、3 番目のパラメータは無視されます。
- DBPRLINELEN － 1 行の最大文字数を指定します。この値は、dbprhead、dbprrow、dbsprhead、dbsprline、dbspr1row で使用されます。デフォルトは 80 文字です。
- DBPRLINESEP － dbprhead、dbprrow、dbsprhead、dbsprline、dbspr1row で使用されるロー・セパレータ文字を指定します。デフォルト・セパレータは、改行です (ASCII 0x0D または 0x0A で、ホスト・システムによって異なる)。3 番目のパラメータである文字列には、null ターミネータは必要ありません。dbsetopt を呼び出すとき、文字列の長さを 4 番目のパラメータとして指定します。デフォルト・セパレータに戻すには、長さとして -1 を指定してください。この場合、3 番目のパラメータは無視されます。
- DBPRPAD － dbprhead、dbprrow、dbsprhead、dbsprline、dbspr1row での結果の出力に使用される埋め込み文字を指定します。埋め込みが行われるようにするには、dbsetopt 呼び出しの 4 番目のパラメータとして DBPADON を指定してください。dbsetopt を呼び出すときの 3 番目のパラメータとして埋め込み文字の指定もできます。文字を指定しない場合は、ASCII 文字 0x20 (スペース) が使用されます。埋め込みをオフにするには、4 番目のパラメータとして DBPADOFF を指定して dbsetopt を呼び出してください。埋め込みをオフにするときは、3 番目のパラメータは無視されます。
- DBROWCOUNT － このオプションが 0 より大きい値に設定されている場合は、サーバは、select 文に対して返す通常ローの数、update 文または delete 文の影響を受けるテーブル・ローの数を制限します。select 文が返す計算ローの数は制限しません。

  DBROWCOUNT の動作は、他のオプションとは多少異なります。DBROWCOUNT は常にオンに設定され、オフになることはありません。DBROWCOUNT を 0 に設定するとデフォルト設定に戻ります。つまり、select 文によって生成されたローはすべて返されます。したがって、DBROWCOUNT をオフにするには、カウントに 0 を指定して DBROWCOUNT をオンにします。

- DBSHOWPLAN ― このオプションが設定されている場合は、サーバはコンパイル後に処理プランの説明を生成し、クエリの実行を続けます。
- DBSTAT ― このオプションは、個々のクエリの後で、パフォーマンス情報 (CPU 時間、経過時間、I/O など) をホストにいつ返すかを決定します。DBSTAT には、2 つのパラメータのいずれかを指定します。Adaptive Server Enterprise の内部 I/O に関する統計情報の場合は「io」を使用し、Adaptive Server Enterprise の解析、コンパイル、実行に要する時間の情報の場合は「time」を使用します。DB-Library は、情報メッセージの形式でこれらの統計情報を受け取ります。アプリケーション・プログラムは、ユーザ提供のメッセージ・ハンドラによってこの情報にアクセスできます。
- DBSTORPROCID ― このオプションが設定されている場合は、サーバは、ストアド・プロシージャが生成したローを送信する前に、そのストアド・プロシージャの ID をホストに送信します。
- DBTEXTLIMIT ― このオプションが設定されている場合は、返される text 値または image 値のサイズを DB-Library が制限します。このオプションを設定するときは、パラメータとして、プログラムで処理できる text 値または image 値の最大長をバイト単位で指定します。text 値または image 値のうち、この限界を超えた部分も DB-Library は読み込みますが、無視します。デフォルトでは、DB-Library はサーバから送信されたデータをすべて読み込んで返します。このデフォルト動作に戻すには、DBTEXTLIMIT を 1 より小さい値に設定します。非常に大きな text 値の場合には、ネットワークを通して text 値全体が返されるのに時間がかかることがあります。このような余分な text をサーバが送信しなくて済むようにするには、DBTEXTLIMIT の代わりに DBTEXTSIZE オプションを使用してください。
- DBTEXTSIZE ― このオプションは、サーバのグローバル変数である *@@textsize* の値を変更します。この変数は、サーバが返す text 値または image 値のサイズを制限するものです。このオプションを設定するときは、パラメータとして、サーバが返す text 値または image 値の最大長をバイト単位で指定します。*@@textsize* のデフォルト値は 32,768 バイトです。アプリケーションのユーザがアドホック・クエリを実行できるプログラムでは、このオプションに優先する値を、Transact-SQL の set textsize コマンドを使用してユーザが上書きできることに注意してください。設定する text の制限値をユーザが上書きできないようにするには、DBTEXTSIZE の代わりに DBTEXTLIMIT オプションを使用してください。

- DBBUFFER、DBNOAUTOFREE、DBTEXTLIMIT は DB-Library のオプションです。これらのオプションは DB-Library に反映されますが、サーバには送信されません。その他のオプションは Adaptive Server Enterprise のオプションであり、サーバに送信されます。Adaptive Server Enterprise のオプションは、Transact-SQL コマンドを使用して設定することもできます。
- 前述のとおり、オプションの中には 表 2-34 で示すパラメータを指定できるものがあります。

表 2-34：オプションのパラメータ値

オプション	パラメータの値
DBTEXTSIZE	「0」～「2,147,483,647」
DBOFFSET	「select」、「from」、「table」、「order」、「compute」、「statement」、「procedure」、「execute」、または「param」
DBSTAT	「io」または「time」
DBROWCOUNT	「0」～「2,147,483,647」
DBBUFFER	*int* データ型の長さが 2 バイトの場合は「0」～「32,767」、4 バイトの場合は「0」～「2,147,483,647」
DBTEXTLIMIT	「0」～「2,147,483,647」

dbsetopt で、前のリストに示したオプションを設定するときは、オプション・パラメータを指定しなければなりません。dbclropt と dbisopt では、DBOFFSET と DBSTAT の場合にのみパラメータを指定する必要があります。これは、DBOFFSET と DBSTAT だけは、同時に複数の設定が存在するオプションであるためです。そのため、これらのオプションをクリアまたはチェックするには詳細な定義が必要になります。

dbsetopt、dbclropt、dbisopt の呼び出しで指定するパラメータは、数値の場合でも常に文字列として渡し、引用符で囲む必要があります。詳細については、dbsetopt のリファレンス・ページを参照してください。

参照 dbclropt、dbisopt、dbsetopt

データ型

説明

DB-Library で使用されるデータ型とデータ型の記号定数。

構文

```
#include <sybfront.h>

#include <sybdb.h>
```

使用法

- サーバのデータ型の記号定数は 表 2-35 に示すとおりです。dbconvert と dbwillconvert では、これらの定数を使用します。また、dbcoltype、dbalttype、および dbrettype の各ルーチンは、これらの型のいずれかを返します。

表 2-35：サーバ・データ型の記号定数

記号定数	内容
SYBDATETIME	datetime 型
SYBDATETIME4	4 バイトの datetime 型
SYBMONEY4	4 バイトの money 型
SYBMONEY	money 型
SYBFLT8	8 バイトの float 型
SYBDECIMAL	decimal 型
SYBNUMERIC	numeric 型
SYBREAL	4 バイトの float 型
SYBINT4	4 バイトの integer 型
SYBINT2	2 バイトの integer 型
SYBINT1	1 バイトの integer 型
SYBIMAGE	image 型
SYBTEXT	text 型
SYBCHAR	char 型
SYBBIT	bit 型
SYBBINARY	binary 型
SYBBOUNDARY	セキュリティ sensitivity_boundary 型 **注意**　プログラム変数の型として DBCHAR を使用する。
SYBSENSITIVITY	セキュリティ sensitivity 型 **注意**　プログラム変数の型として DBCHAR を使用する。

詳細については、Adaptive Server Enterprise の『Transact-SQL ユーザーズ・ガイド』を参照してください。

- 次に、DB-Library 関数で使用する C データ型のリストを示します。これらの型は、プログラム変数、特に dbbind、dbaltbind、dbconvert、および dbdatecrack とともに使用する変数を定義するときに便利です。

```
/* char, text, boundary, and sensitivity types */
 typedef char             DBCHAR;

/* binary and image type */
 typedef unsigned char    DBBINARY;

/* 1-byte integer */
 typedef unsigned char   DBTINYINT;

 /* 2-byte integer */
 typedef short           DBSMALLINT;

 / unsigned 2-byte integer */
 typedef unsigned short  DBUSMALLINT;

 /* 4-byte integer */
 typedef long            DBINT;

 /* 4-byte float type */
 typedef float           DBREAL;

 typedef struct          dbnumeric
 {
      char               precision;
      char               scale;
      unsigned char      val[MAXNUMLEN];
 } DBNUMERIC;

 typedef DBNUMERIC       DBDECIMAL;

 /* 8-byte float type */
 typedef double          DBFLT8;

 /* bit type */
 typedef unsigned char   DBBIT;

 /* SUCCEED or FAIL */
 typedef int             RETCODE;

/* datetime type */
 typedef struct          datetime
 {
      /* number of days since 1/1/1900 */
```

```
    long            dtdays;
    /* 300ths of a second since midnight */
    unsigned long   dttime;
} DBDATETIME;

/* 4-byte datetime type */
typedef struct            datetime4
{
    /* number of days since 1/1/1900 */
    unsigned short  numdays;
    /* number of minutes since midnight */
    unsigned short  nummins;
} DBDATETIME4;

    typedef struct            dbdaterec
    {
    /* 1900 to the future */
    long    dateyear;
    /* 0 - 11 */
    long    datemonth;
    /* 1 - 31 */
    long    datedmonth;
    /* 1 - 366 */
    long    datedyear;
    /* 0 - 6 (day names depend on language */
    long    datedweek;
    /* 0 - 23 */
    long    datehour;
    /* 0 - 59 */
    long    dateminute;
    /* 0 - 59 */
    long    datesecond;
    /* 0 - 997 */
    long    datemsecond;
    /* 0 - 127 -- NOTE:Currently unused.*/
    long    datetzone;
  } DBDATEREC;

/* money type */
typedef struct          money
{
    long            mnyhigh;
    unsigned long   mnylow;
} DBMONEY;

/* 4-byte money type */
```

```
typedef signed long      DBMONEY4;

/* Pascal-type string */
typedef struct           dbvarychar
{
    /* character count */
    DBSMALLINT       len;
    /* non-terminated string */
    DBCHAR           str[DBMAXCHAR];
} DBVARYCHAR;

/* Pascal-type binary array */
typedef struct           dbvarybin
{
    /* byte count */
    DBSMALLINT       len;
    /* non-terminated array */
    BYTE              array[DBMAXCHAR];
} DBVARYBIN;

/* Used by DB-Library for indicator variables */
typedef      DBSMALLINT     DBINDICATOR;
```

注意　記号定数 SYBBOUNDARY と SYBSENSITIVITY は、プログラム変数型 DBCHAR に対応します。

参照　dbaltbind、dbalttype、dbbind、dbcoltype、dbconvert、dbprtype、dbrettype、dbwillconvert、オプション

第 3 章

バルク・コピー・ルーチン

この章では、DB-Library バルク・コピー・ルーチンについて説明します。

バルク・コピーの概要

バルク・コピーは、データベース・テーブルとプログラム変数またはホスト・ファイルとの間で、データを高速に転送するためのツールです。SQL の insert コマンドと select コマンドの代わりに使用できます。

DB-Library/C バルク・コピー特殊ライブラリは、DB-Library/C アプリケーションでバルク・コピー機能を実行するためのルーチンの集合です。DB-Library/C アプリケーションで、非データベース・アプリケーションとデータを交換する必要がある場合、新しいデータベースにデータをロードする必要がある場合、または、1 つのデータベースから他のデータベースにデータを移動する必要がある場合に、バルク・コピーが役立ちます。

データベースへのデータの転送

クライアントのホスト・マシン上のフラット・ファイル、またはプログラム変数からデータをデータベースにコピーできます。

データベース・テーブルにデータをコピーするときに、SQL insert コマンドの代わりにバルク・コピーを使用する最大の利点はその速さです。また、SQL insert では、データは文字列フォーマットでなければなりませんが、バルク・コピーはネイティブ・データ型を転送できます。

インデックスのないテーブルにデータをコピーするときは、バルク・コピーの「高速バージョン」が使用されます。高速とは、転送中にデータをログに記録しないことを意味します。転送が完了する前にシステムに障害が起きると、データベースには新しいデータは残りません。高速転送はデータベースのリカバリ性に影響を及ぼすため、Adaptive Server Enterprise のオプション select into/bulkcopy がオンになっている場合にだけ使用可能です。このオプションがオフの場合に、ユーザがインデックスのないテーブルにデータをコピーしようとすると、Adaptive Server Enterprise はエラー・メッセージを生成します。

システム管理者は、バルク・コピーが完了したら、今後のリカバリ性を保証するためにデータベースをダンプする必要があります。

インデックスの付いたテーブルにデータをコピーするときは、bcp の低速バージョンが自動的に使用され、ローの挿入がログに記録されます。

DB-Library/C アプリケーションでデータベースにデータをコピーするには、次の基本手順を実行する必要があります。

- dblogin を呼び出して、LOGINREC 構造体を取得します。これは、後で dbopen を呼び出すときに使用します。
- BCP_SETL を呼び出して、データベースへのバルク・コピー・オペレーション用に LOGINREC を設定します。
- dbopen を呼び出して、Adaptive Server Enterprise との接続を確立します。
- bcp_init を呼び出して、バルク・コピー・オペレーションを初期化し、プログラム変数からコピーするかホスト・ファイルからコピーするかを Adaptive Server Enterprise に通知します。データベースにデータをコピーするには、bcp_init の *direction* パラメータに DB_IN を渡します。

ここからは、プログラム変数からデータをコピーするアプリケーションと、ホスト・ファイルからデータをコピーするアプリケーションとでは、異なる手順が必要になります。

DB-Library/C アプリケーションでプログラム変数からデータをコピーするには、前述の基本手順に加えて次の手順を実行する必要があります。

- データベース・カラムにバインドするプログラム変数ごとに 1 回ずつ bcp_bind を呼び出します。
- ループ内でデータのバッチを転送します。
  - 転送するデータ値をプログラム変数に割り当てます。
  - bcp_sendrow を呼び出して、データのローを送信します。

- ローのバッチが送信されてから、bcp_batch を呼び出してローを Adaptive Server Enterprise に保存します。
- すべてのデータが送信されてから、bcp_done を呼び出してバルク・コピー・オペレーションを終了します。

DB-Library/C アプリケーションでホスト・ファイルからデータをコピーするには、前述の基本手順に加えて次の手順を実行する必要があります。

- bcp_control を呼び出してバッチ・サイズを設定し、制御パラメータのデフォルト設定を変更します。
- bcp_columns を呼び出して、ホスト・ファイルにあるカラムの総数を設定します。
- ホスト・ファイルのカラムごとに 1 回ずつ bcp_colfmt を呼び出します。ホスト・ファイルがデータベース・テーブルと完全に一致する場合は、bcp_colfmt を呼び出す必要はありません。
- bcp_exec を呼び出して、コピー・インを開始します。

データベースからフラット・ファイルへのデータの転送

データベースからデータをコピー・アウトするときは、オペレーション・システム (ホスト) ファイルへのコピーだけが可能です。バルク・コピーでは、データベースからプログラム変数にデータを転送することはできません。

データベース・テーブルからホスト・ファイルにデータを転送するときに、SQL select の代わりにバルク・コピーを使用する最大の利点は、出力ファイル・フォーマットを詳しく指定できることです。速度の点では、バルク・コピーと SQL select の間に大きな差はありません。

DB-Library/C アプリケーションでデータベースからデータをコピー・アウトするには、次の手順を実行する必要があります。

1　dblogin を呼び出して、LOGINREC 構造体を取得します。これは、後で dbopen を呼び出すときに使用します。

2　dbopen を呼び出して、Adaptive Server Enterprise との接続を確立します。

3　bcp_init を呼び出して、バルク・コピー・オペレーションを初期化します。データベースからデータをコピー・アウトするには、*direction* に DB_OUT を渡します。

4 bcp_control を呼び出してバッチ・サイズを設定し、制御パラメータのデフォルト設定を変更します。

5 bcp_columns を呼び出して、ホスト・ファイルにあるカラムの総数を設定します。

6 ホスト・ファイルのカラムごとに 1 回ずつ bcp_colfmt を呼び出します。ホスト・ファイルがデータベース・テーブルと完全に一致する場合は、bcp_colfmt を呼び出す必要はありません。

7 bcp_exec を呼び出して、コピー・アウトを開始します。

バルク・コピー・ルーチンのリスト

ルーチン	説明
bcp_batch	送信済みのローを Adaptive Server Enterprise に保存します。
bcp_bind	プログラム変数から Adaptive Server Enterprise テーブルにデータをバインドします。
bcp_colfmt	バルク・コピー用のホスト・ファイルのフォーマットを指定します。
bcp_colfmt_ps	バルク・コピー用のホスト・ファイルのフォーマットを指定します。numeric カラムおよび decimal カラムに対する精度と位取りをサポートします。
bcp_collen	データベースへの現在のバルク・コピーのプログラム変数データ長を設定します。
bcp_colptr	データベースへの現在のバルク・コピーのプログラム変数データ・アドレスを設定します。
bcp_columns	ホスト・ファイルにあるカラムの総数を設定します。
bcp_control	各種制御パラメータのデフォルト設定を変更します。
bcp_done	プログラム変数から Adaptive Server Enterprise へのバルク・コピーを終了します。
bcp_exec	データベース・テーブルとホスト・ファイルの間で、データのバルク・コピーを実行します。
bcp_getl	LOGINREC がバルク・コピー・オペレーション用に設定されているかどうかを判断します。
bcp_init	バルク・コピーを初期化します。

ルーチン	説明
bcp_moretext	text 値または image 値の一部を Adaptive Server Enterprise に送信します。
bcp_options	バルク・コピーのオプションを設定します。
bcp_readfmt	データファイル・フォーマット定義をホスト・ファイルから読み込みます。
bcp_sendrow	プログラム変数から Adaptive Server Enterprise にデータのローを送信します。
BCP_SETL	LOGINREC をデータベースへのバルク・コピー・オペレーション用に設定します。
bcp_setxlate	Adaptive Server Enterprise からデータを取得するとき、または Adaptive Server Enterprise にデータを挿入するときに使用する文字セット変換を指定します。
bcp_writefmt	データファイル・フォーマット定義をホスト・ファイルに書き込みます。

bcp_batch

説明

送信済みのローを Adaptive Server Enterprise に保存します。

構文

```
DBINT bcp_batch(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

戻り値

bcp_batch の最後の呼び出し以降に保存されたローの数を返します。エラーの場合は -1 です。

使用法

- アプリケーションで、bcp_bind と bcp_sendrow を使用してプログラム変数から Adaptive Server Enterprise テーブルにローをバルク・コピーする場合は、bcp_batch または bcp_done を呼び出したときにのみ、ローが Adaptive Server Enterprise に永続的に保存されます。

- bcp_batch は、たとえば *n* ローごとに呼び出すか、着信データの小休止のとき (遠隔計測アプリケーションなどの場合) に呼び出します。これ以外の基準を選択しても、あるいは bcp_batch をまったく呼び出さなくてもかまいません。bcp_batch を呼び出さない場合は、bcp_done を呼び出したときにローが Adaptive Server Enterprise に永続的に保存されます。
- デフォルトでは、Adaptive Server Enterprise は指定されたすべてのローを 1 つのバッチでコピーします。バッチはそれぞれ別の bcp オペレーションとみなされます。各バッチは 1 つの insert トランザクションでコピーされ、そのバッチ内のローのいずれかが拒否された場合は、insert 全体がロールバックされ、bcp は次のバッチの処理に進みます。bcp_batch を使用すると、大きな入力ファイルを、リカバリ可能な小さな単位に分割できます。たとえば、300,000 ローをバルク・コピーするときに、100,000 ローごとに bcp_batch を呼び出すようにすれば、200,000 ローの後に致命的なエラーが発生しても、最初の 2 つのバッチ (200,000 ロー) は Adaptive Server Enterprise へのコピーが完了しています。
- bcp_batch は、実際には 2 つのコマンドをサーバに送信します。最初のコマンドは、ローを永続的に保存することをサーバに指示します。2 つめは、新しいトランザクションの開始をサーバに指示します。ローを保存するコマンドが完了しても、新しいトランザクションを開始するコマンドが完了しないこともありえます。この場合、bcp_batch のエラーの戻り値 -1 は、ローの保存が正常に終了しなかったことを示しているのではありません。どちらが発生したかを調べるには、Adaptive Server Enterprise または DB-Library/C によって生成されたメッセージを参照します。

参照 bcp_bind、bcp_done、bcp_sendrow

bcp_bind

説明

プログラム変数から Adaptive Server Enterprise テーブルにデータをバインドします。

構文

```
RETCODE bcp_bind (dbproc, varaddr, prefixlen, varlen,
        terminator, termlen, type,
        table_column)

DBPROCESS    *dbproc;
BYTE         *varaddr;
int          prefixlen;
DBINT        varlen;
BYTE         *terminator;
int          termlen;
int          type;
int          table_column;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

varaddr

データのコピー元であるプログラム変数のアドレスです。データ型が SYBTEXT または SYBIMAGE の場合は、*varaddr* に NULL を指定できます。*varaddr* が NULL の場合は、text 値と image 値を、bcp_sendrow によって一度に送信するのではなく、bcp_moretext によっていくつかのまとまりに分けて Adaptive Server Enterprise に送信することを示します。

prefixlen

このカラムが持つ長さプレフィクスのバイト単位での長さです。たとえば、C 以外のプログラミング言語では、文字列が 1 バイトの長さプレフィクスとそれに続く文字列データから構成されることがあります。データが長さプレフィクスを持たない場合は、*prefixlen* を 0 に設定してください。

varlen

長さプレフィクスかターミネータ、またはその両方の長さを除いた、プログラム変数のデータの長さです。データが NULL であることを示すには、*varlen* を 0 に設定します。*varlen* を -1 に設定すると、システムはこのパラメータを無視します。

integer などの固定長データ型では、データ型自体がそのデータの長さをシステムに対して示します。したがって、固定長データ型では常に *varlen* を -1 に設定します。ただし、データが NULL の場合は、*varlen* を 0 に設定します。

データ型が char、text、binary、image の場合は、*varlen* には -1、0、または正の値を指定します。*varlen* が -1 の場合、システムは、長さプレフィクスまたはターミネータ・シーケンスを使用してその長さを決定します (両方とも指定した場合は、コピーされるデータ量が最小になる方が使用されます)。*varlen* が -1 で、プレフィクスの長さもターミネータ・シーケンスも指定しない場合は、エラー・メッセージが返されます。*varlen* が 0 の場合、システムはそのデータが NULL であるとみなします。*varlen* が正の値の場合、システムは *varlen* をそのデータ長として使用します。ただし、*varlen* に正の値を指定して、プレフィクスの長さとターミネータ・シーケンスの両方またはいずれかを指定した場合は、コピーするデータの量が最小になるようにデータ長が決定されます。

terminator

このプログラム変数の終わりを示すバイト・パターンがある場合に、そのパターンを指すポインタを指定します。たとえば、C の文字列には通常 1 バイトのターミネータがあり、その値は 0 です。変数にターミネータがない場合は、*terminator* を NULL に設定してください。

