SYBASE®

Embedded SQL™/COBOL プログラマーズ・ガイド

Open Client™

15.5

ドキュメント ID：DC37698-01-1550-01
改訂：2009 年 10 月

マニュアルの注文

マニュアルの注文を承ります。ご希望の方は、サイベース株式会社営業部または代理店までご連絡ください。マニュアルの変更は、弊社の定期的なソフトウェア・リリース時にのみ提供されます。

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

目次

はじめに

このマニュアルでは、COBOL アプリケーションで Embedded SQL™ と Embedded SQL プリコンパイラを使う方法について説明します。Embedded SQL は Transact-SQL® のスーパーセットで、COBOL および C で記述されたアプリケーション・プログラムに Transact-SQL 文を埋め込みます。

このマニュアルの情報は、各プラットフォームに共通です。プラットフォーム特有の Embedded SQL の使い方については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

対象読者

このマニュアルは、Embedded SQL の概念と使用に関心のあるアプリケーション開発者を対象としています。このマニュアルの対象読者には、次の知識が必要です。

- 『ASE リファレンス・マニュアル』の内容を理解していること
- COBOL のプログラミング経験があること

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 概要」では、Embedded SQL の簡単な概要を示し、Embedded SQL の利点と機能について説明します。
- 「第 2 章 一般情報」では、Embedded SQL プログラムおよび Embedded SQL を使用したプログラミングの一般的な規則について説明します。
- 「第 3 章 Adaptive Server Enterprise との通信」では、SQLCA、SQLCODE、SQLSTATE による通信領域の確立と使用方法について説明します。また、通信領域で使うシステム変数についても説明します。
- 「第 4 章 変数の使い方」では、Embedded SQL 内でのホスト変数やインジケータ変数の宣言および使用方法について説明します。また、配列および「データ型」の変換についても説明します。
- 「第 5 章 Adaptive Server Enterprise との接続」では、Embedded SQL を使ってアプリケーション・プログラムを Adaptive Server® Enterprise および通常のデータ・サーバに接続する方法について説明します。
- 「第 6 章 Transact-SQL 文の使い方」では、Embedded SQL アプリケーション・プログラム内で Transact-SQL を使う方法について説明します。ここでは、配列およびバッチを使用したローの選択方法と、Transact-SQL 文をグループ化する方法を説明します。

- 「第 7 章 動的 SQL の使い方」では、アプリケーションのユーザが、実行時に対話形式で入力できる Embedded SQL 文の作成方法について説明します。
- 「第 8 章 エラーの処理」では、リターン・コードと、エラーの検出および対処のための Embedded SQL プリコンパイラ機能について説明します。
- 「第 9 章 Embedded SQL 文：リファレンス・ページ」では、個々の Embedded SQL 文のリファレンス・ページを提供します。
- 「第 10 章 Open Client/Server 設定ファイル」では、Embedded SQL での外部設定ファイルの使用方法について説明します。
- 「付録 A プリコンパイラの警告とエラー・メッセージ」では、プリコンパイラおよび実行時メッセージのリストを掲載します。
- 「用語解説」では、このマニュアルで使用される用語の定義を説明します。

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 リリース・ノート Microsoft Windows 版』には、Open Server に関する重要な最新情報が記載されています。
- 『Software Developer's Kit リリース・ノート Microsoft Windows 版』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 バージョン 6.05 および 7.0』には、jConnect に関する重要な最新情報が記載されています。
- この『Open Client/Server 設定ガイド Microsoft Windows 版』では、次のプラットフォームで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 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 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/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 Microsoft Windows 版』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
- 『jConnect for JDBC インストール・ガイド バージョン 6.05』では、jConnect for JDBC のインストール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』(Windows および Linux 版) では、Windows、Linux、および Apple Mac OS X 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、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。

その他の情報

Sybase® Getting Started CD、SyBooks™ CD、Sybase Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。Eclipse ベースの SyBooks ブラウザを使用すれば、使いやすい HTML 形式のマニュアルにアクセスできます。

  一部のマニュアルは PDF 形式で提供されています。これらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の *README.txt* ファイルを参照してください。

- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使用してアクセスできます。また、製品マニュアルのほか、EBFs/Maintenance、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Technical Library Product Manuals Web サイトにアクセスするには、Product Manuals (http://www.sybase.com/support/manuals/) にアクセスしてください。

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 を指定します。(http://www.sybase.com/support/techdocs/)

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 を指定します。(http://certification.sybase.com/)

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 を指定します。(http://www.sybase.com/support/techdocs/)

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 Sybase Support Page を指定します。(http://www.sybase.com/support)

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

キー	定義
command	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。
variable	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

概要

この章では、Embedded SQL および Embedded SQL プリコンパイラの概要について説明します。

Embedded SQL の概要

Embedded SQL は Transact-SQL のスーパーセットで、C および COBOL で記述されたアプリケーション・プログラムに Transact-SQL 文を埋め込みます。

Embedded SQL を使用して、Adaptive Server Enterprise のデータにアクセスしたりデータを更新したりするプログラムを作成できます。Embedded SQL のプログラマは、C や COBOL のような一般的なプログラミング言語で記述されたアプリケーション・プログラムの中に直接 SQL 文を記述します。プリプロセッシング・プログラムである Embedded SQL プリコンパイラは、完成したアプリケーション・プログラムを処理して、ホスト言語コンパイラがコンパイル可能なプログラムにします。このプログラムは実行される前に Open Client Client-Library にリンクされます。