プログラム変数ターミネータとして C の NULL ターミネータを指定する場合に最も簡単な方法は、*terminator* として空文字列 ("") を使用し、*termlen* を 1 に設定することです。これは、NULL ターミネータが 1 バイトで構成されているためです。たとえば、「例」の項にある 2 番目の bcp_bind 呼び出しでは、プログラム変数ターミネータとして 2 つのタブを使用しています。C の NULL ターミネータを代わりに使用するには、次のように書き換えます。

```
bcp_bind (dbproc, co_name, 0, -1, "", 1, 0, 2)
```

termlen

このプログラム変数のターミネータの長さです。変数にターミネータがない場合は、*termlen* を 0 に設定してください。

type
プログラム変数のデータ型を Adaptive Server Enterprise データ型として指定します。プログラム変数のデータはデータベース・カラムの型に自動的に変換されます。このパラメータが 0 の場合、変換は実行されません。サポートされている変換のリストについては、dbconvert のリファレンス・ページを参照してください。このページには、Adaptive Server Enterprise データ型のリストも含まれています。

table_column
データのコピー先であるデータベース・テーブルのカラムです。カラム番号は 1 から始まります。

戻り値

SUCCEED または FAIL。

例

- 次のコードは、bcp_bind の例です。

```
LOGINREC        *login;
DBPROCESS       *dbproc;
char            co_name[MAXNAME];
DBINT           co_id;
DBINT           rows_sent;
DBBOOL          more_data;
char            *terminator = "¥t¥t";

 /* Initialize DB-Library.*/
 if (dbinit() == FAIL)
      exit(ERREXIT);

 /* Install error-handler and message-handler.*/
 dberrhandle(err_handler);
 dbmsghandle(msg_handler);

 /* Open a DBPROCESS.*/
 login = dblogin();
 BCP_SETL(login, TRUE);
 dbproc = dbopen(login, NULL);

 /* Initialize bcp.*/
 if (bcp_init(dbproc, "comdb..accounts_info",
      NULL, NULL, DB_IN) == FAIL)
      exit(ERREXIT);

 /* Bind program variables to table columns.*/
 if (bcp_bind(dbproc, &co_id, 0, -1,
      (BYTE *)NULL, 0, 0, 1) == FAIL)
 {
      fprintf(stderr, "bcp_bind, column 1, failed.¥n");
```

```
        exit(ERREXIT);
    }

    if (bcp_bind
        (dbproc, co_name, 0, -1, (BYTE *)terminator,
        strlen(terminator), 0, 2)
            == FAIL)
    {
        fprintf(stderr, "bcp_bind, column 2, failed.¥n");
        exit(ERREXIT);
    }

    while (TRUE)
    {
        /* Process/retrieve program data.*/
        more_data = getdata(&co_id, co_name);

        if (more_data == FALSE)
            break;

         /* Send the data.*/
         if (bcp_sendrow(dbproc) == FAIL)
            exit(ERREXIT);
    }

    /* Terminate the bulk copy operation.*/
    if ((rows_sent = bcp_done(dbproc)) == -1)
        printf("Bulk-copy unsuccessful.¥n");
    else
        printf("%ld rows copied.¥n", rows_sent);
```

使用法

- 最初にデータをホスト・ファイルとして用意したり、SQL insert コマンドを使用したりすることなく、プログラム変数から Adaptive Server Enterprise のテーブルにデータを直接コピーできます。bcp_bind 関数はこれを素早く効率的に行うための方法です。
- この関数や他のバルク・コピー関数を呼び出す前に、bcp_init を呼び出す必要があります。
- コピー先の Adaptive Server Enterprise テーブルのカラムごとに、個別の bcp_bind 呼び出しが必要です。必要な bcp_bind 呼び出しを実行した後に、bcp_sendrow を呼び出して、データのローをプログラム変数から Adaptive Server Enterprise に送信します。コピー先のテーブルは、bcp_init を呼び出すことによって設定されます。
- bcp_collen を呼び出すと、現在のコピー・インの特定のカラムのプログラム変数のデータ長 (*varlen*) を上書きできます。

- Adaptive Server Enterprise によってすでに受信されたローのチェックポイントを実行するには、bcp_batch を呼び出します。たとえば、1,000 ロー挿入されるたびに一度ずつなどの間隔で bcp_batch を呼び出します。
- すべてのローの挿入が終了したときは、bcp_done を呼び出してください。このようにしなければ、エラーになります。
- bcp_bind を使用するときは、bcp_init の呼び出しでのホスト・ファイル名パラメータ *hfile* を NULL にします。また、方向パラメータ *direction* は DB_IN にします。
- プレフィクスの長さは、integer や float などの固定長データ型では使用しないでください。固定長データ型の場合は、データ型からデータの長さを算出できるので、*prefixlen* には 0、*varlen* には -1 を渡してください。ただし、データが NULL の場合は、*varlen* を 0 にしてください。
- bcp_control で指定した制御パラメータ設定は、bcp_bind によるロー転送には影響しません。
- bcp_bind を使用するときは、bcp_columns を呼び出すとエラーになります。

参照　bcp_batch、bcp_colfmt、bcp_collen、bcp_colptr、bcp_columns、bcp_control、bcp_done、bcp_exec、bcp_init、bcp_moretext、bcp_sendrow

bcp_colfmt

説明　バルク・コピー用のホスト・ファイルのフォーマットを指定します。

構文

```
RETCODE    bcp_colfmt (dbproc, host_colnum, host_type,
                       host_prefixlen, host_collen,
                       host_term, host_termlen,
                       table_colnum)

DBPROCESS      *dbproc;
int            host_colnum;
int            host_type;
int            host_prefixlen;
DBINT          host_collen;
BYTE           *host_term;
int            host_termlen;
int            table_colnum;
```

パラメータ

dbproc
特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

host_colnum
フォーマットを指定するホスト・ファイルのカラムです。最初のカラムの番号は 1 です。

host_type
ホスト・ファイル内のこのカラムのデータ型を Adaptive Server Enterprise データ型として指定します。データベース・テーブルの対応するカラム (*table_colnum*) のデータ型と異なる場合は、自動的に変換が実行されます。可能なデータ変換の表については、dbconvert のリファレンス・ページを参照してください。このページには、Adaptive Server Enterprise データ型のリストも含まれています。

データベース・テーブルの対応するカラム (*table_colnum*) と同じデータ型を指定する場合は、このパラメータを 0 に設定してください。

注意 bcp_colfmt は、numeric 型と decimal 型に対する精度と位取りをサポートしていません。numeric または decimal のホスト・カラムのフォーマットを設定するときは、デフォルトの精度と位取りである 18 と 0 が使用されます。別の精度と位取りを指定するには、bcp_colfmt_ps を呼び出します。

---

host_prefixlen

ホスト・ファイル内の、このカラムの長さプレフィクスの長さです。有効なプレフィクスの長さは、1 バイト、2 バイト、および 4 バイトです。長さプレフィクスが使用されないようにするには、このパラメータを 0 に設定してください。長さプレフィクスを使用するかどうかの判断を bcp に任せるには、このパラメータを -1 に設定してください。この場合、データベース・カラム長が可変であれば、bcp は必要な長さの長さプレフィクスを使用します。

ホスト・ファイル・カラム長の指定に複数の方法を使用する場合 (長さプレフィクスと最大カラム長、または長さプレフィクスとターミネータ・シーケンスなど)、bcp は、これらの方法をすべて調べ、コピーするデータの量が最小になる方法を使用します。

長さプレフィクスの利点の 1 つに、ホスト・ファイルでの null データ値を簡単に指定できることがあります。たとえば、4 バイトの整数カラムに 1 バイトの長さプレフィクスがあるとします。通常、その長さプレフィクスの内容は、4 バイトの値が後に続くことを示す値 4 です。ただし、カラムの値が NULL の場合は、長さプレフィクスを 0 に設定し、プレフィクスに続くカラムが 0 バイトであることを示せます。

host_collen

長さプレフィクスかターミネータ、またはその両方の長さを除いた、ホスト・ファイル内のこのカラムのデータの最大長です。データが NULL であることを示すには、*host_collen* を 0 に設定します。*host_collen* を -1 に設定すると、システムはこのパラメータを無視します (デフォルト最大長はありません)。

integer などの固定長データ型では、null 値の特殊な場合を除いて、そのデータの長さは一定です。したがって、固定長データ型では常に *host_collen* を -1 に設定します。ただし、データが null の場合は、*host_collen* を 0 に設定します。

データ型が char、text、binary、image の場合は、*host_collen* には -1、0、または正の値を指定します。*host_collen* が -1 の場合は、長さプレフィクスまたはターミネータ・シーケンスのいずれかを使用してデータの長さが決定されます (両方とも指定した場合は、コピーされるデータ量が最小になる方が使用されます)。*host_collen* が -1 で、プレフィクスの長さもターミネータ・シーケンスも指定しない場合は、エラー・メッセージが返されます。*host_collen* が 0 の場合、システムはそのデータが NULL であるとみなします。*host_collen* が正の値の場合、システムは *host_collen* をその最大データ長として使用します。ただし、*host_collen* に正の値を指定して、プレフィクスの長さかターミネータ・シーケンス、またはその両方を指定した場合は、コピーするデータの量が最小になる方法を使用してデータ長が決定されます。

host_term
このカラムに使用するターミネータ・シーケンスです。このパラメータは、主にデータ型が char、text、binary、image の場合に使用します。それ以外のデータ型はいずれも固定長です。ターミネータが使用されないようにするには、このパラメータを NULL に設定してください。ターミネータを NULL 文字に設定するには、*host_term* を「¥0」に設定します。タブ文字をターミネータにするには、*host_term* を「¥t」に設定します。改行文字をターミネータにするには、*host_term* を「¥n」に設定します。

ホスト・ファイル・カラム長の指定に複数の方法を使用する場合 (ターミネータと長さプレフィクス、またはターミネータと最大カラム長など)、bcp は、これらの方法をすべて調べ、コピーするデータの量が最小になる方法を使用します。

host_termlen
このカラムに使用するターミネータ・シーケンスのバイト単位での長さです。ターミネータが使用されないようにするには、この値を -1 に設定してください。

table_colnum
データベース・テーブルの対応するカラムです。この値が 0 の場合、このカラムはコピーされません。最初のカラムがカラム 1 です。

戻り値

SUCCEED または FAIL。

使用法

- bcp_colfmt を使用すると、バルク・コピーのホスト・ファイル・フォーマットを指定することができます。バルク・コピーを行うためのフォーマットには、次の要素が含まれます。
  - ホスト・ファイル・カラムからデータベース・カラムへのマッピング

- 各ホスト・ファイル・カラムのデータ型
- 各カラムの長さプレフィクス (省略可能) の長さ
- ホスト・ファイル・カラムのデータの最大長
- 各カラムの終了バイト・シーケンス (省略可能)
- この終了バイト・シーケンス (省略可能) の長さ

- bcp_colfmt の呼び出しは、それぞれ 1 つのホスト・ファイル・カラムのフォーマットを指定します。たとえば、5 つのカラムを持つテーブルがあり、そのうち 3 つのカラムのデフォルト設定を変更する場合は、まず bcp_columns(*dbproc*, 5) を呼び出し、次に bcp_colfmt を 5 回呼び出します。このうちの 3 つの呼び出しで、カスタム・フォーマットを設定します。残りの 2 つの呼び出しでは、*host_type* を 0 に設定し、*host_prefixlen*、*host_collen*、および *host_termlen* の各パラメータを -1 に設定します。この結果、3 つのカラムはカスタム・フォーマットで、2 つのカラムはデフォルト・フォーマットで、5 つのカラムがすべてコピーされます。
- bcp_columns は、bcp_colfmt 呼び出しの前に呼び出さなければなりません。
- デフォルト・フォーマットを使用するカラムや省略されるカラムがあるかどうかに関係なく、ホスト・ファイルのカラムごとに bcp_colfmt を呼び出す必要があります。
- カラムを省略するには、*table_column* パラメータを 0 に設定してください。

参照

bcp_batch、bcp_bind、bcp_colfmt_ps、bcp_collen、bcp_colptr、bcp_columns、bcp_control、bcp_done、bcp_exec、bcp_init、bcp_sendrow

bcp_colfmt_ps

説明

バルク・コピー用のホスト・ファイルのフォーマットを指定します。numeric カラムと decimal カラムに対する精度と位取りをサポートします。

構文

```
RETCODE bcp_colfmt_ps (dbproc, host_colnum, host_type,
                host_prefixlen, host_collen,
                host_term, host_termlen,
                table_colnum, typeinfo)

DBPROCESS    *dbproc;
int          host_colnum;
int          host_type;
int          host_prefixlen;
DBINT        host_collen;
BYTE         *host_term;
int          host_termlen;
int          table_colnum;
DBTYPEINFO   *typeinfo;
```

注意 bcp_colfmt_ps のパラメータは、numeric カラムまたは decimal カラムの精度と位取りに関する情報を指定する *typeinfo* パラメータが追加されている点を除いて、bcp_colfmt のパラメータとまったく同じです。

パラメータ

dbproc
特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

host_colnum
フォーマットを指定するホスト・ファイルのカラムです。最初のカラムの番号は 1 です。

host_type
ホスト・ファイル内のこのカラムのデータ型を Adaptive Server Enterprise データ型として指定します。データベース・テーブルの対応するカラム (*table_colnum*) のデータ型と異なる場合は、自動的に変換が実行されます。可能なデータ変換の表については、dbconvert のリファレンス・ページを参照してください。このページには、Adaptive Server Enterprise データ型のリストも含まれています。

データベース・テーブルの対応するカラム (*table_colnum*) と同じデータ型を指定する場合は、このパラメータを 0 に設定してください。

host_prefixlen

ホスト・ファイル内の、このカラムの長さプレフィクスの長さです。有効なプレフィクスの長さは、1 バイト、2 バイト、および 4 バイトです。長さプレフィクスが使用されないようにするには、このパラメータを 0 に設定してください。長さプレフィクスを使用するかどうかの判断を bcp に任せるには、このパラメータを -1 に設定してください。この場合、データベース・カラム長が可変であれば、bcp は必要な長さの長さプレフィクスを使用します。

ホスト・ファイル・カラム長の指定に複数の方法を使用する場合 (長さプレフィクスと最大カラム長、または長さプレフィクスとターミネータ・シーケンスなど)、bcp は、これらの方法をすべて調べ、コピーするデータの量が最小になる方法を使用します。

長さプレフィクスの利点の 1 つに、ホスト・ファイルでの null データ値を簡単に指定できることがあります。たとえば、4 バイトの整数カラムに 1 バイトの長さプレフィクスがあるとします。通常、その長さプレフィクスの内容は、4 バイトの値が後に続くことを示す値 4 です。ただし、カラムの値が null の場合は、長さプレフィクスを 0 に設定して、プレフィクスに続くカラムが 0 バイトであることを示すことができます。

host_collen

長さプレフィクスかターミネータ、またはその両方の長さを除いた、ホスト・ファイル内のこのカラムのデータの最大長です。データが NULL であることを示すには、*host_collen* を 0 に設定します。*host_collen* を -1 に設定すると、システムはこのパラメータを無視します (デフォルト最大長はありません)。

integer などの固定長データ型では、null 値の特殊な場合を除いて、そのデータの長さは一定です。したがって、固定長データ型の場合は、常に *host_collen* を -1 に設定します。ただし、データが NULL の場合は、*host_collen* を 0 に設定します。

データ型が char、text、binary、image の場合は、*host_collen* には -1、0、または正の値を指定します。*host_collen* が -1 の場合は、長さプレフィクスまたはターミネータ・シーケンスのいずれかを使用してデータの長さが決定されます (両方とも指定した場合は、コピーするデータ量が最小になる方が使用されます)。*host_collen* が -1 で、プレフィクスの長さもターミネータ・シーケンスも指定しない場合は、エラー・メッセージが返されます。*host_collen* が 0 の場合、システムはそのデータが NULL であるとみなします。*host_collen* が正の値の場合、システムは *host_collen* をその最大データ長として使用します。ただし、*host_collen* に正の値を指定して、プレフィクスの長さかターミネータ・シーケンス、またはその両方を指定した場合は、コピーするデータの量が最小になる方法を使用してそのデータ長が決定されます。

host_term

このカラムに使用するターミネータ・シーケンスです。このパラメータは、主にデータ型が char、text、binary、image の場合に使用します。それ以外のデータ型はいずれも固定長です。ターミネータが使用されないようにするには、このパラメータを NULL に設定してください。ターミネータを null 文字に設定するには、*host_term* を「¥0」に設定します。タブ文字をターミネータにするには、*host_term* を「¥t」に設定します。改行文字をターミネータにするには、*host_term* を「¥n」に設定します。

ホスト・ファイル・カラム長の指定に複数の方法を使用する場合 (ターミネータと長さプレフィクス、またはターミネータと最大カラム長など)、bcp は、これらの方法をすべて調べ、コピーするデータの量が最小になる方法を使用します。

host_termlen

このカラムに使用するターミネータ・シーケンスのバイト単位での長さです。ターミネータが使用されないようにするには、この値を -1 に設定してください。

table_colnum

データベース・テーブルの対応するカラムです。この値が 0 の場合、このカラムはコピーされません。最初のカラムがカラム 1 です。

typeinfo

decimal または numeric のホスト・ファイル・カラムの精度と位取りに関する情報が格納されている DBTYPEINFO 構造体を指すポインタです。DBTYPEINFO 構造体の精度と位取りの値を設定してから、bcp_colfmt_ps を呼び出して decimal カラムまたは numeric カラムのホスト・ファイル・フォーマットを指定します。

typeinfo が NULL の場合、bcp_colfmt_ps は bcp_colfmt と同じです。つまり、次のようになります。

- サーバ・カラムの型が numeric または decimal の場合、bcp_colfmt_ps は、このカラムから精度と位取りの値を取り出します。
- サーバ・カラムが numeric と decimal のどちらでもない場合、bcp_colfmt_ps はデフォルト値である精度 18 と位取り 0 を使用します。

host_type が 0 以外で、SYBDECIMAL と SYBNUMERIC のどちらでもない場合は、*typeinfo* は無視されます。

host_type が 0 で、対応するサーバ・カラムが numeric と decimal のどちらでもない場合、*typeinfo* は無視されます。

DBTYPEINFO 構造体は、次のように定義されています。

```
typedef struct typeinfo {
     DBINT    precision;
     DBINT    scale;
 } DBTYPEINFO;
```

precision の有効な値は、1 から 77 までです。*scale* の有効な値は、0 から 77 までです。*scale* の値は *precision* 以下でなければなりません。

戻り値

SUCCEED または FAIL。

使用法

- bcp_colfmt_ps は、numeric データ型と decimal データ型の精度と位取りをサポートする点を除いて、bcp_colfmt と同じです。bcp_colfmt はこれらをサポートしていません。bcp_colfmt を呼び出すことは、*typeinfo* に NULL を指定して bcp_colfmt_ps を呼び出すこととまったく同じです。
- bcp_colfmt_ps を使用すると、バルク・コピーを行うためのホスト・ファイル・フォーマットを指定することができます。バルク・コピーを行うためのフォーマットには、次の要素が含まれます。
  - ホスト・ファイル・カラムからデータベース・カラムへのマッピング

  - 各ホスト・ファイル・カラムのデータ型
  - 各カラムの長さプレフィクス (省略可能) の長さ
  - ホスト・ファイル・カラムのデータの最大長
  - 各カラムの終了バイト・シーケンス (省略可能)
  - この終了バイト・シーケンス (省略可能) の長さ
- bcp_colfmt_ps の呼び出しは、それぞれ 1 つのホスト・ファイル・カラムのフォーマットを指定します。たとえば、5 つのカラムを持つテーブルがあり、そのうち 3 つのカラムのデフォルト設定を変更する場合は、まず bcp_columns(*dbproc*, 5) を呼び出し、次に bcp_colfmt_ps を 5 回呼び出します。このうちの 3 つの呼び出しで、カスタム・フォーマットを設定します。残りの 2 つの呼び出しでは、*host_type* を 0 に設定し、*host_prefixlen*、*host_collen*、および *host_termlen* の各パラメータを -1 に設定します。この結果、3 つのカラムはカスタム・フォーマットで、2 つのカラムはデフォルト・フォーマットで、5 つのカラムがすべてコピーされます。
- bcp_columns は、bcp_colfmt_ps 呼び出しの前に呼び出さなければなりません。
- デフォルト・フォーマットを使用するカラムや省略されるカラムがあるかどうかに関係なく、ホスト・ファイルのカラムごとに bcp_colfmt_ps を呼び出す必要があります。
- カラムを省略するには、*table_column* パラメータを 0 に設定してください。

参照

bcp_batch、bcp_bind、bcp_colfmt、bcp_collen、bcp_colptr、bcp_columns、bcp_control、bcp_done、bcp_exec、bcp_init、bcp_sendrow

bcp_collen

説明

データベースへの現在のバルク・コピーのプログラム変数データ長を設定します。

構文

RETCODE bcp_collen(dbproc, varlen, table_column)

```
DBPROCESS    *dbproc;
DBINT        varlen;
int          table_column;
```

パラメータ

dbproc
特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

varlen
長さプレフィクスやターミネータの長さを除いた、プログラム変数の長さです。データが NULL であることを示すには、*varlen* を 0 に設定します。*varlen* を -1 に設定すると、そのデータが可変長であることを示すことになり、その長さは長さプレフィクスまたはターミネータによって決定されます。長さプレフィクスとターミネータの両方が存在する場合、bcp は、コピーするデータが最小になる方を使用します。

table_column
データのコピー先である Adaptive Server Enterprise テーブルのカラムです。カラム番号は 1 から始まります。

戻り値

SUCCEED または FAIL。

使用法

- bcp_collen 関数を使用すると、bcp_bind を呼び出してコピー・インを実行するときに、特定のカラムのプログラム変数データ長を変更できます。
- プログラム変数データ長の初期値は、bcp_bind を呼び出したときに決定されます。bcp_sendrow の呼び出しと呼び出しの間にプログラム変数データ長を変更する場合に、長さプレフィクスやターミネータを使用しないときは、bcp_collen を呼び出して、その長さを再設定できます。bcp_sendrow への次の呼び出しでは、設定したこの長さが使用されます。
- データ長を変更するテーブル・カラムごとに、別の bcp_collen 呼び出しが必要です。

参照

bcp_bind、bcp_colptr、bcp_sendrow

bcp_colptr

説明

データベースへの現在のバルク・コピーのプログラム変数データ・アドレスを設定します。

構文

RETCODE bcp_colptr(dbproc, colptr, table_column)

```
DBPROCESS    *dbproc;
BYTE         *colptr;
int          table_column;
```

パラメータ

dbproc
特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

colptr
プログラム変数のアドレスです。

table_column
データのコピー先である Adaptive Server Enterprise テーブルのカラムです。カラム番号は 1 から始まります。

戻り値

SUCCEED または FAIL。

使用法

- bcp_colptr 関数を使用すると、bcp_bind を呼び出してコピー・インを実行するときに、特定のカラムのプログラム変数データ・アドレスを変更できます。
- プログラム変数データ・アドレスの初期値は、bcp_bind を呼び出したときに決定されます。bcp_sendrow の呼び出しと呼び出しの間にプログラム変数データ・アドレスを変更する場合は、bcp_colptr を呼び出してデータのアドレスを再設定できます。bcp_sendrow への次の呼び出しでは、設定したこのアドレスのデータが使用されます。
- データ・アドレスを変更するテーブル・カラムごとに、別の bcp_colptr 呼び出しが必要です。

参照

bcp_bind、bcp_collen、bcp_sendrow

bcp_columns

説明

ホスト・ファイルにあるカラムの総数を設定します。

構文

RETCODE bcp_columns(dbproc, host_colcount)

```
DBPROCESS    *dbproc;
int          host_colcount;
```

パラメータ

dbproc
特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

host_colcount
ホスト・ファイル内のカラムの総数です。ホスト・ファイルから Adaptive Server Enterprise テーブルにデータをバルク・コピーするときに、そのホスト・ファイル内のカラムの一部しかコピーしない場合でも、*host_colcount* はホスト・ファイル・カラムの総数に設定してください。

戻り値

SUCCEED または FAIL。

使用法

- この関数は、バルク・コピーで使用するホスト・ファイルにあるカラムの総数を設定します。bcp_init に有効なファイル名を指定して呼び出した後にのみ、このルーチンの呼び出しが可能です。
- デフォルトと異なるホスト・ファイル・フォーマットを使用する場合だけ、このルーチンを呼び出してください。デフォルトのホスト・ファイル・フォーマットについては、bcp_init のリファレンス・ページで説明しています。
- bcp_columns を呼び出した後に、bcp_colfmt を *host_colcount* の数だけ呼び出す必要があります。これは、完全なカスタム・ファイル・フォーマットを定義しなければならないからです。

参照

bcp_colfmt、bcp_init

bcp_control

説明

各種制御パラメータのデフォルト設定を変更します。

構文

```
RETCODE bcp_control(dbproc, field, value)

DBPROCESS    *dbproc;
int          field;
DBINT        value;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

フィールド (field)

制御パラメータの識別子で、次の記号値のうちのいずれかです。

フィールド	説明
BCPMAXERRS	許容されるエラー数。この数を超えると処理を中止する。デフォルトは 10 です。
BCPFIRST	コピーする最初のロー。デフォルトは 1。1 より小さい値を指定すると、このフィールドはデフォルト値の 1 にリセットされる。
BCPLAST	コピーする最後のロー。デフォルトでは、すべてのローがコピーされる。1 より小さい値を指定すると、このフィールドはデフォルト値にリセットされる。
BCPBATCH	バッチ当たりのローの数。デフォルトは 0 で、バルク・コピー全体を 1 つのバッチで行うことを意味する。このフィールドは、ホスト・ファイルから Adaptive Server Enterprise にコピーするときだけ意味がある。

value

対応する制御パラメータの変更後の値です。

戻り値

SUCCEED または FAIL。

使用法

- この関数は、バルク・コピー・オペレーションの各種制御パラメータを設定します。このパラメータには、バルク・コピーをアボートする前の許容エラー数、コピーする最初と最後のローの番号、およびバッチ・サイズがあります。
- これらの制御パラメータが意味を持つのは、ホスト・ファイルと Adaptive Server Enterprise テーブルの間でコピーするときだけです。制御パラメータ設定は、bcp_bind によるロー転送には影響しません。

- デフォルトでは、Adaptive Server Enterprise は指定されたすべてのローを 1 つのバッチでコピーします。バッチはそれぞれ別の bcp オペレーションとみなされます。各バッチは 1 つの insert トランザクションでコピーされ、そのバッチ内のローのいずれかが拒否された場合は、insert 全体がロールバックされ、bcp は次のバッチの処理に進みます。bcp_batch を使用すると、大きな入力ファイルを、リカバリ可能な小さな単位に分割できます。たとえば、300,000 ローを、バッチ・サイズ 100,000 ローでバルク・コピー・インするとき、200,000 ローの後に致命的なエラーが発生しても、最初の 2 つのバッチ (200,000 ロー) は Adaptive Server Enterprise へのコピーが完了しています。バッチを使用していなければ、Adaptive Server Enterprise にはローがまったくコピーされないことになります。
- 次のコードは、bcp_control の例です。

```
LOGINREC      *login;
DBPROCESS     *dbproc;
DBINT         rowsread;

 /* Initialize DB-Library.*/
 if (dbinit() == FAIL)
      exit(ERREXIT);

 /* Install error-handler and message-handler.*/
 dberrhandle(err_handler);
 dbmsghandle(msg_handler);

 /* Open a DBPROCESS.*/
 login = dblogin();
 BCP_SETL(login, TRUE);
 dbproc = dbopen(login, NULL);

 /* Initialize bcp.*/
 if (bcp_init(dbproc, "comdb..address", "address.add",
      "addr.error", DB_IN) == FAIL)
      exit(ERREXIT);

 /* Set the number of rows per batch.*/
 if (bcp_control(dbproc, BCPBATCH, 1000) == FAIL)
 {
      printf("bcp_control failed to set batching behavior.¥n");
      exit(ERREXIT);
 }

 /* Set host column count.*/
```

```
if (bcp_columns(dbproc, 1) == FAIL)
{
     printf("bcp_columns failed.¥n");
     exit(ERREXIT);
}

/* Set the host-file format.*/
if (bcp_colfmt(dbproc, 1, 0, 0, -1, (BYTE *)("¥n"), 1, 1) == FAIL)
{
     printf("bcp_colformat failed.¥n");
     exit(ERREXIT);
}

/* Now, execute the bulk copy. */
if (bcp_exec(dbproc, &rowsread) == FAIL)
{
     printf("Incomplete bulk copy.  Only %ld row%c copied.¥n",
             rowsread, (rowsread == 1) ? ' ': 's');

     exit(ERREXIT);
}
```

参照 bcp_batch、bcp_bind、bcp_colfmt、bcp_collen、bcp_colptr、bcp_columns、bcp_done、bcp_exec、bcp_init

bcp_done

説明 プログラム変数から Adaptive Server Enterprise へのバルク・コピーを終了します。

構文

```
DBINT bcp_done(dbproc)

DBPROCESS*dbproc;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

戻り値 bcp_batch の最後の呼び出し以降に永続的に保存されたローの数を返します。エラーの場合は -1 です。

使用法

bcp_done は、bcp_bind と bcp_sendrow で実行されたバルク・コピーを終了します。これは、bcp_sendrow または bcp_moretext への最後の呼び出しの後で呼び出してください。すべてのデータのコピー・インが完了した後に bcp_done を呼び出さない場合は、予期しないエラーが発生します。

参照

bcp_batch、bcp_bind、bcp_moretext、bcp_sendrow

bcp_exec

説明

データベース・テーブルとホスト・ファイルの間で、データのバルク・コピーを実行します。

構文

```
RETCODE bcp_exec(dbproc, rows_copied)

DBPROCESS    *dbproc;
DBINT        *rows_copied;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

rows_copied

DBINT を指すポインタです。bcp_exec は、コピーが正常に完了したローの数をこの DBINT に格納します。このパラメータを NULL に設定した場合は、ロー数は格納されません bcp_exec。

戻り値

SUCCEED または FAIL。

すべてのローがコピーされた場合は、bcp_exec は SUCCEED を返します。部分的または全体的な障害が発生した場合は、bcp_exec は FAIL を返します。コピーが正常に完了したローの数については、*rows_copied* パラメータを調べてください。

使用法

- このルーチンは、bcp_init の *direction* パラメータの値に応じて、ホスト・ファイルからデータベース・テーブルに、またはその逆にデータをコピーします。
- この関数を呼び出す前に、有効なホスト・ファイル名を指定して bcp_init を呼び出す必要があります。このようにしなければ、エラーになります。
- 次のコードは、bcp_exec の例です。

```
LOGINREC      *login;
DBPROCESS     *dbproc;
DBINT           rowsread;

/* Initialize DB-Library.*/
if (dbinit() == FAIL)
       exit(ERREXIT);

/* Install error-handler and message-handler.*/
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/* Open a DBPROCESS.*/
login = dblogin();
BCP_SETL(login, TRUE);
dbproc = dbopen(login, NULL);

/* Initialize bcp.*/
if (bcp_init(dbproc, "pubs2..authors", "authors.save",
      (BYTE *)NULL, DB_OUT) == FAIL)
      exit(ERREXIT);

/* Now, execute the bulk copy. */
if (bcp_exec(dbproc, &rowsread) == FAIL)
      printf("Incomplete bulk copy.  Only %ld row%c copied.¥n",
              rowsread, (rowsread == 1) ? ' ': 's');
```

参照　bcp_batch、bcp_bind、bcp_colfmt、bcp_collen、bcp_colptr、bcp_columns、bcp_control、bcp_done、bcp_init、bcp_sendrow

bcp_getl

説明　LOGINREC がバルク・コピー・オペレーション用に設定されているかどうかを判断します。

構文　DBBOOL bcp_getl(loginrec)

LOGINREC　*loginrec;

パラメータ　loginrec

dbopen に引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を取得するには、dblogin を呼び出します。

戻り値　「TRUE」または「FALSE」。

使用法

- bcp_getl は、**loginrec* がバルク・コピー・オペレーションに使用可能な場合は「TRUE」を返し、使用可能でない場合は「FALSE」を返します。
- バルク・コピー・イン・オペレーションで DBPROCESS 接続を使用するには、その接続をオープンするときに使用した LOGINREC が、バルク・コピーを使用できるように設定されている必要があります。マクロ BCP_SETL は、バルク・コピーを使用できるように LOGINREC を設定します。デフォルトでは、DBPROCESS のバルク・コピー・オペレーションが無効に設定されています。
- ユーザがアドホック・クエリを行うことができるアプリケーションでは、ユーザが SQL コマンドを使用してバルク・コピー・シーケンスを開始できないようにするために、BCP_SETL の呼び出しを避ける (または、*enable* パラメータを「false」にして呼び出す) ことを検討します。いったん開始したバルク・コピー・シーケンスを、通常の SQL コマンドによって停止することはできません。
- LOGINREC が NULL の場合、bcp_getl は「FALSE」を返します。

参照

bcp_init、BCP_SETL、dblogin、dbopen

bcp_init

説明

バルク・コピーを初期化します。

構文

```
RETCODE bcp_init(dbproc, tblname, hfile, errfile,
        direction)

DBPROCESS    *dbproc;
char         *tblname;
char         *hfile;
char         *errfile;
int          direction;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

tblname
コピー・インまたはコピー・アウトするデータベース・テーブルの名前です。この名前には、データベース名や所有者名が含まれていてもかまいません。たとえば、pubs2.gracie.titles、pubs2.titles、gracie.titles、titles はすべて有効なテーブル名です。

hfile
コピー・インまたはコピー・アウトするホスト・ファイルの名前です。ホスト・ファイルを使用しない場合 (データを変数から直接コピーする場合) は、*hfile* には NULL を指定してください。

errfile
使用するエラー・ファイルの名前です。このエラー・ファイルには、進行メッセージとエラー・メッセージ、および何らかの理由でホスト・ファイルから Adaptive Server Enterprise テーブルにコピーできなかったローのコピーが出力されます。

errfile が NULL の場合、エラー・ファイルは使用されません。

hfile が NULL の場合、*errfile* は無視されます。これは、プログラム変数からバルク・コピーを行うときはエラー・ファイルが必要ないためです。

direction
コピーの方向です。値は DB_IN と DB_OUT のいずれかです。DB_IN は、ホストからデータベース・テーブルへのコピーを示し、DB_OUT は、データベース・テーブルからホスト・ファイルへのコピーを示します。

データベース・テーブルからのバルク・コピー (DB_OUT) を要求するときは、ホスト・ファイル名を指定してください。

戻り値

SUCCEED または FAIL。

使用法

- bcp_init は、フロントエンドと Adaptive Server Enterprise の間でのデータのバルク・コピーに必要な初期化を行います。このルーチンはホスト・ファイルのデフォルトのデータ・フォーマットを設定し、データベース・テーブルの構造を調べます。
- bcp_init は、他のすべてのバルク・コピー関数の前に呼び出す必要があります。このようにしなければ、エラーになります。
- ホスト・ファイルを使用する場合 (前述の「パラメータ」の *hfile* の説明を参照) は、デフォルトのデータ・フォーマットは次のようになります。

- ホスト・ファイルのカラムの順序、型、長さ、数は、データベース・テーブルのカラムの順序、型、数と同一であると見なされます。
- データベース・カラムのデータが固定長の場合は、ホスト・ファイルのデータ・カラムも固定長となります。データベース・カラムのデータが可変長の場合や、NULL 値が可能なカラムの場合は、SYBTEXT データ型と SYBIMAGE データ型ではホスト・ファイルのデータ・カラムに 4 バイトの値がプレフィクスとして付けられ、それ以外のデータ型では 1 バイトのプレフィクスが付けられます。
- ホスト・ファイルのカラム間には、ターミネータはありません。

これらのデフォルトに優先する値を指定するには、bcp_columns と bcp_colfmt. を呼び出します。

- バルク・コピー・ルーチンを使用してデータベース・テーブルにデータをコピーするには、次の条件を満たす必要があります。
  - DBPROCESS 構造体が、バルク・コピーに使用可能でなければなりません。使用可能に設定するには、BCP_SETL を呼び出します。

    ```
    login = dblogin();
    BCP_SETL(login, TRUE);
    ```

  - テーブルにインデックスがない場合は、データベース・オプション select into/bulkcopy が「true」に設定されている必要があります。次の SQL コマンドでこの設定を行います。

    ```
    sp_dboption 'mydb', 'select into/bulkcopy',
    'true'
    ```

  sp_dboption の詳細については、『ASE リファレンス・マニュアル』を参照してください。
- ホスト・ファイルを使用しない場合は、bcp_bind を呼び出して、各カラムのデータ値のフォーマットとメモリ内でのロケーションを指定する必要があります。
- ホスト・ファイルを使用しない場合は、*errfile* は無視されます。プログラム変数からのバルク・コピーを行うときは、エラーが発生すると bcp_sendrow から FAIL が返されるので、エラー・ファイルは不要です。この場合は、バルク・コピーのプログラム変数を調べることによって、エラーを起こしたローの値を特定します。

参照　bcp_batch、bcp_bind、bcp_colfmt、bcp_collen、bcp_colptr、bcp_columns、bcp_control、bcp_done、bcp_exec、bcp_sendrow

bcp_moretext

説明

text 値または image 値の一部を Adaptive Server Enterprise に送信します。

構文

```
RETCODE bcp_moretext(dbproc, size, text)

DBPROCESS    *dbproc;
DBINT        size;
BYTE         *text;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

size

text 値または image 値のうち、Adaptive Server Enterprise に送信する部分のサイズです。Adaptive Server Enterprise に送信する text または image のバイト数が、bcp_bind または bcp_collen の呼び出し時に指定したバイト数を超えている場合は、エラーになります。

text

書き出す text または image の部分を指すポインタです。

戻り値

SUCCEED または FAIL。

使用法

- このルーチンは、大きな SYBTEXT 値または SYBIMAGE 値を多数の小さなまとまりに分けて Adaptive Server Enterprise に送信するときに、bcp_bind および bcp_sendrow とともに使用します。これは、非常に大きなデータ・バッファを割り付けることができないオペレーティング・システムで、特に有効です。

- bcp_bind を呼び出すときに、*type* パラメータに SYBTEXT または SYBIMAGE を指定し、*varaddr* パラメータに NULL 以外の値を指定した場合は、bcp_sendrow は他のすべてのデータ型と同じように、text または image のデータ値全体を送信します。ただし、bcp_bind の *varaddr* パラメータが NULL の場合は、bcp_sendrow は text と image 以外のすべてのカラムを Adaptive Server Enterprise に送信した後で、ただちにアプリケーションに制御を返します。このとき、アプリケーションは、bcp_moretext を繰り返し呼び出して text カラムと image カラムのまとまりを 1 つずつ Adaptive Server Enterprise に送信します。
- 次のコードは、bcp_bind と bcp_sendrow とともに bcp_moretext を使用する方法を示しています。

```
LOGINREC        *login;
DBPROCESS       *dbproc;

DBINT           id = 5;
char            *part1 = "This text value isn't very long,";
char            *part2 = " but it's broken up into three parts";
char            *part3 = " anyhow.";

 /* Initialize DB-Library.*/
 if (dbinit() == FAIL)
       exit(ERREXIT);

 /* Install error handler and message handler.*/
 dberrhandle(err_handler);
 dbmsghandle(msg_handler);

 /* Open a DBPROCESS */
 login = dblogin();
 BCP_SETL(login, TRUE);
 dbproc = dbopen(login, NULL);

 /* Initialize bcp.*/
 if (bcp_init(dbproc, "comdb..articles", (BYTE *)NULL,
      (BYTE *)NULL, DB_IN) == FAIL)
      exit(ERREXIT);

 /* Bind program variables to table columns.*/
 if (bcp_bind(dbproc, (BYTE *)&id, 0, (DBINT)-1,
      (BYTE *)NULL, 0, SYBINT4, 1) == FAIL)
 {
      fprintf(stderr, "bcp_bind, column 1, failed.¥n");
      exit(ERREXIT);
```

```
}

if (bcp_bind
      (dbproc, (BYTE *)NULL, 0,
      (DBINT) (strlen(part1) + strlen(part2) + strlen(part3)),
      (BYTE *)NULL, 0, SYBTEXT, 2)
      == FAIL)
{
      fprintf(stderr, "bcp_bind, column 2, failed.¥n");
      exit(ERREXIT);
}

/*
** Now send this row, with the text value broken into
** three chunks.
*/
if (bcp_sendrow(dbproc) == FAIL)
     exit(ERREXIT);
if (bcp_moretext(dbproc, (DBINT)strlen(part1), part1) == FAIL)
     exit(ERREXIT);
if (bcp_moretext(dbproc, (DBINT)strlen(part2), part2) == FAIL)
     exit(ERREXIT);
if (bcp_moretext(dbproc, (DBINT)strlen(part3), part3) == FAIL)
     exit(ERREXIT);

/* We're all done.*/
bcp_done(dbproc);
dbclose(dbproc);
```

- bcp_moretext を使用してロー内の text カラムまたは image カラムを送信する場合は、そのロー内の他のすべての text カラムおよび image カラムの送信にも bcp_moretext を使用する必要があります。
- ローに text または image のカラムが複数ある場合は、bcp_moretext は、番号が最小の (最も左にある) text カラムまたは image カラムから、カラムの番号順にデータを送信します。
- アプリケーションで多数のデータ・ローを送信するために、通常は、bcp_sendrow と bcp_moretext をループの内側で呼び出します。2 つの text カラムを持つテーブルに対してこれを行う方法を次に示します。

```
while (there are still rows to send)
 {
     bcp_sendrow(...);

     for (all the data in the first text column)
```

```
                bcp_moretext(...);

            for (all the data in the second text column)
                bcp_moretext(...);
        }
```

参照　bcp_bind、bcp_sendrow、dbmoretext、dbwritetext

bcp_options

説明　バルク・コピー・オプションを設定します。

構文　RETCODE bcp_options (dbproc, option, value, valuelen)

```
DBPROCESS    *dbproc;
BYTE         *value;
int          valuelen;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

value

指定したオプションの値を指す汎用の BYTE ポインタです。次の表に示すように、value が何を指すかは *option* によって異なります。

表 3-1 : value の値 (bcp_options)

option	*value
BCPLABELED	DBBOOL の値。sensitivity ラベル付きのバルク・コピーを許可するには、**value* を「true」に設定する。通常のバルク・コピー・オペレーションでは **value* を「false」に設定する。

valuelen

value が指すデータの長さです。value が固定長の項目 (DBBOOL、DBINT など) を指す場合は、*valuelen* に -1 を渡してください。

戻り値　SUCCEED または FAIL。

使用法

- bcp_options はバルク・コピー・オプションを設定します。
- 現在使用できるバルク・コピー・オプションは、BCPLABELED だけです。

参照　bcp_init、bcp_control

bcp_readfmt

説明

データファイル・フォーマット定義をホスト・ファイルから読み込みます。

構文

RETCODE bcp_readfmt(dbproc, filename)

```
DBPROCESS    *dbproc;
char         *filename;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

filename

フォーマット定義が記述されているファイルのフル・ディレクトリ指定です。

戻り値

SUCCEED または FAIL。

使用法

- bcp_readfmt は、データファイル・フォーマット定義をホスト・ファイルから読み込み、bcp_columns と bcp_colfmt を呼び出します。これにより、共通データ・フォーマットを共有する複数のファイルのバルク・コピーを自動化します。
- バルク・コピー・ユーティリティである bcp は、データベース・テーブルと、ユーザ指定フォーマットのホスト・ファイルの間でコピーを実行します。ユーザ指定フォーマットは、bcp によってデータファイル・フォーマット定義ファイルに保存できます。このフォーマットを使用すると、共通フォーマットを持つファイルのバルク・コピーを自動化できます。『Open Client/Server プログラマーズ・ガイド補足』を参照してください。
- bcp_writefmt を呼び出すと、データファイル・フォーマット定義のファイルを作成できます。
- 次のコードは、bcp_readfmt の使用例です。

```
bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", DB_IN);
bcp_readfmt(dbproc, "my_fmtfile");
bcp_exec(dbproc, &rows_copied);
```

参照

bcp_colfmt、bcp_columns、bcp_writefmt

bcp_sendrow

説明

プログラム変数から Adaptive Server Enterprise にデータのローを送信します。

構文

```
RETCODE bcp_sendrow(dbproc)

DBPROCESS    *dbproc;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

戻り値

SUCCEED または FAIL。

使用法

- bcp_sendrow は、プログラム変数からローを構築して Adaptive Server Enterprise に送信します。
- bcp_sendrow を呼び出す前に、bcp_bind を呼び出して、使用するプログラム変数を指定してください。
- bcp_bind を呼び出すときに、*type* パラメータに SYBTEXT または SYBIMAGE を指定し、*varaddr* パラメータに NULL 以外の値を指定した場合は、bcp_sendrow は他のすべてのデータ型と同じように、text または image のデータ値全体を送信します。ただし、bcp_bind の *varaddr* パラメータが NULL の場合は、bcp_sendrow は text と image 以外のすべてのカラムを Adaptive Server Enterprise に送信した後で、ただちにアプリケーションに制御を返します。このとき、アプリケーションは、bcp_moretext を繰り返し呼び出して text カラムと image カラムのまとまりを 1 つずつ Adaptive Server Enterprise に送信します。例については、bcp_moretext のリファレンス・ページを参照してください。
- 最後の bcp_sendrow の呼び出しの後で、bcp_done を呼び出してください。これは、内部のクリーンアップ処理が正しく行われるようにするためです。
- bcp_sendrow を使用してプログラム変数から Adaptive Server Enterprise テーブルにローをバルク・コピーした場合は、ユーザが bcp_batch または bcp_done を呼び出したときにのみ、ローが Adaptive Server Enterprise に永続的に保存されます。

bcp_batch は、たとえば *n* ローごとに 1 回ずつ呼び出すか、着信データの小休止のとき (遠隔計測アプリケーションなどの場合) に呼び出します。これ以外の基準を選択しても、あるいは bcp_batch をまったく呼び出さなくてもかまいません。bcp_batch を呼び出さない場合は、bcp_done を呼び出したときにローが Adaptive Server Enterprise に永続的に保存されます。

参照

bcp_batch、bcp_bind、bcp_colfmt、bcp_collen、bcp_colptr、bcp_columns、bcp_control、bcp_done、bcp_exec、bcp_init、bcp_moretext

BCP_SETL

説明

LOGINREC をデータベースへのバルク・コピー・オペレーション用に設定します。

構文

```
RETCODE BCP_SETL(loginrec, enable)

LOGINREC        *loginrec;
DBBOOL          enable;
```

パラメータ

loginrec

dbopen の引数として渡される LOGINREC 構造体を指すポインタです。LOGINREC 構造体を取得するには、dblogin を呼び出します。

enable

作成される DBPROCESS をバルク・コピー・オペレーションに使用可能とするかどうかを指定するブール値 (「true」または「false」) です。デフォルトでは、DBPROCESS はバルク・コピー・オペレーションが使用できないように設定されます。

戻り値

SUCCEED または FAIL。

使用法

- このマクロは、DBPROCESS 接続がバルク・コピー・オペレーションに使用可能であることを Adaptive Server Enterprise に通知する、LOGINREC 構造体内のフィールドを設定します。設定を有効にするには、実際に DBPROCESS 構造体を割り付けるルーチンである dbopen よりも前にこのマクロを呼び出してください。
- ユーザがアドホック・クエリを行うことができるアプリケーションでは、ユーザが SQL コマンドを使用してバルク・コピー・シーケンスを開始できないようにするために、BCP_SETL の呼び出しを避ける (または、*enable* パラメータを「false」にして呼び出す) ことを検討します。いったん開始したバルク・コピー・シーケンスを、通常の SQL コマンドによって停止することはできません。

- BCP_SETL は、「コピー・イン」オペレーションだけに適用されます。

参照

bcp_init、bcp_getl、dblogin、dbopen、DBSETLAPP、DBSETLHOST、DBSETLPWD、DBSETLUSER

bcp_setxlate

説明

Adaptive Server Enterprise からデータを取得するとき、または Adaptive Server Enterprise にデータを挿入するときに使用する文字セット変換を指定します。

構文

```
RETCODE bcp_setxlate(dbproc, xlt_tosrv, xlt_todisp)

DBPROCESS    *dbproc;
DBXLATE      *xlt_tosrv;
DBXLATE      *xlt_todisp;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

xlt_tosrv

変換構造体を指すポインタです。変換構造体は、dbload_xlate によって割り付けられます。*xlt_tosrv* は、アプリケーション・プログラムから Adaptive Server Enterprise に (「コピー・イン」つまり DB_IN 方向に) データを移動するときに使用する文字セット変換を示します。

xlt_todisp

変換構造体を指すポインタです。変換構造体は、dbload_xlate によって割り付けられます。*xlt_todisp* は、Adaptive Server Enterprise からアプリケーション・プログラムに (「コピー・アウト」つまり DB_OUT 方向に) データを移動するときに使用する文字セット変換を示します。

戻り値

SUCCEED または FAIL。

使用法

- bcp_setxlate は、bcp を使用して Adaptive Server Enterprise とフロントエンド・アプリケーション・プログラムの間で文字データを転送するときに使用される文字セット変換を指定します。

- 指定する文字セット変換は、ユーザの端末でデータの表示や入力に使用されている文字セット変換と同じである必要はありません。この変換は、すぐに表示する必要のない、まったく別の文字セットでデータ・ファイルを読み書きするために使用します。
- 次のコードは、bcp_setxlate の例です。

```
bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", DB_OUT);
bcp_setxlate(dbproc, xlt_tosrv, xlt_todisp);
bcp_columns(dbproc, 3);
bcp_colfmt(dbproc, 1, SYBCHAR, 0, -1, "¥t", 1, 1);
bcp_colfmt(dbproc, 2, SYBCHAR, 0, -1, "¥t", 1, 2);
bcp_colfmt(dbproc, 3, SYBCHAR, 0, -1, "¥n", 1, 3);
bcp_exec(dbproc);
```

参照 dbfree_xlate、dbload_xlate、dbxlate

bcp_writefmt

説明 データファイル・フォーマット定義をホスト・ファイルに書き込みます。

構文 RETCODE bcp_writefmt(dbproc, filename)

```
DBPROCESS    *dbproc;
char         *filename;
```

パラメータ

dbproc

特定のフロントエンド／ Adaptive Server Enterprise プロセスを接続するための DBPROCESS 構造体へのポインタです。この構造体には、DB-Library がフロントエンド／ Adaptive Server Enterprise 間の通信およびデータの管理に使用するすべての情報が含まれます。

filename

フォーマット定義が格納されるファイルのフル・ディレクトリ指定です。

戻り値 SUCCEED または FAIL。

使用法

- bcp_writefmt は、データファイル・フォーマット定義をホスト・ファイルに書き込みます。このフォーマットには、これより前の bcp_columns と bcp_colfmt の呼び出しが反映されています。

- バルク・コピー・ユーティリティである bcp は、データベース・テーブルと、ユーザ指定フォーマットのホスト・ファイルの間でコピーを実行します。ユーザ指定フォーマットは、bcp によって「データファイル・フォーマット定義ファイル」に保存できます。このフォーマットを使用すると、共通フォーマットを持つファイルのバルク・コピーを自動化できます。『Open Client/Server プログラマーズ・ガイド補足』を参照してください。
- フォーマット定義ファイルを読み込むには、bcp_readfmt を使用します。
- 次のコードは、bcp_writefmt の例です。

```
bcp_init(dbproc, "mytable", "bcpdata", "bcperrs", DB_OUT);

bcp_columns(dbproc, 3);
bcp_colfmt(dbproc, 1, SYBCHAR, 0, -1, "¥t", 1, 1);
bcp_colfmt(dbproc, 2, SYBCHAR, 0, -1, "¥t", 1, 2);
bcp_colfmt(dbproc, 3, SYBCHAR, 0, -1, "¥n", 1, 3);

bcp_writefmt(dbproc, "my_fmtfile");
bcp_exec(dbproc, &rows_copied);
```

参照　bcp_colfmt、bcp_columns、bcp_readfmt

第 4 章

2 フェーズ・コミット・サービス

Adaptive Server Enterprise には、2 フェーズ・コミット・サービスの機能があります。2 フェーズ・コミット・サービスを利用すると、クライアント・アプリケーションは、2 つ以上の Adaptive Server Enterprise に分散しているトランザクションを調整できます。

この章では、2 フェーズ・コミット・プロセスと、関連する DB-Library ルーチンについて説明します。

分散トランザクションのプログラミング

2 フェーズ・コミット・サービスによって、アプリケーションは、複数の Adaptive Server Enterprise 間の更新を調整できます。分散トランザクションの初期の実装では、別々のトランザクション (別の Adaptive Server Enterprise にある場合もある) を 1 つのトランザクションのように取り扱っています。コミット・サービスは、1 つの Adaptive Server Enterprise (コミット・サーバ) を中央記録管理サーバとして使用します。アプリケーションは、これを利用して、障害時にトランザクションをコミットするかロールバックするかを判断します。このようにして、2 フェーズ・コミットは、関与するサーバ上のデータベースがすべて更新されるか、まったく更新されないかのどちらかとなることを保証します。

分散トランザクションを実行するには、DB-Library ルーチンを使用して Transact-SQL 文を Adaptive Server Enterprise に渡します。アプリケーション・プログラムは、各サーバとのセッションをオープンし、更新コマンドを発行して、トランザクションをコミットする準備を行います。DB-Library を通じて、アプリケーションは関与するサーバに次のものを発行します。

- アプリケーション、トランザクション、およびコミット・サーバを識別する情報を指定した begin transaction
- Transact-SQL 更新文
- 更新が実行され、サーバがコミットの準備ができたことを示す prepare transaction 文

分散トランザクションに関与するすべてのサーバで更新が実行されると、2 フェーズ・コミットが開始されます。第 1 フェーズでは、すべてのサーバがコミットの準備ができていることに合意します。第 2 フェーズでは、アプリケーションは、トランザクションが完了したこと (つまり、コミットを実行すること) をコミット・サービスに知らせます。次に commit transaction をすべてのサーバに発行し、これによってサーバがコミットを行います。

第 1 フェーズと第 2 フェーズの間でエラーが発生した場合は、すべてのサーバがコミット・サービスとの調整を行い、トランザクションをコミットするか、またはアボートするかを決定します。

注意　2 フェーズ・トランザクション中にエラーが発生したとき、エラーのタイプによっては、2 フェーズ・プロセスが影響を受けたことを示すマーク (infected) が付けられることがあります。これは、プロセスを強制終了するよりも、「infected」というマークを付けておいた方が、後にエラーのリカバリをしやすいためです。Adaptive Server Enterprise がプロセスに「infected」というマークを付けることができるようにするには、コマンド・ラインでフラグ -T3620 を指定して Adaptive Server Enterprise をブートしてください。

コミット・サービスとアプリケーション・プログラム

コミット・サービスの目的は、記録を 1 か所で管理して、アプリケーションがトランザクションのコミットまたはアボートを行う判断に役立てることです。

すべての Adaptive Server Enterprise でコミットする準備が整うと、アプリケーションはコミット・サービスに通知して、トランザクションに「committed」というマークを付けます。このマークを付けた後は、障害が発生してもそのトランザクションはコミットされます。

prepare transaction 文の前に Adaptive Server Enterprise またはアプリケーション・プログラムで障害が発生した場合、その Adaptive Server Enterprise はトランザクションを rollback します。

prepare の後で commit の前に Adaptive Server Enterprise またはアプリケーション・プログラムで障害が発生した場合、その Adaptive Server Enterprise は、コミット・サービスとして機能するサーバと通信して、rollback するか commit するかを問い合わせます。

Adaptive Server Enterprise が、コミット・サービスとして機能するサーバと通信できない場合は、Adaptive Server Enterprise のユーザ・タスク・プロセスに「infected」というマークが付けられます。この時点で、システム管理者は、影響を受けたプロセスをただちに強制終了することも、コミット・サービスとの通信が回復するのを待って、影響を受けたプロセスを強制終了することもできます。

- 影響を受けたプロセスをシステム管理者がただちに強制終了した場合は、2 フェーズ・コミット・プロトコル違反が発生し、2 フェーズ・トランザクションの整合性は保証されません。そのトランザクションに関与しているサーバの一貫性が失われることがあります。
- コミット・サービスとの通信が回復した後に、影響を受けたプロセスをシステム管理者が強制終了した場合は、Adaptive Server Enterprise はコミット・サービスと通信し、トランザクションをローカルにコミットするかどうかを決定します。2 フェーズ・トランザクションの整合性は保証されます。

影響を受けたプロセスをただちに強制終了するかどうかを決定するには、システム管理者は、コミット・サービスの予想ダウン時間、影響を受けたプロセスが保持しているロックの数と重要性、および進行中のトランザクションの複雑さを考慮する必要があります。

アプリケーション・プログラムの役割は、正しい DB-Library ルーチンを使用して、Transact-SQL 文を Adaptive Server Enterprise に正しい順序で配信することです。コミット・サービスの役割は、コミット／ロールバックのステータスを 1 か所で管理することです。Adaptive Server Enterprise は、2 フェーズ・コミット中に障害が発生した場合にだけ、コミット・サービスと通信します。

コミット・サービスは、この記録管理のために、分散トランザクションに使用する DBPROCESS とは別に、自分自身の DBPROCESS を必要とします。ただし、コミット・サービスを処理するサーバは、コミット・サービスが自分自身の DBPROCESS を持つかぎり、トランザクションに関与するサーバの 1 つでもあることに注意してください。トランザクションに関与するすべてのサーバが、1 つの同じサーバということもありえます。

probe プロセス

サーバは、トランザクションをリカバリしなければならない場合に、トランザクションの最後に認識されたステータスを調べるプロセス probe を起動します。トランザクションのステータスをコミット・サービスに返すと、probe プロセスは終了します。probe プロセスは、コミット・サービスが分散トランザクションの進行のチェックに使用するのと同じステータス・チェック・ルーチン stat_xact を使用します。

2 フェーズ・コミット・ルーチン

2 フェーズ・コミット・サービスを構成するルーチンは、次のとおりです。

ルーチン	説明
abort_xact	トランザクションのアボートをコミット・サービスに通知する。
build_xact_string	関与する各 Adaptive Server Enterprise の begin transaction 文および prepare transaction 文で使用される名前文字列を組み立てる。この文字列は、アプリケーションのトランザクション名、コミット・サービス名、および *commid* をコード化する。
close_commit	コミット・サービスとの接続をクローズする。
commit_xact	トランザクションのコミットをコミット・サービスに通知する。

ルーチン	説明
open_commit	コミット・サービスとの接続をオープンする。このルーチンには、セッションを開始するユーザのログイン ID とコミット・サービスの名前を指定する。このルーチンは、後続のコミット・サービス・コールに使用される DBPROCESS 構造体へのポインタを返す。
remove_xact	トランザクションに関与しているサーバのカウント数を減らす。
start_xact	分散トランザクションの開始を記録し、そのトランザクションに関する初期情報 (DBPROCESS id、アプリケーション名、トランザクション名、関与するサイトの数) をコミット・サーバのルックアップ・テーブルに保管する。トランザクションを識別する番号 *commid* を返す。

この他に、進行中のステータスをレポートする 2 つのルーチンがあります。

ルーチン	説明
scan_xact	1 つまたはすべての分散トランザクションのステータスを返す。
stat_xact	分散トランザクションの終了ステータスを返す。

セッションの進行中に、診断ルーチン scan_xact と stat_xact を使用して、コミット・サービスが要求を実行したかどうかを調べます。

scan_xact ルーチンは、コミット・サービス・ルックアップ・テーブル spt_committab を使用します。このテーブルには次の値が保持されています。

- トランザクション ID
- トランザクションを開始した時刻
- ローを最後に更新した時刻
- トランザクションに最初に関与したサーバの数
- まだ完了していないサーバの数
- ステータス :「a」(abort)、「c」(commit)、「b」(begin)
- アプリケーション名
- トランザクション名

2 フェーズ・コミット・ルーチンは、各サーバの master データベース内に作成された内部ストアド・プロシージャ (sp_start_xact など) を呼び出します。*installmaster* スクリプトを実行すると、コミット・サービス・ルックアップ・テーブルとストアド・プロシージャが各サーバの master データベース内に作成されます。これらは、そのサーバがコミット・サーバになったときに使用されます。

コミット・サーバの指定

分散トランザクションに関与する各マシン上の interfaces ファイルに、コミット・サーバのエントリが必要です。コミット・サーバが実際に動作しているマシン上のコミット・サーバのエントリには、『Open Client/Server 設定ガイド』で説明しているように、クエリ・ポートを含む通常ポートが指定されている必要があります。次に例を示します。

```
SERVICE
 master tcp sun-ether rose 2001
 query tcp sun-ether rose 2001
```

分散トランザクションに関与する他のサーバが存在するマシン上では、コミット・サーバのエントリにクエリ・ポートだけを指定する必要があります。

```
SERVICE
query tcp sun-ether rose 2001

SITEA
master tcp sun-ether gaia 2011
query tcp sun-ether gaia 2011
```

コミット・サーバの名前 (この例では「SERVICE」) は、open_commit ルーチン、build_xact_string ルーチンの呼び出しのパラメータとして使用されます。コミット・サーバ名は、トランザクションに関与するすべてのマシン上で同じでなければなりません。この名前にはピリオド (.) やコロン (:) を含めることはできません。

2 フェーズ・コミットのサンプル・プログラム

2 フェーズ・コミット・サービスの例を示すサンプル・プログラムが、DB-Library のサンプル・プログラムに含まれています。これと同じ例を次に示しますが、ここでは、トランザクションのさまざまな時点で発生する異なるタイプの障害のリカバリ方法を示すコメントが追加されています。

```
/*
**    twophase.c
**
**    Demo of Two-Phase Commit Service
**
**    This example uses the two-phase commit service
**    to perform a simultaneous update on two servers.
**    In this example, one of the servers participating
**    in the distributed transaction also functions as
**    the commit service.
**
**    In this particular example, the same update is
**    performed on both servers. You can, however, use
**    the commit server to perform completely different
**    updates on each server.
**
*/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#include "sybdbex.h"

int err_handler();
int msg_handler();

char     cmdbuf[256];
char     xact_string[128];

main()
{

DBPROCESS     *dbproc_server1;
DBPROCESS     *dbproc_server2;
DBPROCESS     *dbproc_commit;
LOGINREC      *login;
int           commid;
```

```
RETCODE       ret_server1;
RETCODE       ret_server2;

/* Initialize DB-Library.*/
if (dbinit() == FAIL)
exit(ERREXIT);


     dberrhandle(err_handler);
     dbmsghandle(msg_handler);

     printf("Demo of Two Phase Commit¥n");

     /* Open connections with the servers and the
     ** commit service.*/
     login = dblogin();
     DBSETLPWD(login, "server_password");
     DBSETLAPP(login, "twophase");

     dbproc_server1 = dbopen (login, "SERVICE");
     dbproc_server2 = dbopen (login, "PRACTICE");
     dbproc_commit = open_commit (login, "SERVICE");

     if (dbproc_server1 == NULL ||
         dbproc_server2 == NULL ||
         dbproc_commit  == NULL)
     {
          printf (" Connections failed!¥n");
          exit(ERREXIT);
     }

     /* Use the "pubs2" database.*/
     sprintf(cmdbuf, "use pubs2");
     dbcmd(dbproc_server1, cmdbuf);
     dbsqlexec(dbproc_server1);
     dbcmd(dbproc_server2, cmdbuf);
     dbsqlexec(dbproc_server2);

     /*
     ** Start the distributed transaction on the
     ** commit service.
     */
     commid = start_xact(dbproc_commit, "demo", "test", 2);
```

注意　ここで、アプリケーションは 2 フェーズ・コミット・トランザクションの *begin* フェーズに入ります。

```
    /* Build the transaction name.*/
     build_xact_string ("test", "SERVICE", commid, xact_string);

     /* Build the first command buffer.*/
     sprintf(cmdbuf, "begin transaction %s", xact_string);

     /* Begin the transactions on the different servers.*/
     dbcmd(dbproc_server1, cmdbuf);
     dbsqlexec(dbproc_server1);
     dbcmd(dbproc_server2, cmdbuf);
     dbsqlexec(dbproc_server2);

     /* Do various updates.*/
     sprintf(cmdbuf, " update titles set price = $1.50 where");
     strcat(cmdbuf, " title_id = 'BU1032'");
     dbcmd(dbproc_server1, cmdbuf);
     ret_server1 = dbsqlexec(dbproc_server1);
     dbcmd(dbproc_server2, cmdbuf);
     ret_server2 = dbsqlexec(dbproc_server2);
```

注意　「プログラム注意事項 1」(501 ページ) を参照してください。

```
    if (ret_server1 == FAIL || ret_server2 == FAIL)
     {
          /* Some part of the transaction failed.*/
          printf(" Transaction aborted -- dbsqlexec failed¥n");
          abortall(dbproc_server1, dbproc_server2,
                   dbproc_commit, commid);
     }

     /* Find out if all servers can commit the transaction.*/
     sprintf(cmdbuf, "prepare transaction");
     dbcmd(dbproc_server1, cmdbuf);
     dbcmd(dbproc_server2, cmdbuf);
     ret_server1 = dbsqlexec(dbproc_server1);
```

注意　「プログラム注意事項 2」(501 ページ) を参照してください。

```
    ret_server2 = dbsqlexec(dbproc_server2);
```

注意　「プログラム注意事項 3」(501 ページ) を参照してください。

```
if (ret_server1 == FAIL || ret_server2 == FAIL)
```

```
 {
      /* One or both of the servers failed to prepare.*/
      printf(" Transaction aborted -- prepare failed¥n");
      abortall(dbproc_server1, dbproc_server2,
               dbproc_commit, commid);
 }
```

注意 「プログラム注意事項 4」(501 ページ) を参照してください。

```
/* Commit the transaction.*/
 if (commit_xact(dbproc_commit, commid) == FAIL)
 {
      /* The commit server failed to record the commit.*/
      printf( " Transaction aborted -- commit_xact failed¥n");
      abortall(dbproc_server1, dbproc_server2,
               dbproc_commit, commid);
      exit(ERREXIT);
 }
```

注意 「プログラム注意事項 5」(503 ページ) を参照してください。

```
/* The transaction has successfully committed.
 ** Inform the servers.
 */
 sprintf(cmdbuf, "commit transaction");
 dbcmd(dbproc_server1, cmdbuf);
 if (dbsqlexec(dbproc_server1) != FAIL)
      remove_xact(dbproc_commit, commid, 1);
```

注意 「プログラム注意事項 6」(503 ページ) を参照してください。

```
dbcmd(dbproc_server2, cmdbuf);
 if (dbsqlexec(dbproc_server2) != FAIL)
 remove_xact(dbproc_commit, commid, 1);
```

注意 「プログラム注意事項 7」(504 ページ) を参照してください。

```
/* Close the connection to the commit server.*/
 close_commit(dbproc_commit);
```

注意 「プログラム注意事項 8」(505 ページ) を参照してください。

```
    printf( "We made it!¥n");
     dbexit();
     exit(STDEXIT);
}

/* Function to abort the distributed transaction.*/

abortall( dbproc_server1, dbproc_server2, dbproc_commit, commid )
DBPROCESS     *dbproc_server1;
DBPROCESS     *dbproc_server2;
DBPROCESS     *dbproc_commit;
int           commid;
{
     /* Some part of the transaction failed.*/

     /* Inform the commit server of the failure.*/
     abort_xact(dbproc_commit, commid);

     /* Roll back the transactions on the different servers.*/
     sprintf(cmdbuf, "rollback transaction");
     dbcmd(dbproc_server1, cmdbuf);
     if (dbsqlexec(dbproc_server1) != FAIL)
          remove_xact(dbproc_commit, commid, 1);
     dbcmd(dbproc_server2, cmdbuf);
     if (dbsqlexec(dbproc_server2) != FAIL)
          remove_xact(dbproc_commit, commid, 1);

     dbexit();
     exit(ERREXIT);
}

/* Message and error handling functions.*/

int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
                servername, procname, line)

DBPROCESS       *dbproc;
DBINT           msgno;
int             msgstate;
int             severity;
char            *msgtext;
char            *servername;
char            *procname;
DBUSMALLINT     line;

{
```

```
     /*   Msg 5701 is just a use database message, so skip it.*/
     if (msgno == 5701)
          return (0);

     /* Print any severity 0 message as is, without extra stuff.*/
     if (severity == 0)
     {
          (void) fprintf (ERR_CH, "%s¥n", msgtext);
          return (0);
     }

     (void) fprintf (ERR_CH, "Msg %ld, Level %d, State %d¥n",
             msgno, severity, msgstate);

     if (strlen(servername) > 0)
          (void) fprintf (ERR_CH, "Server '%s', ", servername);
     if (strlen(procname) > 0)
          (void) fprintf (ERR_CH, "Procedure '%s', ", procname);
     if (line > 0)
          (void) fprintf (ERR_CH, "Line %d", line);

     (void) fprintf (ERR_CH, "¥n¥t%s¥n", msgtext);

     if (severity >= 16)
     {
          (void) fprintf (ERR_CH, "Program Terminated!Fatal¥
                          Adaptive Server Enterprise error.¥n");
          exit(ERREXIT);
     }

     return (0);
}

int err_handler(dbproc, severity, dberr, oserr, dberrstr, oserrstr)
DBPROCESS    *dbproc;
int          severity;
int          dberr;
int          oserr;
char         *dberrstr;
char         *oserrstr;
{
     if ((dbproc == NULL) || (DBDEAD(dbproc)))
          return (INT_EXIT);
     else
     {
          (void) fprintf (ERR_CH, "DB-Library error:¥
```

```
                        ¥n¥t %s¥n", dberrstr);
        if (oserr != DBNOERR)
            (void) fprintf (ERR_CH, "Operating system error:¥
                        ¥n¥t%s¥n", oserrstr);
    }

    return(INT_CANCEL);
}
```

プログラム注意事項

この項では、サンプル・コード内に示した注意事項を説明します。

プログラム注意事項 1

この時点でなんらかの障害が発生した場合、abort_xact を使用してトランザクションをロールバックする処理は、アプリケーションが行います。

プログラム注意事項 2

アプリケーションは、2 フェーズ・コミット・トランザクションの *prepare* ステージに入ります。ただし、コミット・サーバが認識するまで、アプリケーションは *begin* フェーズにあります。

プログラム注意事項 3

この時点でなんらかの障害が発生した場合、abort_xact を使用してトランザクションをロールバックする処理は、アプリケーションが行います。

プログラム注意事項 4

この時点では、次の障害が考えられます。

- コミット・サーバへのアプリケーションのリンク、またはコミット・サーバ自体がダウンする

  この場合、後に続く commit_xact 呼び出しは失敗します。アプリケーションは、abort_xact を使用してトランザクションをロールバックする必要があります。

- 関与しているサーバへのアプリケーションのリンクがダウンする

  この場合、後に続く commit_xact 呼び出しは正常に終了しますが、関与しているサーバに対するアプリケーションの commit transaction コマンドは失敗します。ただし、サーバは、アプリケーションとの接続が切断されたことを認識します。サーバは probe を使用してコミット・サーバと通信し、トランザクションをローカルにコミットするかどうかを決定します。

- 関与しているサーバがダウンする

  この場合、後に続く commit_xact 呼び出しは正常に終了しますが、関与しているサーバに対するアプリケーションの commit transaction コマンドは失敗します。関与しているサーバは、復帰すると、probe を使用してトランザクションをローカルにコミットするかどうかを決定します。

- コミット・サーバへのアプリケーションのリンクと、関与しているサーバへのアプリケーションのリンクの両方がダウンする

  この場合、後に続く commit_xact 呼び出しは失敗します。アプリケーションは、abort_xact を使用してトランザクションをロールバックしますが、関与するサーバと通信することはできません。関与しているサーバは、probe を使用してコミット・サーバと通信します。関与するサーバは、トランザクションがコミット・サーバではコミットされなかったことを認識し、トランザクションをローカルにロールバックします。

- 関与するサーバへのアプリケーションのリンクと、コミット・サーバへの関与するサーバのリンクの両方がダウンする

  この場合、後に続く commit_xact 呼び出しは正常に終了しますが、アプリケーションは、関与するサーバにこれを伝えることができません。アプリケーションとの接続が切断されているとき、関与するサーバは、トランザクションをローカルにコミットするかどうかを決定するために probe を使用してコミット・サーバと通信しようとします。しかし、コミット・サーバとのリンクはダウンしているので、コミット・サーバと通信することはできません。

トランザクションを解決することができないので、関与しているサーバは、ユーザ・タスク・プロセスに、影響を受けたことを示すマーク (infected) を付けます。

コミット・サーバがダウンしている間に、影響を受けたプロセスをシステム管理者が強制終了した場合は、2 フェーズ・コミット・プロトコル違反が発生し、トランザクションの整合性は保証されません。

システム管理者が、コミット・サーバが復帰するのを待って影響を受けたプロセスを強制終了する場合は、そのプロセスを強制終了しようとしたときに probe が自動的に起動されます。probe はコミット・サーバと通信して、関与するサーバにトランザクションをローカルにコミットさせるかどうかを決定します。トランザクションの整合性は保証されます。

プログラム注意事項 5

アプリケーションは、2 フェーズ・コミット・トランザクションの *committed* フェーズに入ります。つまり、probe プロセスがコミット・サーバに問い合わせると、トランザクションをローカルにコミットするという指示を受け取ることになります。これ以降、アプリケーションは、トランザクションのアボートを考慮する必要はありません。

プログラム注意事項 6

サーバ 1 へのアプリケーションのリンクがダウンしているために前述のサーバ 1 への dbsqlexec が失敗した場合、サーバ 1 は probe を使用してコミット・サーバと通信します。probe は、トランザクションがコミット・サーバでコミットされたことを認識し、ローカルにコミットするようにサーバ 1 に指示します。

probe がコミット・サーバと通信できない場合は、Adaptive Server Enterprise のユーザ・タスク・プロセスはサーバ 1 の影響を受けた状態になります。コミット・サーバとの通信が再確立される前に、影響を受けたプロセスをシステム管理者が強制終了した場合、トランザクションはロールバックされます。したがって、2 フェーズ・プロトコル違反が生じ、データベースの一貫性が失われます。システム管理者は、可能なかぎりコミット・サーバとの通信が再確立されるまで待ってから、影響を受けたプロセスを強制終了してください。

サーバ 1 がダウンしているためにサーバ 1 への dbsqlexec が失敗した場合、ローカル・トランザクションはサーバ 1 がリストアされるまで無視されます。リカバリ処理の一部として、サーバ 1 は probe を使用してコミット・サーバと通信します。probe は、トランザクションがコミット・サーバでコミットされたことを認識し、ローカルにコミットするようにサーバ 1 に指示します。

probe がコミット・サーバと通信できない場合、サーバ 1 は、データベースに疑わしいことを示すマーク (suspect) を付けます。コミット・サーバとの通信が再確立された後で、この suspect データベースを再度リカバリします。

プログラム注意事項 7

サーバ 2 へのアプリケーションのリンクがダウンしているために上記のサーバ 2 への dbsqlexec が失敗した場合、サーバ 2 は probe を使用してコミット・サーバと通信します。probe は、トランザクションがコミット・サーバでコミットされたことを認識し、ローカルにコミットするようにサーバ 2 に指示します。

probe がコミット・サーバと通信できない場合は、Adaptive Server Enterprise のユーザ・タスク・プロセスはサーバ 2 の影響を受けた状態になります。コミット・サーバとの通信が再確立される前に、影響を受けたプロセスをシステム管理者が強制終了した場合、トランザクションはロールバックされます。したがって、2 フェーズ・プロトコル違反が生じ、データベースの一貫性が失われます。システム管理者は、可能なかぎりコミット・サーバとの通信が再確立されるまで待ってから、影響を受けたプロセスを強制終了してください。

サーバ 2 がダウンしているためにサーバ 2 への dbsqlexec が失敗した場合、ローカル・トランザクションはサーバ 2 がリストアされるまで無視されます。リカバリ処理の一部として、サーバ 2 は probe を使用してコミット・サーバと通信します。probe は、トランザクションがコミット・サーバでコミットされたことを認識し、ローカルにコミットするようにサーバ 2 に指示します。

probe がコミット・サーバと通信できない場合、サーバ 2 は、データベースに疑わしいことを示すマーク (suspect) を付けます。コミット・サーバとの通信が再確立された後で、この suspect データベースを再度リカバリします。

プログラム注意事項 8

close_commit は、コミット・サーバ上の spt_committab テーブル内で、トランザクションに完了したことを示すマーク (complete) を付けます。close_commit で障害が発生した場合は、トランザクションに「complete」というマークは付けられません。これによる実害はありませんが、この場合、システム管理者は spt_committab を手動で更新します。

abort_xact

説明

分散トランザクションに、アボートされたことを示すマーク (aborted) を付けます。

構文

```
RETCODE abort_xact(connect, commid)

DBPROCESS     *connect;
DBINT         commid;
```

パラメータ

connect

コミット・サービスとの通信に使用する DBPROCESS を指すポインタです。

commid

コミット・サービスがトランザクションを識別するための *commid* です。

戻り値

SUCCEED または FAIL。

使用法

このルーチンは、分散トランザクションのステータスを「begin」から「abort」に変更するようにコミット・サービスに通知します。

参照

commit_xact、remove_xact、scan_xact、start_xact、stat_xact

build_xact_string

説明

分散トランザクションの名前を組み立てます。

構文

```
void build_xact_string(xact_name, service_name,
                commid, result)

char      *xact_name;
char      *service_name;
```

```
DBINT   commid;
char    * result;
```

パラメータ

xact_name
トランザクションのアプリケーション名またはユーザ名です。この名前は名前文字列内ではコード化されますが、コミット・サービスや Adaptive Server Enterprise が使用するのではありません。これは、デバッグのためにトランザクションを識別するものです。

service_name
コミット・サービスと交信するために Adaptive Server Enterprise によって使用される名前で、トランザクションのリカバリに必要です。*service_name* が NULL の場合は、DSCOMMIT が使用されます。

service_name は、コミット・サービスの interfaces ファイル・エントリ名と対応していなければなりません。*service_name* が NULL の場合は、interfaces ファイルに DSCOMMIT のエントリがなければなりません。

commid
コミット・サービスがトランザクションを識別するために使用する番号です。*commid* は、start_xact の呼び出しによって返された番号です。

result
文字列を組み立てるバッファのアドレスです。この領域は、呼び出し側が割り付けなければなりません。

戻り値

なし。

使用法

- このルーチンは、Adaptive Server Enterprise トランザクションの SQL begin transaction と prepare transaction に使用する名前文字列を組み立てます。Adaptive Server Enterprise は、トランザクションをリカバリしなければならない場合に、この名前の中のコード化された情報を使用して、どのコミット・サービスと交信するかと、そのサービスのどのトランザクションについて問い合わせるかを決定します。アプリケーションは、build_xact_string を使用して組み立てた文字列を使用して、SQL begin transaction を発行します。
- build_xact_string によって組み立てられる文字列は、*commid*、*xact_name*、*service_name*、2 つの追加文字、null ターミネータの ASCII 表現を保持できる大きさにしてください。

参照

commit_xact、start_xact

close_commit

説明　コミット・サービスとの接続を終了します。

構文

```
void close_commit(connect)

DBPROCESS    *connect;
```

パラメータ

connect
　open_commit によって最初に返された DBPROCESS 構造体を指すポインタです。

戻り値　なし。

使用法　このルーチンは、dbclose を呼び出してコミット・サービスとの接続を終了します。アプリケーションがコミット・サービスの利用を終了するときは、リソースを解放するために close_commit を呼び出します。

参照　dbclose

commit_xact

説明　分散トランザクションに、コミットされたことを示すマーク (committed) を付けます。

構文

```
RETCODE commit_xact(connect, commid)

DBPROCESS    *connect;
DBINT        commid;
```

パラメータ

connect
　コミット・サービスとの通信に使用する DBPROCESS を指すポインタです。

commid
　コミット・サービスがトランザクションを識別するための *commid* です。

戻り値　SUCCEED または FAIL。

commit_xact が失敗した場合は、必ずトランザクションをロールバックしてください。

使用法　このルーチンは、分散トランザクションのステータスを「begin」から「commit」に変更するようにコミット・サービスに通知します。

参照　abort_xact、remove_xact、scan_xact、start_xact、stat_xact

open_commit

説明

コミット・サービスとの接続を確立します。

構文

```
DBPROCESS *open_commit(login, servername)

LOGINREC        *login;
char            *servername;
```

パラメータ

login

ログイン名、パスワード、希望するオプションなどの、セッションを開始するユーザに関する情報を持つ LOGINREC です。LOGINREC は、DB-Library ルーチン dblogin の呼び出しで事前に取得していなければなりません。呼び出し側は、LOGINREC のフィールドを初期化できます。詳細については、dblogin のリファレンス・ページを参照してください。

servername

コミット・サービスの名前です。たとえば DSCOMMIT_SALESNET などです。*servername* が NULL の場合は、DSCOMMIT が使用されます。この名前にはピリオド (.) やコロン (:) を含めることはできません。

戻り値

後続のコミット・サービス・コールに使用する DBPROCESS 構造体を指すポインタを返します。オープンできなかった場合は NULL です。

使用法

- このルーチンは、dbopen を呼び出してコミット・サービスとの接続を確立します。open_commit は、start_xact、commit_xact、abort_xact、remove_xact、scan_xact などの、他のコミット・サービス・ルーチンよりも前に呼び出す必要があります。コミット・サービスとのセッションをクローズするには、close_commit を呼び出します。
- このルーチンは、コミット・サービスとの通信に使用する DBPROCESS 構造体を返します。この DBPROCESS はコミット・サービス専用とし、分散トランザクションの他の用途に使用しないでください。

参照

dblogin、dbopen

remove_xact

説明

分散トランザクションでアクティブになっているサイトのカウント数を減らします。

構文

RETCODE remove_xact(connect, commid, n)

```
DBPROCESS    *connect;
DBINT        commid;
int          n;
```

パラメータ

connect

コミット・サービスとの通信に使用する DBPROCESS を指すポインタです。

commid

コミット・サービスがトランザクションを識別するための *commid* です。

n

トランザクションから削除するサイトの数です。

戻り値

SUCCEED または FAIL。

使用法

- コミット・サービスは、分散トランザクションに関与しているサイトの数のカウントを保持します。このルーチンは、1 つまたは複数のサイトが、トランザクションに対してローカル・コミットまたはアボートを行ったために、トランザクションに関与しなくなったことをコミット・サービスに通知します。コミット・サービスは、サイトのカウント数を減らすことによって、そのサイトをトランザクションから削除します。
- このカウント数がゼロになった場合は、トランザクション・レコード全体が削除されます。

参照

abort_xact、commit_xact、scan_xact、start_xact、stat_xact

scan_xact

説明

分散トランザクションのコミット・サービス・レコードを出力します。

構文

RETCODE scan_xact(connect, commid)

```
DBPROCESS    *connect;
DBINT        commid;
```

パラメータ

connect
コミット・サービスとの通信に使用する DBPROCESS を指すポインタです。

commid
コミット・サービスがトランザクションを識別するための *commid* です。*commid* が -1 の場合、すべてのコミット・サービス・レコードが表示されます。

戻り値

SUCCEED または FAIL。

使用法

このルーチンは、特定の分散トランザクション、またはコミット・サービスが認識しているすべての分散トランザクションのコミット・サービス・レコードを表示します。

参照

abort_xact、commit_xact、remove_xact、start_xact、stat_xact

start_xact

説明

コミット・サービスを使用して分散トランザクションを開始します。

構文

```
DBINT start_xact(connect, application_name, xact_name,
        site_count)

DBPROCESS    *connect;
char         *application_name;
char         *xact_name;
int          site_count;
```

パラメータ

connect
コミット・サービスとの通信に使用する DBPROCESS を指すポインタです。

application_name
アプリケーションの名前です。アプリケーション名は、アプリケーションの開発者が任意に選択できます。これはコミット・サービスが管理するテーブルに表示されますが、コミット・サービスや Adaptive Server Enterprise リカバリ・システムがこの名前を使用することはありません。

xact_name
トランザクションの名前です。この名前は、コミット・サービスが管理するテーブルに表示されます。また、build_xact_string で組み立てられるトランザクションの名前文字列の一部として指定される必要があります。この名前にはピリオド (.) やコロン (:) を含めることはできません。

site_count
トランザクションに関与するサイトの数です。

戻り値

commid と呼ばれる整数です。この番号は、コミット・サービスへのこれ以降の呼び出しでトランザクションの識別に使用されます。エラーの場合、このルーチンは 0 を返します。

使用法

このルーチンは、コミット・サービスを使用する分散トランザクションの開始を記録します。コミット・サービスの内部にレコードが置かれ、この中に *commid* が含まれています。この *commid* は、これ以降、呼び出し側がコミット・サービスに対するトランザクションの識別に使用する番号です。

参照

abort_xact、build_xact_string、commit_xact、remove_xact、scan_xact、stat_xact

stat_xact

説明

分散トランザクションの現在のステータスを返します。

構文

```
int stat_xact(connect, commid)

DBPROCESS    *connect;
DBINT        commid;
```

パラメータ

connect
コミット・サービスとの通信に使用する DBPROCESS を指すポインタです。

commid
コミット・サービスがトランザクションを識別するための *commid* です。*commid* が -1 の場合、すべてのコミット・サービス・レコードが表示されます。

戻り値

文字コード :「a」(abort)、「b」(begin)、「c」(commit)、「u」(unknown)、または -1 (要求が失敗)

使用法

このルーチンは、指定された分散トランザクションのトランザクション・ステータスを返します。

参照

abort_xact、commit_xact、remove_xact、scan_xact、start_xact

付 録　A

カーソル

この章では、DB-Library のカーソルについて説明します。

カーソルの概要

リレーショナル・データベースはセット指向なので、次のローという概念は存在しません。これは、セット内のローを個別には操作できないことを意味します。カーソル機能は、ディスク上のファイルを読み込んで更新する方法と同様に、結果セットを 1 ローずつ処理することによって、この問題を解決します。DB-Library カーソルは、画面上のカーソルがテキストのブロック内の現在位置を示すのとまったく同じように、結果セット内の現在位置を示します。

DB-Library カーソルは「クライアント側」のカーソルです。これは Adaptive Server Enterprise のカーソルには対応していませんが、ユーザからはサーバ側のカーソルのように見えます。DB-Library カーソルは、キーセットの管理、ローの位置付け、および同時実行制御をクライアント側で透過的に行います。

DB-Library カーソルの機能

DB-Library カーソル・ルーチンは次の機能を備えていますが、多少の制限があります。

- 前方スクロールと後方スクロール (dbcursoropen でキーセットがどのように定義されたかによって異なる)
- 結果セット内の位置指定による直接アクセス

- 位置指定の更新 (結果セットが order by 付きで定義された場合も可能)
- 他のユーザによる変更に対する感度の調整
- オプションによる同時実行の制御

DB-Library カーソルとブラウズ・モードの相違

カーソルを使用して結果セットをスクロールしたり更新したりするときは、ブラウズ・モードよりも制約は少なくなります。カーソルを使用する場合は、ユニーク・インデックスが必要ですが、更新のためのタイムスタンプやデータベースへの別の接続は必要ありません。また、カーソルは、結果セット全体のコピーも作成しません。次の表に、これらの違いを要約します。

表 A-1 : カーソルとブラウズ・モード

項目	カーソル	ブラウズ・モード
ロー・タイムスタンプ	不要	必須
更新のための複数の接続	不要	必要
テーブルの使用	元のテーブルを使用	テーブルのコピーを使用

DB-Library カーソルと Client-Library カーソルの相異

DB-Library カーソルは実際の Adaptive Server Enterprise のカーソルには対応していません。代わりに、dbcursoropen でカーソルを宣言するときに、DB-Library は Adaptive Server Enterprise からキーセットを「暗黙的に」フェッチします。次に、現在のローのキーに基づいて修飾子を構築し、Adaptive Server Enterprise に送ります。サーバはクエリを解析し、結果セットを返します。さらにデータを取り出すために dbcursorfetch が呼び出されると、DB-Library カーソルは別の選択を行わなければならない場合もあります。さらに、Adaptive Server Enterprise は dbcursorfetch が呼び出されるたびにクエリを解析しなければならない場合があります。

Client-Library カーソルは Adaptive Server Enterprise の実際のカーソルに対応します。このため、「ネイティブ・カーソル」と呼ばれることがあります。新しい TDS プロトコルにより、Client-Library はサーバと対話してカーソルを管理することが可能になりました。

Client-Library カーソルは DB-Library カーソルより高速です。これは、SQL コマンドをサーバに送る必要がなく、クエリが何度も再解析されることがないからです。しかし、結果セットがサーバ側に残るため、DB-Library カーソルと同様の同時実行制御のオプションはありません。

2 つのカーソルの相違点を次の表にまとめます。

表 A-2 : DB-Library カーソルと Client-Library カーソルの違い

DB-Library カーソル	Client-Library カーソル
カーソル・ローの位置はクライアントによって定義される。	カーソル・ローの位置はサーバによって定義される。
オプティミスティック同時実行制御を定義できる (ダーティ・リードができる)。	オプティミスティック同時実行制御を定義できない (ダーティ・リードができない)。
後方フェッチができる (dbcursoropen の *scrollopt* に CUR_KEYSET または CUR_DYNAMIC が指定されている場合)。	前方フェッチだけが可能。
dbcursoropen の実行時に指定するフェッチ・バッファのロー数を少なくしないで、ロー・サイズの大きいクエリを行うと、必要なメモリ量が増えることがある。	ロー・サイズに関係なく、余分なメモリを必要としない。
Open Server アプリケーションにアクセスするには、そのアプリケーションが必要な DB-Library ストアド・プロシージャをインストールしていなければならない。	カーソルをサポートするようにコーディングされたリリース 10.0 以降の Open Server アプリケーションにアクセスできる。
パフォーマンスが低い。	パフォーマンスが高い。

変更に対する感度

変更に対する感度に応じて、カーソルは 3 つのカテゴリに分類されます。

- *静的* ― カーソルがオープンされている間は、結果セットの値、順序、メンバシップは変わりません。
- キーセット駆動型 ― 値は変更できますが、結果セットの順序とメンバシップは「オープン時」(カーソルがオープンされたとき) に決定され、固定されます。
- *動的* ― 結果セットの値、順序、メンバシップはすべて変更できます。

静的カーソル

静的カーソルでは、カーソルがオープンされている間は、カーソル所有者も他のユーザも結果セットを変更することはできません。値、メンバシップ、および順序は、カーソルがクローズされるまで固定されます。アプリケーションでは、結果セットのスナップショットを作成するか (更新が行われるとスナップショットと結果セットは異なるものになります)、または、更新を防ぐために結果セット全体をロックします。

カーソル・ルーチンが静的カーソルを直接サポートする必要はありません。静的動作は、次の方法のいずれかによって行うことができます。

- select...into を使用して結果セットのスナップショット・コピーを作成し、そのスナップショット (テンポラリ・テーブル) に対して dbcursoropen を呼び出します。
- select 文にキーワード holdlock を指定して dbcursoropen を呼び出すことにより、結果セットをロックします。ただし、この方法では同時実行性が大幅に低下します。

キーセット駆動型カーソル

キーセット駆動型カーソルでは、結果セット内のローのメンバシップと順序はオープン時に固定されますが、値はカーソル所有者が変更できます。他のユーザによってコミットされた変更も反映されます。変更によってローの順序が変わった場合や、ローがメンバシップの条件を満たさなくなった場合も、カーソルをクローズして再オープンしないかぎり、そのローが消失することも移動することもありません。カーソルがオープンしたままの状態で、削除されたローにアクセスすると、そのローが見つからないことを示す特別なエラー・コードが返されます。キーが更新されたときも、ローは「見つからない」状態になります。

挿入されたデータは反映されませんが、既存のデータに対する変更は、バッファがリフレッシュされたときに反映されます。orderby の有無に関わらず、ユーザは「相対位置」でも「絶対位置」でもローにアクセスすることができます。

相対位置でローにアクセスするには、現在の位置を基準としてカーソルを移動します。たとえば、カーソルの位置がロー 3 のときに、ロー 8 にアクセスするには、現在の位置からロー 5 つ分ジャンプするようカーソルに指示してください。カーソルは、ロー 5 つ分ジャンプしてロー 8 へ移動します。

絶対位置でローにアクセスするには、アクセスするローの番号をカーソルに指示します。たとえば、カーソルの位置がロー 3 のときに、ロー 8 にアクセスするには、ロー 8 にジャンプするようカーソルに指示してください。

動的カーソル

動的カーソルでは、カーソル所有者によるコミットされていない変更と、任意のユーザによってコミットされた変更は、ユーザが次にスクロールしたときに反映されます。変更には、挿入と削除の他に、順序とメンバシップの変更もあります (ローの削除によって間隔があくことはありません)。ローにアクセスするには、結果セット内の相対位置を指定します (絶対位置ではアクセスできません)。動的カーソルでは、order by 句を使用することはできません。

同時実行制御

カーソルは、いくつかのオプションを通して「同時実行アクセス」を制御します。同時実行アクセスとは、複数のユーザが同時に同じデータにアクセスして更新するときに発生します。同時実行アクセスが行われるときは、なんらかの制御がなければ、データの信頼性が失われます。特定の同時実行制御をアクティブにするには、カーソルをオープンするときに次のオプションのいずれかを指定してください。

表 A-3： 同時実行制御オプション

オプション	結果
CUR_READONLY	更新を受け付けない。
CUR_LOCKCC	現在クライアント・バッファにあるローのセットは、ユーザが開始したトランザクションの中でフェッチされたときにロックされる。他のユーザがこのローを更新したり読み込んだりすることはできない。カーソル所有者が行う更新は完了が保証される。 アプリケーションが最初に begin transaction を発行しないかぎり、ローがロックされることはない。ロックは、アプリケーションが commit transaction を発行するまで保持される。次のフェッチが実行されても、ロックは自動的には解放されない。
CUR_OPTCC および CUR_OPTCCVAL	現在バッファにあるローはロックされない。他のユーザは、ローを自由に更新したり読み込んだりできる。

カーソル所有者が発行した更新と他のユーザが発行した更新の間の競合を検出するために、カーソルは、タイムスタンプまたはカラム値を保存して比較します。したがって、オプティミスティック同時実行制御オプション (CUR_OPTCC または CUR_OPTCCVAL) を指定した場合は、他の更新との競合が原因で更新に失敗することがあります。更新に失敗した場合はバッファをリフレッシュしてからリトライするように、アプリケーションを設計してください。

2 つのオプティミスティック同時実行制御オプションは、競合を検出する方法が異なります。

表 A-4： 同時実行競合の検出

オプション	検出の方法
CUR_OPTCC	タイムスタンプ値を基にオプティミスティック同時実行制御を行う。タイムスタンプを使用できる場合はその値を比較する。使用できない場合は、テーブル内の text と image を除くすべてのカラムの値を保存して、前の値と比較する。
CUR_OPTCCVAL	値を基にオプティミスティック同時実行制御を行う。タイムスタンプが使用できるかどうかに関わらず、選択された値を比較する。

DB-Library カーソル関数

次の一覧は、DB-Library カーソル・ルーチンの要約です。

ルーチン	説明
dbcursoropen	カーソルの宣言とオープン、フェッチ・バッファのサイズの指定とキーセットの定義、および同時実行制御オプションの設定を行う。
dbcursorinfo	オープン・カーソル内のカラムとローの数を返す。
dbcursorcolinfo	オープン・カーソル内の指定カラム番号のカラム情報を返す。
dbcursorbind	プログラム変数をカラムに関連付ける。
dbcursorfetch	フェッチ・バッファをスクロールする。
dbcursor	フェッチ・バッファ内のローを更新、削除、挿入、およびリフレッシュする。
dbcursorclose	カーソルをクローズする。

それぞれのルーチンの詳細については、各ルーチンのリファレンス・ページを参照してください。

ロックの保持

Adaptive Server Enterprise トランザクション・モデルの柔軟性を保つために、カーソルは、begin transaction または commit transaction を自動的には発行しません。カーソル・オペレーション中に取得されたロックの持続期間は、完全にアプリケーションの制御下にあります。つまり、dbcursoropen ルーチンまたは dbcursor ルーチンのいずれかで CUR_LOCKCC を使用する場合に、そのロックを有効にするには、begin transaction も発行しなければなりません。

dbcursoropen で CUR_LOCKCC を使用する場合に、現在バッファ内にあるローのロックを保持するには、ローカル・バッファをスクロールする dbcursorfetch の前に commit transaction と begin transaction を発行しなければなりません (ただし、最初の dbcursorfetch は例外で、この前には begin transaction だけを発行します)。

短期間のロック機能を使用するには、begin transaction を発行してから、更新するローを dbcursor の CUR_LOCKCC オプションでロックしてください。更新がそれぞれ独立している場合は、各更新の後で commit transaction を発行してください。同じ画面のデータに対する複数の更新が相互に依存している場合は、その画面がスクロールされるときに commit transaction を発行してください。

繰り返し可能読み出しの一貫性を保つために、dbcursoropen の select 文に holdlock を指定し、最初の dbcursorfetch の前に begin transaction を発行してください。ロックは、データがフェッチされたときに取得され、アプリケーションが commit transaction または rollback transaction を発行するまで保持されます。

繰り返し可能読み出しのカーソルをクローズして再オープンするのと同じ効果を、FETCH_FIRST によって得ることができます。

他の組み合わせも可能です。begin transaction が有効でないかぎりロックは保持されないということに注意してください。begin transaction が有効な間に取得されたロックは、commit transaction または rollback transaction が発行されるまで保持されます。

DB-Library カーソルが使用するストアド・プロシージャ

DB-Library のカーソル・ルーチンは、Adaptive Server Enterprise のカタログ・ストアド・プロシージャを呼び出して、テーブル・フォーマットを検出し、キー・カラムを識別します。

『ASE リファレンス・マニュアル』を参照してください。

付 録 B

DB-Library エラー・メッセージ

20001

記号定数 SYBESYNC

メッセージ `Adaptive Server Enterprise` と同期化されていない状態で読み込みが行われました。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20002

記号定数 SYBEFCON

メッセージ サーバ接続に失敗しました。

考えられる原因 内部 I/O エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン 15.7 ESD #3 より前のバージョン

20003

記号定数 SYBETIME

メッセージ サーバ接続がタイム・アウトしました。

考えられる原因 ネットワーク I/O オペレーションがタイム・アウトになっている。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 dbsettime() を使用して、アプリケーションがサーバからの応答を待機する時間を長くすることができます。

バージョン すべて

20004

記号定数 SYBEREAD

メッセージ サーバからの読み込みに失敗しました。

考えられる原因 内部 I/O エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20005

記号定数 SYBEUFL

メッセージ `DB-Library 内部エラー - 送信バッファ長が壊されています。`

考えられる原因 内部メモリ・エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20006

記号定数　SYBEWRIT

メッセージ　サーバへの書き込みに失敗しました。

考えられる原因　接続が使用できなくなっている。サーバが応答を停止している可能性があります。

アクション / 解決法　接続をいったん閉じて再確立してください。

追加情報

バージョン　すべて

20008

記号定数　SYBESOCK

メッセージ　ソケットをオープンできません。

考えられる原因　内部 I/O エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　15.7 ESD #3 より前のバージョン。

20009

記号定数　SYBECONN

メッセージ　ソケットを接続できません -- サーバが利用できないか、存在しません。

考えられる原因　内部 I/O エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20010

記号定数 SYBEMEM

メッセージ 十分なメモリの割り当てができません。

考えられる原因 ヒープ・メモリを取得できない。

アクション / 解決法 ヒープ・メモリのシステム設定を確認してください。

追加情報

バージョン すべて

20011

記号定数 SYBEDBPS

メッセージ 最大数の DBPROCESS がすでに割り当てられています。

考えられる原因 設定されている DBPROCESS の最大数を超えている。

アクション / 解決法 dbsetmaxprocs() を使用して、上限を上げてください。

追加情報 デフォルト値は 25 です。

バージョン すべて

20012

記号定数 SYBEINTF

メッセージ interfaces ファイル内にサーバ名がありません。

考えられる原因 DSQUERY が正しく設定されていない。

アクション / 解決法 サーバ名が interfaces ファイルに含まれていることを確認してください。

追加情報

バージョン すべて

20013

記号定数　SYBEUHST

メッセージ　ホスト・マシン名がわかりません。

考えられる原因　interfaces ファイルに未知のマシン名がある。

アクション / 解決法　interfaces ファイルを修正してください。

追加情報

バージョン　すべて

20014

記号定数　SYBEPWD

メッセージ　ログインが正しくありません。

考えられる原因　ユーザ名またはパスワードが正しくありません。

アクション / 解決法　正しいユーザ名またはパスワードを指定してください。

追加情報

バージョン　すべて

20015

記号定数　SYBEOPIN

メッセージ　`interfaces` ファイルをオープンできない。

考えられる原因　interfaces ファイルをオープンできない。

アクション / 解決法　interfaces ファイルがあるかないか、または interfaces ファイルのパーミッションを確認してください。

追加情報

バージョン　すべて

20016

記号定数 SYBEINLN

メッセージ `interfaces` ファイル： 予期しない行の終わりです。

考えられる原因 interfaces ファイルに予期しない行の終わり (EOL) がある。

アクション / 解決法 interfaces ファイルのフォーマットが正しいことを確認してください。

追加情報

バージョン すべて

20017

記号定数 SYBESEOF

メッセージ 予期しない行の終わりです。

考えられる原因 サーバが停止している。

アクション / 解決法 サーバのステータスを確認してください。

追加情報

バージョン 15.7 ESD #3 より前のバージョン。

20018

記号定数 SYBESMSG

メッセージ サーバの一般エラー： サーバからのメッセージを確認します。

考えられる原因 サーバ上でエラーが発生している。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20019

記号定数　SYBERPND

Message Text　保留されている結果に対して新しいサーバ・オペレーションを開始しようとしました。

考えられる原因　すべての前の結果がまだ処理されていない状態で dbsqlexec() が呼び出されている。

アクション / 解決法　新しいクエリを実行する前に dbresults() ですべての結果を処理してください。

追加情報

バージョン　すべて

20020

記号定数　SYBEDBPS

メッセージ　サーバからの不正なトークン： データ・ストリーム処理が同期化されていません。

考えられる原因　内部のクライアント / サーバ通信エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20021

記号定数　SYBEITIM

メッセージ　不正な `timeout` 値が指定されました。

考えられる原因　dbsettime() または dbsetlogintime() に渡された値が無効だった。

アクション / 解決法　*seconds* パラメータが正の整数値であることを確認します。

追加情報

バージョン　すべて

20022

記号定数 SYBEOOB

メッセージ 帯域外データをサーバに送信中にエラーが発生しました。

考えられる原因 ネットワークから送信された帯域外データでエラーが発生している。

アクション / 解決法 DBPROCESS をいったんクローズして、オープンしてください。

追加情報 帯域外データが dbcancel() を呼び出した結果として送信されている可能性があります。

バージョン すべて

20023

記号定数 SYBEBTYP

メッセージ `DB-Library` 関数に渡されたバインド・タイプがわかりません。

考えられる原因 dbbind() (またはその他のバインド関数) が *vartype* に未知の値を渡している。

アクション / 解決法 関数ドキュメントから有効なバインド・タイプを確認して、アプリケーション・コーディングを修正してください。

追加情報

バージョン すべて

20024

記号定数 SYBEBNCR

メッセージ 存在しない `compute` ローにユーザ変数をバインドしようとしました。

考えられる原因 結果セットに compute ローが含まれていない。

アクション / 解決法 dbaltbind() の呼び出し時に、クエリから compute ローが返されることを確認してください。

追加情報

バージョン すべて

20025

記号定数　SYBEIICL

メッセージ　サーバから返された整数カラム長が正しくありません。

考えられる原因　受信した整数長が正しくない (1、2、4 以外)。

アクション / 解決法　発生頻度は高くありませんが、Open Server からの受信である場合は、Open Server のコーディングを確認してください。

追加情報

バージョン　すべて

20026

記号定数　SYBECNOR

メッセージ　範囲外のカラム番号です。

考えられる原因　dbdata() に指定されているカラム番号が結果セットに含まれていない。

アクション / 解決法　使用可能なカラムを参照するようにコードを修正してください。

追加情報

バージョン　すべて

20027

記号定数　SYBENPRM

メッセージ　この dboption には NULL パラメータは使えません。

考えられる原因　NULL パラメータが dbsetopt() に指定されている。

アクション / 解決法　有効なパラメータを指定するようにコードを修正してください。

追加情報

バージョン　すべて

20028

記号定数 SYBEUVDT

メッセージ 不明な可変長データ型を検出しました。

考えられる原因 不明な可変長データ型をサーバから受信した。

アクション / 解決法 発生頻度は高くありませんが、サーバが Open Server である場合は、Open Server のコーディングを確認してください。

追加情報

バージョン すべて

20029

記号定数 SYBEUFDT

メッセージ 不明な固定長データ型を検出しました。

考えられる原因 不明な固定長データ型をサーバから受信した。

アクション / 解決法 発生頻度は高くありませんが、サーバが Open Server である場合は、Open Server のコーディングを確認してください。

追加情報

バージョン すべて

20030

記号定数 SYBEWAID

メッセージ `DB-Library 内部エラー: ALTNAME の後の ALTFMT に正しくない ID があります。`

考えられる原因 サーバが compute ローに対して正しくない ID を送信している。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20031

記号定数　SYBECDNS

メッセージ　データストリームは存在しない選択リストのメンバから compute カラムが得られたことを示しています。

考えられる原因　存在しない選択リストのメンバから compute カラムが抽出されている。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20033

記号定数　SYBEABMT

メッセージ　正しくないカラムと変数タイプで dbbind() を実行しようとしました。

考えられる原因　dbbind() (またはその他のバインド関数) が互換性のない**カラム**と *vartype* 値で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20034

記号定数　SYBEABNP

メッセージ　NULL ポインタを使ってバインドしようとしました。

考えられる原因　dbbind() の *varaddr* 引数が不正だった。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20035

記号定数 SYBEAAMT

メッセージ 正しくないカラムと変数タイプで dbbind() を実行しようとしました。

考えられる原因 プログラムのデータ型がカラムのデータ型と一致していない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20036

記号定数 SYBENXID

メッセージ サーバが分散トランザクション ID を付与しませんでした。

考えられる原因 コミット・サービスの初期化中に問題が発生した。

アクション / 解決法 interfaces ファイル内のエントリが正しいことを確認してください。

追加情報

バージョン すべて

20037

記号定数 SYBERXID

メッセージ サーバがクライアントの分散トランザクション ID を認識しませんでした。

考えられる原因 プログラミング・エラー。

アクション / 解決法 stat_xact() に指定されている *commid* パラメータを確認してください。

追加情報

バージョン すべて

20038

記号定数　SYBEICN

メッセージ　`compute ID または compute カラム番号が正しくありません。`

考えられる原因　無効な *computeid* または compute カラム番号が dbalt*() ルーチンに渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20039

記号定数　SYBENMOB

メッセージ　`'ORDER BY' 句のそのようなメンバはありません。`

考えられる原因　無効なカラム ID が dbordercol() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20040

記号定数　SYBEAPUT

メッセージ　不明なトークンを印刷しようとしました。

考えられる原因　未知の *type* パラメータが dbprtype() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20041

記号定数 SYBEASNL

メッセージ `NULL の loginrec にフィールドを設定しようとしました。`

考えられる原因 *LOGINREC* ポインタが NULL。

アクション / 解決法 dblogin() を呼び出してログイン・レコードを割り当ててください。

追加情報 メモリが割り当てられない場合、dblogin() は NULL ポインタを返します。

バージョン すべて

20042

記号定数 SYBENTLL

メッセージ `loginrec フィールドには名前が長すぎます。`

考えられる原因 DBSETL* ルーチンに渡された *name* パラメータが *DBMAXNAME* より長い。

アクション / 解決法 違う名前を使用してください。

追加情報

バージョン すべて

20043

loginrec==>loginrec

記号定数 SYBEASUL

メッセージ `設定しようとした loginrec フィールドがわかりません。`

考えられる原因 内部関数 dbsetlname が、存在しない *loingrec* フィールドを設定しようとした。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20044

記号定数　SYBERDNR

メッセージ　`Attempt to retrieve data from a non-existent row.`

考えられる原因　使用できるデータがない状態で dbdata() が呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20045

記号定数　SYBENSI

メッセージ　負の開始インデックスが `dbstrcpy()` に渡されました。

考えられる原因　dbstrcpy() に渡された *start* パラメータがゼロ未満だった。

アクション / 解決法　値を修正してください。

追加情報

バージョン　すべて

20046

記号定数　SYBEABNV

メッセージ　`NULL` のプログラム変数にバインドしようとしました。

考えられる原因　dbbind に渡された *destvar* パラメータが NULL だった。

アクション / 解決法　プログラム変数にメモリのポインタを指定してください。

追加情報

バージョン　すべて

20047

記号定数 SYBEDDNE

メッセージ DBPROCESS が dead であるかまたは使えません。

考えられる原因 DBPROCESS でエラーが発生して、使用できなくなっている。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20048

記号定数 SYBECUFL

メッセージ データ変換がアンダフローしました。

考えられる原因 dbconvert() の結果、アンダフローが発生した。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報 変換の結果、精度が損なわれています。この結果を受け入れられない場合、送信先変数が送信元の値を完全に表現できるようにアプリケーションを修正する必要があります。

バージョン すべて

20049

記号定数 SYBECOFL

メッセージ データ変換がオーバフローしました。

考えられる原因 dbconvert() の結果、オーバフローが発生した。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報 デスティネーション・バッファのサイズが小さくて変換後の値を格納できません。

バージョン すべて

20050

記号定数　SYBECSYN

メッセージ

```
Attempt to convert data stopped by syntax error in
source field.
```

考えられる原因　変換エラーが発生している。

アクション / 解決法　dbconvert() の引数が正しいことを確認してください。関数ドキュメントで、変換元の型が変換先の型に変換できることを確認します。

追加情報

バージョン　すべて

20051

記号定数　SYBECLPR

メッセージ　データ変換で総桁数が失われました。

考えられる原因　dbconvert() を呼び出した結果、総桁数が失われている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20052

記号定数　SYBECNOV

メッセージ　オーバフローした変数を NULL に設定しようとしました。

考えられる原因　dbconvert() *src* パラメータは NULL にできない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20053

記号定数 SYBERDCN

メッセージ 要求されたデータ変換は存在しません。

考えられる原因 dbconvert() が *srctype* を *desttype* に変換できない。

アクション / 解決法 データ型変換テーブルで、有効な型の組み合わせを確認してください。

追加情報

バージョン すべて

20054

記号定数 SYBESFOV

メッセージ `dbsafestr()` でデスティネーション・バッファがオーバフローしました。

考えられる原因 デスティネーション・バッファのサイズが小さい。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20055

記号定数 SYBEUNT

メッセージ

`interfaces` ファイルにあるネットワーク・タイプがわかりません。

考えられる原因 interfaces ファイルに適切なサーバ・エントリがない。

アクション / 解決法 interfaces ファイルを修正してください。

追加情報

バージョン 15.7 ESD #3 より前のバージョン。

20056

記号定数　SYBECLOS

メッセージ　ネットワーク接続をクローズ中にエラーが発生しました。

考えられる原因　ネットワーク・エンドポイントのクローズ中にエラーが発生した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20060

記号定数　SYBECSYN

メッセージ　不明な `datatype` を検出しました。

考えられる原因　DB-Library ルーチンが呼び出されて、不明なデータ型が渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20061

記号定数　SYBETSIT

メッセージ　正しくないタイムスタンプで `dbtsput()` を呼び出そうとしました。

考えられる原因　Adaptive Server カラムにタイムスタンプがない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20062

記号定数 SYBECSYN

メッセージ タイムスタンプ・カラムを持たないテーブルのタイムスタンプを更新しようとしました。

考えられる原因 ブラウズできないローで dbtsput() が呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20063

記号定数 SYBEBDIO

メッセージ バルク・コピーの方向が正しくありません。`IN` か `OUT` のどちらかでなければなりません。

考えられる原因 bcp_init() が無効な *direction* パラメータで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20064

記号定数 SYBEBCNT

メッセージ 存在しないサーバ・テーブルでバルク・コピーを使用しようとしました。

考えられる原因 サーバで無効な *table* パラメータが bcp_init() に渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20065

記号定数　SYBEIFNB

メッセージ　`bcp_control()` に渡されたフィールド番号が正しくありません。

考えられる原因　bcp_control() に渡された *field* パラメータが適切でない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20066

記号定数　SYBETTS

メッセージ　バルク・コピーを使ってホスト・ファイルにコピーしようとしているテーブルの長さが、スキップするようにバルク・コピーに指示したロー数を下回っています。

考えられる原因　テーブル内のロー数より多い BCPFIRST 値で dbcontrol() が呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20067

記号定数　SYBEKBCO

メッセージ　`1000` ローがホスト・ファイルにバルク・コピーされました。

考えられる原因　1000 ローがコピーされた。

アクション / 解決法　対処する必要はありません。

追加情報　これは、ホスト・ファイルにローがコピーされる進捗状況をユーザに通知する情報メッセージです。

バージョン　すべて

20068

記号定数 SYBEBBCI

メッセージ バッチによって、サーバへのバルク・コピーが正常に行われました。

考えられる原因 バッチのコピーが正常に行われた。

アクション / 解決法 対処する必要はありません。

追加情報 これは、コピーの進捗情報をユーザに通知する情報メッセージです。

バージョン すべて

20069

記号定数 SYBEKBCI

メッセージ `Bcp: 1000 ローがサーバに送信されました。`

考えられる原因 ローがサーバに送信された。

アクション / 解決法 対処する必要はありません。

追加情報 これは、コピーの進捗情報をユーザに通知する情報メッセージです。

バージョン すべて

20070

記号定数 SYBEBCRE

メッセージ `BCP データ・ファイルを読み込み中に I/O エラーが発生しました。`

考えられる原因 システム I/O ルーティングに失敗した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20071

記号定数　SYBETPTN

メッセージ　構文エラー : テーブル名には 2 つのピリオドしか使えません。

考えられる原因　テーブル参照構文が正しくない。

アクション / 解決法　クエリを修正してください。

追加情報

バージョン　すべて

20072

記号定数　SYBEBCWE

メッセージ　BCP データ・ファイルを書き込み中に I/O エラーが発生しました。

考えられる原因　システム I/O ルーティングに失敗した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20073

記号定数　SYBEBCNN

メッセージ　NULL 値を受け入れないサーバ・カラム <colname> に NULL 値をバルク・コピーしようとしました。

考えられる原因　挿入 / 更新クエリが正しくない。

アクション / 解決法　クエリを修正してください。

追加情報

バージョン　すべて

20074

記号定数 SYBEBCOR

メッセージ サイズが大きすぎるローをサーバにバルク・コピーしようとしました。

考えられる原因 コピーするデータがサーバ・カラムに対して大きすぎる。

アクション / 解決法 *bcp* ファイル / サーバ・スキーマを調整してください。

追加情報

バージョン すべて

20075

記号定数 SYBEBCIS

メッセージ 正しくないサイズのカラム値をサーバにバルク・コピーしようとしました。

考えられる原因 コピーするデータがサーバ・カラムに対して大きすぎる。

アクション / 解決法 *bcp* ファイル / サーバ・スキーマを調整してください。

追加情報

バージョン すべて

20076

記号定数 SYBEBCPI

メッセージ `bcp_init()` は他の `BCP` ルーチンの前に呼び出さなければなりません。

考えられる原因 bcp_init() が呼び出されていない。

アクション / 解決法 bcp_init() を呼び出してください。

追加情報

バージョン すべて

20077

記号定数　SYBEBCOR

メッセージ　`bcp_bind()`、`bcp_collen()`、および `bcp_colptr()` は、コピー方向を `DB_IN` に設定して `bcp_init()` を呼び出した後にだけ、呼び出すことができます。

考えられる原因　上記の関数が bcp_init() の前に呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20078

記号定数　SYBEBCPB

メッセージ　`NULL` 以外の入力ファイル名を `bcp_init()` に渡した後では `bcp_bind()` を使用できません。

考えられる原因　入力ファイルが指定されている状態でプログラム変数からコピーしようとした。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20079

記号定数　SYBEVDPT

メッセージ　`For bulk copy, all variable-length data must have either a length-prefix or a terminator specified.`

考えられる原因　*length-prefix* または *terminator* がない。

アクション / 解決法　bcp_colfmt() に渡すパラメータを修正してください。

追加情報

バージョン　すべて

20080

記号定数 SYBEBIVI

メッセージ `bcp_columns()` と `bcp_colfmt()` は、正しい入力ファイルを `bcp_init()` に渡した後にだけ使用できます。

考えられる原因 bcp_init() が無効な *hfile* パラメータで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20081

記号定数 SYBEBCBC

メッセージ `bcp_columns()` は `bcp_colfmt()` の前に呼び出さなければなりません。

考えられる原因 bcp_columns() が呼び出される前に、bcp_colfmt() が呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20082

記号定数 SYBEBCFO

メッセージ `BCP` ホスト・ファイルは少なくとも `1` つのカラムを持たなければなりません。

考えられる原因 bcp_columns() が無効な *host_colcount* パラメータで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20083

記号定数　SYBEBCVH

メッセージ　`bcp_exec()` は正しいホスト・ファイルを `bcp_init()` に渡した後にだけ呼び出すことができます。

考えられる原因　bcp_init() が無効な *hfile* パラメータで呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20084

記号定数　SYBEBCUO

メッセージ　`BCP:` ホスト・データ・ファイルをオープンできません。

考えられる原因　bcp_init() が無効な *hfile* パラメータで呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20085

記号定数　SYBEBCUC

メッセージ　`BCP:` ホスト・データ・ファイルをクローズできません。

考えられる原因　システムのクローズ・ルーチンが失敗した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20086

記号定数 SYBEBUOE

メッセージ BCP: エラー・ファイルをオープンできません。

考えられる原因 bcp_init() が無効な *errfile* パラメータで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20087

open==>close

記号定数 SYBEBUCE

メッセージ BCP: エラー・ファイルをオープンできません。

考えられる原因 システムのクローズ・ルーチンが失敗した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20088

記号定数 SYBEBWEF

メッセージ bcp 入力ファイルへの書き込み中に I/O エラーが発生しました。

考えられる原因 システム I/O ルーチンに失敗した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20091

記号定数　SYBEASEC

メッセージ　空のコマンド・バッファをサーバに送信しようとしました。

考えられる原因　dbcmd() が呼び出されていないか、空の文字列で呼び出されている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20092

記号定数　SYBETMTD

メッセージ　dbmoretext() 呼び出しで送信しようとした TEXT データが多すぎます。

考えられる原因　dbwritetext() が、設定する合計テキスト長を指定している。このエラーは、dbmoretext() の呼び出しで送信される合計テキスト長が dbwritetext() の呼び出しで指定されている値を超えていることを示しています。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20093

記号定数　SYBENTTN

メッセージ　dbtxtsput() を使用して新しいテキスト・タイムスタンプを存在しないデータ・ローに入れようとしました。

考えられる原因　dbtxtsput() の呼び出しが正しくない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20094

記号定数 SYBEDNTI

メッセージ dbtxtsput() を使用して新しいテキスト･タイムスタンプをデータ型が SYBTEXT でも SYBIMAGE でもないカラムに入れようとしました。

考えられる原因 dbtxtsput() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20095

記号定数 SYBEBTMT

メッセージ bcp_moretext() 呼び出しで、送信しようとした TEXT データが多すぎます。

考えられる原因 bcp_moretext() の呼び出しで送信されたテキストのサイズがカラム・サイズより大きい。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20096

記号定数 SYBEORPF

メッセージ リモート・パスワードを設定すると、ログイン・レコードのリモート・パスワード・フィールドがオーバフローする可能性があります。

考えられる原因 リモート・パスワードが長すぎる。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報 リモート・パスワード・バッファ長は 255 バイトです。バッファ内の各パスワードのエントリは、パスワード自体と、関連付けられているサーバ名、および追加の 2 バイトで構成されます。

バージョン すべて

20097

記号定数　SYBEUVBF

メッセージ　読み込もうとした BCP フォーマット・ファイルのバージョンがわかりません。

考えられる原因　フォーマット・ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20098

記号定数　SYBEBUOF

メッセージ　BCP: フォーマット・ファイルをオープンできません。

考えられる原因　不適切な *filename* パラメータが bcp_readfmt() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20099

記号定数　SYBEBUCF

メッセージ　BCP: フォーマット・ファイルをクローズできません。

考えられる原因　システムの close ルーチンが失敗した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20100

記号定数 SYBEBRFF

メッセージ bcp フォーマット・ファイルの読み取り中に I/O エラーが発生しました。

考えられる原因 システム I/O ルーチンに失敗した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20101

記号定数 SYBEBWFF

メッセージ bcp フォーマット・ファイルを書き込み中に I/O エラーが発生しました。

考えられる原因 システム I/O ルーチンに失敗した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20102

記号定数 SYBEBUDF

メッセージ BCP: フォーマット・ファイルで、認識できないデータ型が検出されました。

考えられる原因 入力ファイルが破損している。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20103

記号定数　SYBEBIHC

メッセージ　bcp フォーマット・ファイル内で不正なホストカラム番号が検出されました。

考えられる原因　入力ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20104

記号定数　SYBEBEOF

メッセージ　BCP データ・ファイル内で予期しない EOF を検出しました。

考えられる原因　入力ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20105

記号定数　SYBEBCNL

メッセージ　BCP データ・ファイル内で値が負の長さプレフィクスを検出しました。

考えられる原因　入力ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20106

記号定数 SYBEBCSI

メッセージ ホスト・ファイル・カラムは、サーバにコピーするときだけスキップすることができます。

考えられる原因 プログラミング・エラー。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20107

記号定数 SYBEBCIT

メッセージ SYBCHAR、SYBBINARY、SYBTEXT、または SYBIMAGE 以外のプログラム変数では、BCP ターミネータの使用は正しくありません。

考えられる原因 bcp_bind() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20108

記号定数 SYBEBCSA

メッセージ BCP ホスト・ファイル <filename> には <n> ローしか含まれていません。これらのすべてのローを省略するこはできません。

考えられる原因 bcp_control() が、BCPFIRST を入力ファイル内のロー数より大きい値に設定している。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20109

記号定数　SYBENULL

メッセージ　NULL の DBPROCESS ポインタが DB-Library に渡されました。

考えられる原因　DB-Library ルーチンが NULL DBPROCESS ポインタを渡した。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20110

記号定数　SYBEUNAM

メッセージ　カレント・ユーザ名をオペレーティング・システムから得られません。

考えられる原因　getpwuid() システム・コールが失敗した。

アクション / 解決法　システム管理者に連絡してください。

追加情報

バージョン　すべて

20111

記号定数　SYBEBCRO

メッセージ　BCP ホスト・ファイル <filename> には <n> ローしか含まれていません。要求された <m> ローを読み込むことはできませんでした。

考えられる原因　bcp_control() が、*BCPLAST* を入力ファイル内のロー数より大きい値に設定している。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20112

記号定数 SYBEMPLL

メッセージ DBPROCESS の最大数を 1 より小さく設定しようとしました。

考えられる原因 dbsetmaxprocs() が 1 未満の値で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20113

記号定数 SYBERPIL

メッセージ タイプが SYBCHAR、SYBVARCHAR、SYBBINARY、または SYBVARBINARY のパラメータの datalen として -1 を dbrpcparam() に渡すことはできません。

考えられる原因 可変長データ型に不適切なデータ長が指定された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20114

記号定数 SYBERPUL

メッセージ rpcparam() で SYBINTN、SYBDATETIMN、SYBMONEYN、または SYBFLTN パラメータを渡すとき、DB-Library がそれを SYBINT1、SYBINT2、SYBINT4、SYBMONEY、SYBMONEY4 などとして認識できるようにパラメータの最大長または実際の長さを指定する必要があります。

考えられる原因 dbrpcparam() が不適切な *datlen* 値で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20115

記号定数　SYBEUNOP

メッセージ　dbsetopt() に渡されたオプションがわかりません。

考えられる原因　dbsetopt() の呼び出しが正しくない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20116

記号定数　SYBECRNC

メッセージ　現在のローは COMPUTE 句 <computeid> の結果ではないため、このローからそのデータを抽出することはできません。

考えられる原因　dbadata() が、ALTER ROW 句の結果ではないローで呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20117

記号定数　SYBERTCC

メッセージ　dbreadtext() は COMPUTE 句を持つクエリーの結果を受信するために使用できません。

考えられる原因　dbreadtext() が呼び出されて、ALTER ROW 句の結果からテキストが取得された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20118

記号定数 SYBERTSC

メッセージ `dbreadtext()` は `1` つの結果カラムを持つクエリーの結果を受信するためにだけ使用できます。

考えられる原因 複数のカラムを含む結果セットで dbreadtext() が呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20119

記号定数 SYBEUCRR

メッセージ `Internal software error:dbpasswd()` によって報告された接続結果がわかりません。

考えられる原因 サーバとの接続に問題がある。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20120

記号定数 SYBERPNA

メッセージ `RPC` 機能は、バージョン番号が `4.0` 以上のサーバを使用しているときにのみ利用できます。

考えられる原因 このバージョンの Adaptive Server が RPC をサポートしていない。

アクション / 解決法 Adaptive Server をアップグレードします。

追加情報

バージョン すべて

20121

記号定数　SYBEOPNA

メッセージ　テキスト / イメージ機能は、バージョン番号が 4.0 以上のサーバを使用しているときにのみ利用できます。

考えられる原因　このバージョンの Adaptive Server がテキスト / イメージをサポートしていない。

アクション / 解決法　Adaptive Server をアップグレードします。

追加情報

バージョン　すべて

20122

記号定数　SYBEFGTL

メッセージ　BCP: コピーする最初のローのロー番号はコピーする最後のローのロー番号より大きくてはいけません。

考えられる原因　bcp_control() が、矛盾する *BCPFIRST* と *BCPLAST* で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20123

記号定数　SYBECWLL

メッセージ　カラム幅を 1 より小さく設定しようとしました。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　廃止

バージョン　なし

20124

記号定数 SYBEUFDS

メッセージ `dbstrbuild()` で認識できないフォーマットがありました。

考えられる原因 旧式関数 dbfmtinstall() でカスタムインストールされているどのフォーマット指定子とも一致しないフォーマット指定子がフォーマット文字列で指定されている。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20125

記号定数 SYBEUCPT

メッセージ `dbstrbuild()` に認識できないカスタム・フォーマットのパラメータ・タイプがありました。

考えられる原因 dbstrbuild() がフォーマット文字列の無効なパラメータ・タイプで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20126

記号定数 SYBETMCF

メッセージ `dbfmtinstall()`（廃止関数）でインストールしようとしたカスタム・フォーマットが多すぎます。

考えられる原因 *MAXFMTS (20)* の指定を超えるフォーマットで dbfmtinstall() が呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20127

記号定数　SYBEAICF

メッセージ

```
Error in attempting to install custom format.
```

考えられる原因　dbfmtinstall() の一般エラー。

アクション / 解決法　アプリケーション・コードを検証してください。

追加情報

バージョン　すべて

20128

記号定数　SYBEADST

メッセージ

```
Error in attempting to determine the size of a pair of
translation tables.
```

考えられる原因　dbload_xlate() が無効な *srv_charset* で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報　指定されている文字セットが *$SYBASE/charsets* に存在しているか確認してください。

バージョン　すべて

20129

記号定数　SYBEALTT

メッセージ

```
Error in attempting to load a pair of translation
tables.
```

考えられる原因　変換テーブルのロード中に内部エラーが発生した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20130

記号定数 SYBEAPCT

メッセージ 文字セット変換を実行しようとしてエラーが発生しました。

考えられる原因 dbxlate() が失敗した。

アクション / 解決法 dbxlate() に渡すパラメータを確認してください。

追加情報

バージョン すべて

20131

記号定数 SYBEXOCI

メッセージ バルク・コピーを使用してホスト・ファイルからサーバにデータをコピーするときに、文字セット変換で出力先バッファがオーバフローしました。

考えられる原因 内部変換エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20132

記号定数 SYBEFSHD

メッセージ `Error in attempting to find the Sybase home directory.`

考えられる原因 *$SYBASE* 環境変数が正しくない。

アクション / 解決法 設定を修正してください。

追加情報

バージョン すべて

20133

記号定数

メッセージ　　ローカライゼーション・ファイルをオープンしようとして、エラーが発生しました。

考えられる原因　　ローカライゼーション・ファイルをオープンできない。

アクション / 解決法　　*$SYBASE* が正しいか確認してください。

追加情報

バージョン　　すべて

20134

記号定数　　SYBEARDI

メッセージ　　`Error in attempting to read datetime information from a localization file.`

考えられる原因　　*locales* ファイルが破損している。

アクション / 解決法　　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　　すべて

20135

記号定数　　SYBEURCI

メッセージ　　コピーライト情報を `DBLIB` ローカライゼーション・ファイルから読み込めません。

考えられる原因　　*locales* ファイルが破損している。

アクション / 解決法　　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　　すべて

20136

記号定数 SYBEARDL

メッセージ `dblib.loc` ローカライゼーション・ファイルを読み込もうとしてエラーが発生しました。

考えられる原因 *locales* ファイルが破損している。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20137

記号定数 SYBEURMI

メッセージ `money` フォーマット情報を `DBLIB` ローカライゼーション・ファイルから読み込めません。

考えられる原因 *locales* ファイルが破損している。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20138

記号定数 SYBEUREM

メッセージ エラー・ニモニックを `DBLIB` ローカライゼーション・ファイルから読み込めません。

考えられる原因 *locales* ファイルが破損している。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20139

記号定数　SYBEURES

メッセージ　エラー文字列を DBLIB ローカライゼーション・ファイルから読み込めません。

考えられる原因　*locales* ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20140

記号定数　SYBEUREI

メッセージ　エラー情報を DBLIB ローカライゼーション・ファイルから読み込めません。

考えられる原因　*locales* ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20141

記号定数　SYBEOREN

メッセージ　警告 : 範囲外のエラー番号が dblib.loc に見つかりました。最大許容エラー番号は sybdb.h の DBERRCOUNT で定義されています。

考えられる原因　*locales* ファイルが破損している。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20142

記号定数 SYBEISOI

メッセージ 無効なソート順の情報を検出しました。

考えられる原因 *locales* ファイルが破損している。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20143

記号定数 SYBEIDCL

メッセージ サーバによって返された `datetime` カラム長が正しくありません。有効な `datetime` カラム長は `4` および `8` バイトです。

考えられる原因 内部 Adaptive Server エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20144

記号定数 SYBEIMCL

メッセージ サーバによって返された `money` カラム長が正しくありません。有効な `money` カラム長は `4` および `8` バイトです。

考えられる原因 内部 Adaptive Server エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20145

記号定数　SYBEIFCL

メッセージ　サーバによって返された浮動小数点カラム長が正しくありません。有効な浮動小数点カラム長は 4 および 8 バイトです。

考えられる原因　内部 Adaptive Server エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20146

記号定数　SYBEUTDS

メッセージ　サーバから受信した TDS バージョンが認識できません。

考えられる原因　内部エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20147

記号定数　SYBEBUCF

メッセージ　BCP: フォーマット・ファイルを作成できません。

考えられる原因　アクセス権限が十分でない。

アクション / 解決法　アクセス権限を確認してください。

追加情報　このエラーは、作成時とクローズ時の両方の失敗に対して発生します。

バージョン　すべて

20148

記号定数 SYBEACNV

メッセージ NULL のデスティネーション変数でデータ変換を実行しようとしました。

考えられる原因 dbconvert() または dbdatechar() に NULL ポインタが渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20149

記号定数 SYBEDPOR

メッセージ datepart 定数が範囲外です。

考えられる原因 無効な *datepart* が dbdatechar() に渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20150

記号定数 SYBENDC

メッセージ 負の構成要素を数値フォームの日付に持つことはできません。

考えられる原因 無効な *value* が dbdatechar() に渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20151

記号定数　SYBEMVOR

メッセージ　`Month 値は 1 から 12 です。`

考えられる原因　無効な *month* 値が dbdatechar() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20152

記号定数　SYBEDVOR

メッセージ　`Day 値は 1 から 7 です。`

考えられる原因　無効な *day* 値が dbdatechar() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20153

記号定数　SYBENBVP

メッセージ　`dbsetnull() に NULL の bindval ポインタを渡すことはできません。`

考えられる原因　*bindval* として NULL ポインタが渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報　NULL 値が検出されたときに、*bindval* が代わりに使用される値になります。NULL はここでは不適切です。

バージョン　すべて

20154

記号定数 SYBESPID

メッセージ dbspid() を NULL の dbproc で呼び出しました。

考えられる原因 コーディングが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20155

記号定数 SYBENDTP

メッセージ dbdatecrack() を NULL の datetime パラメータで呼び出しました。

考えられる原因 コーディングが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20156

記号定数 SYBEXTN

メッセージ dbfree_xlate() への xlt_tosrv および xlt_todisp パラメータが NULL でした。

考えられる原因 無効な NULL パラメータが渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20157

記号定数　SYBEXTDN

メッセージ　警告 : dbfree_xlate() への xlt_todisp パラメータが NULL でした。xlt_tosrv パラメータに関連する空き領域は解放されました。

考えられる原因　NULL *xlt_todisp* パラメータが dbfree_xlate() に渡された。

アクション / 解決法　*xlt_tosrv* を解放しないでください。

追加情報

バージョン　すべて

20158

記号定数　SYBEXTSN

メッセージ　警告 : dbfree_xlate() への xlt_tosrv パラメータが NULL でした。xlt_todisp パラメータに関連する空き領域は解放されました。

考えられる原因　NULL *xlt_tosrv* パラメータが dbfree_xlate() に渡された。

アクション / 解決法　*xlt_todisp* を解放しないでください。

追加情報

バージョン　すべて

20159

記号定数　SYBENUM

メッセージ　DB-Library に渡された引数の数が正しくありません。

考えられる原因　間違った数の引数が DB-Library ルーチンに渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20160

記号定数 SYBETYPE

メッセージ `Hyper/DB-Library` に渡された引数タイプが正しくありません。

考えられる原因 dbregparam() が無効なタイプで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20161

記号定数 SYBEGENOS

メッセージ オペレーティング・システムの一般エラーです。

考えられる原因 内部状態変数の作成に失敗した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20162

記号定数 SYBEPAGE

メッセージ `dbpage??()` オペレーションに指定されたリソース・タイプまたは長さが正しくありません。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20163

記号定数　SYBEOPTNO

メッセージ　オプションは使用できません。

考えられる原因　セキュリティ・オプションの設定に失敗した内部エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20164

記号定数　SYBEETD

メッセージ　`dbmoretext()` で期待された量の `TEXT` または `IMAGE` データの送信ができませんでした。

考えられる原因　テキスト・データの送信が完了する前にアプリケーションがサーバから結果を読み込もうとした。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20165

記号定数　SYBERTYPE

メッセージ　`Hyper/DB-Library` に渡されたリソース・タイプが正しくありません。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　廃止

バージョン　なし

20166

記号定数 SYBERFILE

メッセージ リソース・ファイルをオープンできません。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20167

記号定数 SYBEFMODE

メッセージ 読み込み / 書き込み / 追加モードがファイルに対して拒否されました。

考えられる原因 dbstrbuild() がフォーマット文字列の無効なパラメータ・タイプで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報 廃止

バージョン なし

20168

記号定数 SYBESLCT

メッセージ 指定されたフィールドを選択またはコピーできませんでした。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20169

記号定数　SYBEZTXT

メッセージ　dbwritetext() で、長さがゼロの TEXT または IMAGE を dataserver に送信しようとしました。

考えられる原因　dbwritetext() 呼び出しの引数が無効。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20170

メッセージ・タイプ　エラー

記号定数　SYBENTST

メッセージ　VMS: オープンされているファイルは stream_lf でなければなりません。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　廃止

バージョン　なし

20171

記号定数　SYBEOSSL

メッセージ　不正なログイン : オペレーティング・システムのログイン・セキュリティ・レベルが secure サーバの範囲内にありません。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　廃止

バージョン　なし

20172

記号定数 SYBEESSL

メッセージ 不正なログイン： 入力されたログイン・セキュリティ・レベルがオペレーティング・システムのレベルと一致しません。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20173

記号定数 SYBENLNL

メッセージ プログラムが指定のネットワーク・ライブラリとリンクされていません。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20174

記号定数 SYBENHAN

メッセージ `dbrecvpassthru()` を `NULL` の `handle` パラメータで呼び出しました。

考えられる原因 dbrecvpassthru() が無効なパラメータで呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20175

記号定数　SYBENBUF

メッセージ　`dbsendpassthru()` を `NULL` の `buf` パラメータで呼び出しました。

考えられる原因　dbsendpassthru() が無効なパラメータで呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20176

記号定数　SYBENULP

メッセージ　`<paramname>` パラメータに `NULL` を指定して `<routine>` が呼び出されました。

考えられる原因　DB-Library ルーチンが無効な NULL パラメータで呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20177

記号定数　SYBENOTI

メッセージ　イベント・ハンドラは通知要求を行う前にインストールしなければなりません。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　廃止

バージョン　なし

20178

記号定数 SYBEEVOP

メッセージ `dbregwatch()` を呼び出したオプション・パラメータが正しくありません。

考えられる原因 無効なオプション値が dbregwatch() に渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20179

記号定数 SYBENEHA

メッセージ `dbreghandle() を NULL の handler パラメータで呼び出しました。`

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 廃止

バージョン なし

20180

記号定数 SYBETRAN

メッセージ `DBPROCESS が別のトランザクションで使用されています。`

考えられる原因 この DBPROCESS に対する前のコマンドの処理が完了していない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20181

記号定数　SYBEEVST

メッセージ　`dbregparam()` を呼び出す前にトランザクションを開始しなければなりません。

考えられる原因　dbreginit() または dbregparam() が、dbregparam() の呼び出し前に呼び出されていない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20182

記号定数　SYBEEINI

メッセージ　`dbregexec()` の前に `dbreginit()` を呼び出さなければなりません。

考えられる原因　dbreginit() が dbregexec() の呼び出し前に呼び出されていない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20183

記号定数　SYBEECRT

メッセージ　`dbnpcreate()` の前に `dbnpdefine()` を呼び出さなければなりません。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　廃止

バージョン　なし

20184

記号定数 SYBEECAN

メッセージ

```
Attempted to cancel unrequested event notification.
```

考えられる原因 以前に dbregwatch() が実行されていないのに dbregnowatch() が呼び出されている。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20185

記号定数 SYBEEUNR

メッセージ

```
Unsolicited event notification received.
```

考えられる原因 dbreghandle() が呼び出されてイベント・ハンドラがアンインストールされているが、dbregnowatch() の呼び出しによりノーティフィケーション・リクエストがキャンセルされていない。イベントが発生しても、インストールされているハンドラがありません。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20186

記号定数 SYBERPCS

メッセージ

```
dbrpcparam() または dbrpcsend() の前に dbrpcinit() を呼び出さなければなりません。
```

考えられる原因 dbrpcinit() が、dbrpcparam() または dbrpcsend() を呼び出す前に呼び出されていない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20187

記号定数　SYBETPAR

メッセージ　`SYBTEXT` または `SYBIMAGE` パラメータが定義されませんでした。

考えられる原因　テキスト / イメージパラメータが定義されていない RPC の実行中に dbwritetext() または dbmoretext() が呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20188

記号定数　SYBETEXS

メッセージ　`dbmoretext()` を呼び出したサイズ・パラメータが正しくありません。

考えられる原因　負の *bufsize* パラメータ値で、dbmoretext() または dbreadtext() が呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20189

記号定数　SYBETRAC

メッセージ　オンではないトレース・フラグをオフしようとしました。

考えられる原因　dbtraceoff() が無効な *flag* パラメータで呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20190

記号定数 SYBETRAS

メッセージ `DB-Library 内部エラー - トレース構造体が見つかりません。`

考えられる原因 DBPROCESS 構造体にトレース・レコードがない。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20191

記号定数 SYBEPRTF

メッセージ `dbtracestring() は printfunc() からだけ呼び出すことができます。`

考えられる原因 dbtraceon() で設定された printfunc() からでなく、dbtracestring() が直接呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20192

記号定数 SYBETRSN

メッセージ `dbtracestring() に渡された numbytes パラメータが正しくありません。`

考えられる原因 dbtracestring() に渡された *num* パラメータに負の値が設定されている。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20193

記号定数　SYBEBPKS

メッセージ　DBSETLPACKET() ではパケット・サイズ・パラメータは 256 から 9999 までででなければなりません。

考えられる原因　無効なサイズが DBSETLPACKET() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20194

記号定数　SYBEIPV

メッセージ　<value> は <routine> の <paramname> パラメータには正しくない値です。

考えられる原因　パラメータ値がそのパラメータの範囲外である。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20195

記号定数　SYBEMOV

メッセージ　関数 <routine> で通貨算術演算がオーバフローしました。

考えられる原因　無効なパラメータが dbmny*() 関数に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20196

記号定数 SYBEDIVZ

メッセージ 関数 <routine> の中で $0.00 で除算しようとしました。

考えられる原因 無効な値が dbmydiv() または dbmny4div() に渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20197

記号定数 SYBEASTL

メッセージ AST レベルで同期 I/O をしようとしました。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 古い形式の VMS 固有のエラー・メッセージ

バージョン なし

20198

記号定数 SYBESEFA

メッセージ 接続が存在する場合、DB_SETEVENT_VMS を呼び出すことはできません。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 古い形式の VMS 固有のエラー・メッセージ

バージョン なし

20199

記号定数　SYBEPOLL

メッセージ　アクティブな dbpoll() がすでにあります。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　古い形式の VMS 固有のエラー・メッセージ

バージョン　なし

20200

記号定数　SYBENOEV

メッセージ　レジスタード・プロシージャ・ノーティフィケーションが使用できないので、DBPOLL を呼び出すことはできません。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報　古い形式の VMS 固有のエラー・メッセージ

バージョン　なし

20201

記号定数　SYBEBADPK

メッセージ　パケット・サイズ <requested size> はサポートされていません。代わりにサイズ <other size> を使用します。

考えられる原因　DB-Library はリクエストされたパケット・サイズを格納できない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20202

記号定数 SYBESECURE

メッセージ このバージョンでは、secure サーバ機能はサポートされていません。

考えられる原因 古い形式のルーチン DBSETLHIER() が呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20203

記号定数 SYBECAP

メッセージ DB-Library 機能は、サーバによって受け入れられませんでした。

考えられる原因 無効な TDS をサーバから受信した。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20204

記号定数 SYBEFUNC

メッセージ

```
Functionality not supported at the specified version
level.
```

考えられる原因 このバージョンでサポートされていない DB-Library ルーチンが呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20205

記号定数　SYBERESP

メッセージ　dbresponse() に渡す応答関数アドレスは NULL 以外でなければなりません。

考えられる原因　ドキュメント化されていない関数 dbresponse() に渡された *response_func* パラメータが NULL だった。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20206

記号定数　SYBEIVERS

メッセージ　無効な version レベルが指定されました。

考えられる原因　無効な *version* パラメータが dbsetversion() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20207

記号定数　SYBEONCE

メッセージ　関数の呼び出しは 1 回のみです。

考えられる原因　dbsetversion() が複数回呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20208

記号定数 SYBERPNULL

メッセージ `dbprcparam()` の値パラメータは `datalen` パラメータが `0` の場合だけ `NULL` にすることができます。

考えられる原因 dbrpcparam() のパラメータが一致しない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20209

記号定数 SYBERPTXTIM

メッセージ `RPC` パラメータはタイプをテキスト / イメージにすることはできません。

考えられる原因 dbrpcparam() に渡される *type* パラメータは SYBTEXT または SYBIMAGE にできない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20210

記号定数 SYBENEG

メッセージ `Negotiated login attempt failed.`

考えられる原因 サーバへのセキュア・ログインに失敗した。

アクション / 解決法 セキュリティ資格情報と、アプリケーションによって提供されているセキュリティ設定を確認してください。

追加情報

バージョン すべて

20211

記号定数　SYBELBLEN

メッセージ　セキュリティ・ラベルの長さは 256 文字より短くしてください。

考えられる原因　ドキュメント化されていないルーチン dbsetsecurity() に渡されたラベル値が *DB_MAX_LABELLEN* を超えている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20212

記号定数　SYBEUMSG

メッセージ　Unknown message-id in MSG datastream.

考えられる原因　内部 TDS エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20213

記号定数　SYBECAPTYP

メッセージ　CAPABILITY データストリームに予期しない機能タイプがあります。

考えられる原因　内部 TDS エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20214

記号定数 SYBEBNUM

メッセージ `dbstrcpy()` に渡された `numbytes` パラメータが正しくありません。

考えられる原因 無効な *numbytes* パラメータ値が dbstrcpy() ルーチンに渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20215

記号定数 SYBEBBL

メッセージ `dbsetnull()` に渡された `bindlen` パラメータが正しくありません。

考えられる原因 DB-Library ルーチン dbsetnull() の *bindlen* パラメータに負の値が渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20216

記号定数 SYBEBPREC

メッセージ `Illegal precision specified.`

考えられる原因 数値または 10 進数カラムの DBTYPEINFO 構造体に無効な総桁数が指定されている。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20217

記号定数　SYBEBSCALE

メッセージ　無効な桁数が指定されました。

考えられる原因　数値または 10 進数カラムの DBTYPEINFO 構造体に無効な小数点以下桁数が指定されている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20218

記号定数　SYBECDOMAIN

メッセージ

```
Source field value is not within the domain of legal
values.
```

考えられる原因　dbconvert() DB-Library ルーチンを使用した変換のソース値が無効。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20219

記号定数　SYBECINTERNAL

メッセージ　内部変換エラーです。

考えられる原因

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20220

記号定数 SYBEBTYPSRV

メッセージ サーバがサポートしていない `Datatype` です。

考えられる原因 サーバが、このバージョンの TDS に対してこのデータ型を認識しない。

アクション / 解決法 サーバをアップグレードしてください。

追加情報

バージョン すべて

20221

記号定数 SYBEBCSET

メッセージ 文字セットがわかりません。

考えられる原因 サーバによって、認識されていない文字セットが指定された。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20222

記号定数 SYBEFENC

メッセージ パスワード暗号化ができませんでした。

考えられる原因 dbsechandle() によってインストールされている暗号化ハンドラまたはデフォルトの暗号化ハンドラが失敗した。

アクション / 解決法 アプリケーションのコーディングを修正するか、Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20223

記号定数　SYBEFRES

メッセージ

```
Challenge-Response function failed.
```

考えられる原因　ドキュメント化されていない関数 dbresponse() によってインストールされているログイン応答ハンドラまたはデフォルトのログイン応答ハンドラが失敗した。

アクション / 解決法　アプリケーションのコーディングを修正するか、Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20224

記号定数　SYBEISRVPREC

メッセージ　サーバによって返された総桁数が正しくありません。

考えられる原因　10 進数または数値カラムの総桁数値が、有効な総桁数値の範囲外です。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20225

記号定数　SYBEISRVSCL

メッセージ　サーバによって返された小数点以下桁数が正しくありません。

考えられる原因　10 進数または数値カラムの小数点以下桁数値が、有効な小数点以下桁数値の範囲外です。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20226

記号定数 SYBEINUMCL

メッセージ

```
Invalid numeric column length returned by the server.
```

考えられる原因 サーバによって正しくない値が送信された。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20227

記号定数 SYBEIDECCL

メッセージ

```
Invalid decimal column length returned by the server.
```

考えられる原因 サーバによって正しくない値が送信された。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20228

記号定数 SYBEBCMTXT

メッセージ

`bcp_moretext()` は少なくとも `1` つのテキストまたはイメージ・カラムがサーバ・テーブルにあるときだけ使用できます。

考えられる原因 bcp_moretext() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20229

記号定数　SYBEBCPREC

メッセージ　カラム <column>: 不正な精度値を検出しました。

考えられる原因　ホスト・ファイルで検出された精度値が無効だった。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20230

記号定数　SYBEBCBNPR

メッセージ　bcp_bind():varaddr が NULL の場合は prefixlen を 0 にするか、ターミネータを指定しないでください。

考えられる原因　bcp_bind() の呼び出しが正しくない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20231

記号定数　SYBEBCBNTYP

メッセージ　bcp_bind():varaddr が NULL で varlen が 0 より大きい場合、テーブル・カラム・タイプは SYBTEXT または SYBIMAGE で、プログラム変数タイプは SYBTEXT、SYBCHAR、SYBIMAGE、または SYBBINARY でなければなりません。

考えられる原因　bcp_bind() の呼び出しが正しくない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20232

記号定数 SYBEBCSNTYP

メッセージ カラム番号 <colnum>:varaddr が NULL で varlen が 0 より大きい場合、テーブル・カラム・タイプは SYBTEXT または SYBIMAGE で、プログラム変数タイプは SYBTEXT、SYBCHAR、SYBIMAGE、または SYBBINARY でなければなりません。

考えられる原因 bcp_bind() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20233

記号定数 SYBEBCPCTYP

メッセージ bcp_colfmt():table_colnum が 0 の場合、host_type は 0 ではいけません。

考えられる原因 bcp_colfmt() の呼び出しが正しくなかった。*table_colnum* が 0 ということは、カラムがコピーされないことを意味します。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20234

記号定数 SYBEBCVLEN

メッセージ varlen は -1 以上にしてください。

考えられる原因 bcp_bind() または bcp_collen() が -1 未満の *varlen* 値で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20235

記号定数　SYBEBCHLEN

メッセージ　`host_collen は -1 以上にしてください。`

考えられる原因　無効な *host_collen* 値が bcp_colfmt_ps() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20236

記号定数　SYBEBCBPREF

メッセージ　`無効な prefix 値です。有効な値は 0、1、2　または 4 です。`

考えられる原因　無効な *prefixle*n 値が bcp_bind() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20237

記号定数　SYBEBCBPREF

メッセージ　`無効なプレフィックス値です。正しい値は 0、1、2、または 4 です。`

考えられる原因　無効な *host_prefixle*n 値が bcp_colfmt_ps() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20238

記号定数 SYBEBCITBNM

メッセージ `bcp_init():tblname` パラメータは `NULL` ではいけません。

考えられる原因 bcp_init() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20239

記号定数 SYBEBCITBLEN

メッセージ `bcp_init():tblname` パラメータが長すぎます。

考えられる原因 bcp_init() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20240

記号定数 SYBEBCSNDROW

メッセージ 前のローのすべてのテキスト・データが `bcp_moretext()` で送信されないかぎり、`bcp_sendrow()` を呼び出すことはできません。

考えられる原因 送信されていないテキスト / イメージ・データがある。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20241

記号定数　SYBEBPROCOL

メッセージ　bcp プロトコル・エラー：返されたカラム数と受信した実際のカラム数が異なります。

考えられる原因　カラムの内部不一致。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20242

記号定数　SYBEBPRODEF

メッセージ　bcp プロトコル・エラー：デフォルト情報が想定されていましたが、取得されませんでした。

考えられる原因　カラムの内部不一致。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20243

記号定数　SYBEBPRONUMDEF

メッセージ　bcp プロトコル・エラー：想定されたデフォルトの数と実際に受信されたデフォルトの数が異なります。

考えられる原因　カラムのデフォルト内部不一致。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20244

記号定数 SYBEBPRODEFID

メッセージ bcp プロトコル・エラー：デフォルトのカラム ID と実際のカラム ID が異なります。

考えられる原因 カラムのデフォルト内部不一致。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20245

記号定数 SYBEBPRONODEF

メッセージ bcp プロトコル・エラー：デフォルトを持たないカラムのデフォルト値を受信しました。

考えられる原因 カラムのデフォルト内部不一致。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20246

記号定数 SYBEBPRODEFTYP

メッセージ bcp プロトコル・エラー：デフォルト値のデータ型がカラムのデータ型と異なります。

考えられる原因 カラムのデフォルト内部不一致。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20247

記号定数　SYBEBPROEXTDEF

メッセージ　bcp プロトコル・エラー：複数のローのデフォルト情報を受信しました。

考えられる原因　カラムのデフォルト内部不一致。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20248

記号定数　SYBEBPROEXTRES

メッセージ　bcp プロトコル・エラー：予期しない結果のセットを受信しました。

考えられる原因　サーバから余分な結果セットが送信された。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20249

記号定数　SYBEBPROBADDEF

メッセージ　bcp プロトコル・エラー：正しくないデフォルト・カラム ID を受信しました。

考えられる原因　カラムのデフォルト内部不一致。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20250

記号定数 SYBEBPROBADTYP

メッセージ bcp プロトコル・エラー：不明なカラム・データ型です。

考えられる原因 サーバから不明なデータ型が送信された。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20251

記号定数 SYBEBPROBADLEN

メッセージ bcp プロトコル・エラー：受信したデータ型の長さが正しくありません。

考えられる原因 サーバから受信したデータ型の長さが正しくない。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20252

記号定数 SYBEBPROBADPREC

メッセージ bcp プロトコル・エラー：正しくない精度値を受信しました。

考えられる原因 サーバから受信した精度値が正しくない。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20253

記号定数　SYBEBPROBADSCL

メッセージ　`bcp` プロトコル・エラー：正しくない位取り値を受信しました。

考えられる原因　正しくない位取り値をサーバから受信した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20254

記号定数　SYBEBADTYPE

メッセージ　`<routine>` に指定された `type` パラメータの値が正しくありません。

考えられる原因　無効な *type* が dbsechandle() に渡された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20255

記号定数　SYBECRSNORES

メッセージ　カーソル文は結果を生成しませんでした。

考えられる原因　カーソル文が結果を返さなかった。

アクション / 解決法　アクションは不要です。

追加情報

バージョン　すべて

20256

メッセージ・タイプ エラー

記号定数 SYBECRSNOIND

メッセージ

```
One of the tables involved in the cursor statement does
not have a unique index.
```

考えられる原因 不適切なスキーマ。

アクション / 解決法 データベース・スキーマを修正してください。

追加情報

バージョン すべて

20257

記号定数 SYBECRSVIEW

メッセージ

```
A view cannot be joined with another table or a view in
a cursor statement.
```

考えられる原因 カーソル文がビューの結合を実行した。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20258

記号定数 SYBECRSVIIND

メッセージ

```
The view used in the cursor statement does not include
all the unique index columns of the underlying tables.
```

考えられる原因 カーソル文が正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20259

記号定数　SYBECRSORD

メッセージ　'ORDER BY'、'GROUP BY'、または 'HAVING' 句を持つことができるのは、完全なキーセット方式カーソルだけです。

考えられる原因　カーソル文が正しくない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20260

記号定数　SYBECRSBUFR

メッセージ　カーソル API を使用しているときはロー・バッファリングを行わないでください。

考えられる原因　ロー・バッファリングを dbsetopt(…DBBUFFER…) で有効にしている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報　ロー・バッファリングはカーソルと互換性がありません。

バージョン　すべて

20261

記号定数　SYBECRSNOFREE

メッセージ　カーソル API を使用しているときは DBNOAUTOFREE オプションを使用しないでください。

考えられる原因　dbsetopt が dbsetopt(…DBNOAUTOFREE…) で有効になっている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20262

記号定数 SYBECRSDIS

メッセージ カーソル文が使用できない句 'COMPUTE'、'UNION'、'FOR BROWSE'、または 'SELECT INTO' の 1 つを含んでいます。

考えられる原因 カーソル文が正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20263

記号定数 SYBECRSAGR

メッセージ Aggregate functions are not allowed in a cursor statement.

考えられる原因 カーソル文が正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20264

記号定数 SYBECRSFRAND

メッセージ フェッチ・タイプ RANDOM と RELATIVE はキーセット方式カーソルのキーセットのなかでだけ使用できます。

考えられる原因 dbcursorfetch() の呼び出しが正しくない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20265

記号定数　SYBECRSFLAST

メッセージ　フェッチ・タイプ LAST には完全なキーセット方式カーソルが必要です。

考えられる原因　dbcursoropen() が CUR_KEYSET 以外の *scrollopt* で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20266

記号定数　SYBECRSBROL

メッセージ　Backward scrolling cannot be used in a forward scrolling cursor.

考えられる原因　scrollopt CUR_FORWARD でオープンしたカーソルで FETCH_PREV を実行しようとした。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20267

記号定数　SYBECRSFROWN

メッセージ　フェッチするロー番号が正しい範囲外です。

考えられる原因　*firstrow* より前、または *lastrow*. より後のローをフェッチしようとした。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20268

記号定数 SYBECRSBSKEY

メッセージ

```
Keyset cannot be scrolled backward in mixed cursors with
a previous fetch type.
```

考えられる原因 キーセット方式カーソルで *firstrow* より前のローをフェッチしようとした。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20269

記号定数 SYBECRSRO

メッセージ

```
READONLY カーソルではデータのロッキングまたは変更を行うことはできません。
```

考えられる原因 dbcursor() が読み込み専用カーソルで CRS_REFRESH 以外の *optype* により呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20270

記号定数 SYBECRSNOCOUNT

メッセージ

```
dbcursor() で更新または削除を行っているときは DBNOCOUNT オプションを使用しないでください。
```

考えられる原因 DBNOCOUNT オプションが dbsetopt() で設定されている。これは、カーソルの更新 / 削除オペレーションと互換性がありません。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20271

記号定数　SYBECRSTAB

メッセージ

```
Table name must be determined in operations involving
data locking or modifications.
```

考えられる原因　dbcursor() が無効な *table* 値で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20272

記号定数　SYBECRSUPDNB

メッセージ　バインディング・タイプが NOBIND のとき、更新または挿入オペレーションではバインド変数を使用できません。

考えられる原因　dbcursorfetch() が以前 vartype *NOBIND* で呼び出されている。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20273

記号定数　SYBECRSNOWHERE

メッセージ　カーソル更新または挿入では WHERE 句を使用できません。

考えられる原因　dbcursor() が不適切な values 引数で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20274

記号定数 SYBECRSSET

メッセージ カーソル更新または挿入では SET 句が必要です。

考えられる原因 dbcursor() 引数 *values* に SET 句が含まれている必要がある。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20275

記号定数 SYBECRSUPDTAB

メッセージ バインド変数を用いた更新または挿入オペレーションでは 1 つのテーブル・カーソルが必要です。

考えられる原因 dbcursoropenI() stmt 引数が複数のテーブルに影響している。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20276

記号定数 SYBECRSNOUPD

メッセージ 更新または削除オペレーションはどのローも該当しませんでした。

考えられる原因 *CRS_UPDATE* または *CRS_DELETE optype* 引数の dbcursor() がどのローにも影響しなかった。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20277

記号定数　SYBECRSINV

メッセージ　カーソル文が正しくありません。

考えられる原因　dbcursoropen() stmt に SELECT 句が含まれていない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20278

記号定数　SYBECRSNOKEYS

メッセージ　`The entire keyset must be defined for KEYSET type cursors.`

考えられる原因　KEYSET カーソルに含まれるテーブルではキーの存在が必要である。

アクション / 解決法　アプリケーションのコーディングまたはデータベース・スキーマを修正してください。

追加情報

バージョン　すべて

20279

記号定数　SYBECRSNOBIND

メッセージ　カーソル・バインドは `dbcursor` 呼び出しより先に呼び出さなければなりません。

考えられる原因　dbcursor() が NULL *values* 引数で呼び出された。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20280

記号定数 SYBECRSFTYPE

メッセージ 未知の `fetch type` です。

考えられる原因 dbcursorfetch() が未知の *fetchtype* 値で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20282

記号定数 SYBECRSMROWS

メッセージ 複数のローが返されています。`DBNAME` の検索中は `1` つのローだけが要求されます。

考えられる原因 内部カーソル初期化エラー。

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン すべて

20283

記号定数 SYBECRSNROWS

メッセージ `No rows returned, at least one is expected.`

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 予期しない内部エラー。

バージョン すべて

20284

記号定数　SYBECRSNOLEN

メッセージ　ユニーク・インデックスがありません。

考えられる原因　テーブルのユニーク・インデックスがない。

アクション / 解決法　データベース・スキーマを修正してください。

追加情報

バージョン　すべて

20285

記号定数　SYBECRSNOPTCC

メッセージ　OPTCC が検出できません。

考えられる原因　CUR_OPTCC が設定されているカラムを検出できなかった。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20286

記号定数　SYBECRSNORDER

メッセージ　句の順序は FROM、WHERE、および ORDER BY でなければなりません。

考えられる原因　dbcursoropen() に渡された stmt 内の句の順序が正しくない。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20287

記号定数 SYBECRSNOTABLE

メッセージ テーブル名が NULL です。

考えられる原因 無効なテーブルが dbcursor() に渡された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20288

記号定数 SYBECRSNUNIQUE

メッセージ このビューに関連付けされているユニーク・キーはありません。

考えられる原因 このビューの基になっているテーブルに関連付けられているキーがない。

アクション / 解決法 データベース・スキーマを修正してください。

追加情報

バージョン すべて

20289

記号定数 SYBECRSVAR

メッセージ このバインドに関連付けられている有効なアドレスはありません。

考えられる原因 dbcursorbind() が *vartype NOBIND* と無効な *pvaraddr* で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20290

記号定数　SYBENOVALUE

メッセージ　セキュリティ・ラベルには名前と値の両方が必要です。

考えられる原因　セキュリティ・ラベル・ハンドラが *namelen* または *valuelen* のいずれかを値 <= 0 に設定した。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20291

記号定数　SYBEVOIDRET

メッセージ　リターン・パラメータのタイプは `SYBVOID` 以外です。

考えられる原因　dbrpcparam() がリターン・パラメータを *SYBVOID* として定義しようとした。

アクション / 解決法　アプリケーションのコーディングを修正してください。

追加情報

バージョン　すべて

20292

記号定数　SYBECLOSEIN

メッセージ　インタフェース・ファイルをクローズできません。

考えられる原因　内部クローズ・エラー。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20293

記号定数 SYBEBOOL

メッセージ BOOL 型パラメータの値は TRUE または FALSE です。

考えられる原因 bcp_options() がオプション *BCPLABELED* で呼び出され、**value* が TRUE または FALSE に設定されていない。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20294

記号定数 SYBEBCPOPT

メッセージ <option> オプションは、バルク・コピー・オペレーションの実行中には呼び出せません。

考えられる原因 バルク・コピー・オペレーションの実行中に bcp_options() がオプション *BCPLABELED* で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20295

記号定数 SYBEERRLABEL

メッセージ セキュリティ・ラベル・ハンドラから返された値が不正です。

考えられる原因 セキュリティ・ラベル・ハンドラが *DMORELABEL*、*DBENDLABLE*、または *DBERRLABEL* 以外の値を返した。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20296

記号定数　SYBEATTNACK

メッセージ　サーバからのアテンション承認までの待機時間がタイムアウトしました。

考えられる原因　サーバが dbcancel() リクエストに応答しなかった。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20297

記号定数　SYBEBBFL

メッセージ　サーバへのバルク・コピーでバッチが失敗しました。

考えられる原因　バルク・コピーでサーバがエラーを報告した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20298

記号定数　SYBEDCL

メッセージ　ディレクトリ制御レイヤ（DCL）エラーが発生しました。

考えられる原因　ディレクトリ・サービスの読み込み中にエラーが発生した。

アクション / 解決法　Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン　すべて

20299

記号定数 SYBECS

メッセージ CS コンテキスト・エラーが発生しました。

考えられる原因 廃止

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報

バージョン なし

20300

記号定数 SYBEVERENV

メッセージ SYBOCS_DBVERSION に無効な値が使用されました。

考えられる原因 dbsetversion() が無効な *version* で呼び出された。

アクション / 解決法 アプリケーションのコーディングを修正してください。

追加情報

バージョン すべて

20301

記号定数 SYBCOPNOV

メッセージ dbcursoropen():scrollopt と nrows の乗算結果がオーバフローします。

考えられる原因

アクション / 解決法 Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

追加情報 内部カーソル・エラー。

バージョン 15.7 以降

20302

記号定数	SYBEINTOVFL
メッセージ	DB-Library 内部エラー：算術演算子結果が整数オーバフローしました。
考えられる原因	文字セット変換を行うにはバッファのサイズが小さすぎた。
アクション / 解決法	Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。
追加情報	
バージョン	15.7 以降

索引

数字

A

B

C

D

E

I

L

M

N

O

R

S

T

W

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

つ

て

と

に

ね

の

は

ひ

ふ

へ

ほ

め

も

ゆ

よ

り

る

れ

ろ

わ