Embedded SQL は、Adaptive Server Enterprise へアクセスするために Sybase が提供する 2 つのプログラミング方法のうちの 1 つです。もう 1 つのプログラミング方法は、呼び出しレベルのインタフェースです。呼び出しレベルのインタフェースでは、Client-Library 呼び出しを直接アプリケーション・プログラムに記述してから Client-Library とリンクします。

Embedded SQL 文は「ホスト・プログラム」の任意の場所に、ホスト言語の文と混在して記述することができます。すべての Embedded SQL 文は exec sql というキーワードで開始し、end-exec で終了しなければなりません。

Adaptive Server Enterprise から取得したデータを保管して、または select 文の where 句のように Embedded SQL 文中のパラメータとして、「ホスト変数」を Embedded SQL 文中に使用できます。「動的 **SQL**」では、ホスト変数は Embedded SQL 文に対するテキストも含むことができます。

Embedded SQL の特長

Embedded SQL は、呼び出しレベル・インタフェースに比べ、次のようないくつかの有利な点があります。

- Embedded SQL は、Transact-SQL に、アプリケーション内で使いやすくするための機能を単に追加したものなので、操作が簡単です。
- Embedded SQL は、ANSI/ISO 標準のプログラミング言語です。
- 呼び出しレベルの場合と比べて、少ないコーディングで同じ結果が得られます。
- Embedded SQL は、異なるホスト言語でも本質的に同じです。プログラム規約や構文の変更はそれほどありません。したがって、異なる言語でアプリケーションを作成する場合にも、新たに構文を覚える必要はありません。
- プリコンパイラは、Embedded SQL 文のストアド・プロシージャを作成して実行時間を最適化できます。

Embedded SQL での Transact-SQL のサポート

print、raiserror、readtext、writetext といった例外を除いて、すべての Transact-SQL 文、関数、フロー制御言語が Embedded SQL で有効です。アプリケーションのデバッグを容易にするため、Transact-SQL で Embedded SQL アプリケーションの対話的なプロトタイプを開発し、それをアプリケーションに組み込むことができます。

Adaptive Server Enterprise データ型のほとんどは、同等のデータ型が Embedded SQL にもあります。また、ホスト言語のデータ型も Embedded SQL の中で使用できます。ホスト言語のデータ型が Adaptive Server Enterprise のデータ型に完全に一致しない場合には、自動的にデータ型が変換されます。

Transact-SQL でリテラル引用符を使用できる場所であれば、Embedded SQL 文にホスト言語の変数を使用できます。一重引用符 (‘) または二重引用符 (“) のどちらかでリテラルを囲みます。引用符を含むリテラルの区切り方については、『Adaptive Server Enterprise リファレンス・マニュアル』を参照してください。

Embedded SQL には、Transact-SQL には含まれていない次のような機能があります。

- ホスト言語のデータ型と Adaptive Server Enterprise のデータ型の間の自動データ型変換
- ユーザが実行時に SQL 文を定義できる動的 SQL

- Adaptive Server Enterprise とアプリケーション・プログラム間の通信を可能にする *SQLCA*、*SQLCODE*、*SQLSTATE*。この 3 つのエンティティには、Adaptive Server Enterprise が生成するエラー、警告、情報メッセージのコードが含まれています。
- 実行中のエラーを検出するリターン・コードのテスト・ルーチン

はじめる前に

プリコンパイラはランタイム・ライブラリとして Client-Library を使用するので、Client-Library バージョン 12.5 以降がインストールされていることを確認してから、プリコンパイラを起動してください。また、Adaptive Server Enterprise バージョン 12.5 以降がインストールされていることも確認してください。不足している製品がある場合は、「システム管理者」に連絡してください。

オペレーティング・システムのプロンプトで、適切な「コマンド」を発行してプリコンパイラを起動します。詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

プリコンパイラ・コマンドには、プリコンパイラのオプションを決定するフラグがあります。これらのオプションを使用すると、入力ファイル、ログイン・ユーザ名とパスワード、HA フェールオーバの起動、プリコンパイラのモードなどが指定できます。Embedded SQL アプリケーションのプリコンパイル、コンパイル、リンクに関するオペレーティング・システムごとの情報については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

プログラム例の利用法

このマニュアルのプログラム例では、「pubs2 データベース」を使用しています。プログラム例を実行するには、Transact-SQL の use 文で pubs2 データベースを指定します。

この製品には、いくつかのオンライン・プログラム例が添付されています。これらのプログラム例の実行方法については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

旧バージョンとの互換性

プリコンパイラは、ANSI SQL-89 準拠のプリコンパイラと互換性があります。しかし、システムには ANSI に準拠していない古いバージョンの Embedded SQL で作成されたアプリケーションも存在するかもしれません。現在のプリコンパイラは以前のバージョンとほぼ同じ Embedded SQL 文を使いますが、その処理方法は異なります。

以前のバージョンのプリコンパイラ用に作成されたアプリケーションを移行するには、次の手順を実行します。

1 次の SQL 文とキーワードは System 11 以降ではサポートしていないので、アプリケーションから取り除きます。

- release *connection_name*
- recompile
- noparse
- noproc
- pcoptions sp_syntax
- cancel

release 文はプリコンパイラ・エラーの原因となります。つまり、プリコンパイラは他のキーワードを無視します。cancel 文はランタイム・エラーの原因となります。

2 プリコンパイラを使用して、アプリケーションをもう一度プリコンパイルします。

Embedded SQL プログラムの作成と実行

次に、Embedded SQL アプリケーション・プログラムを作成、実行する手順を示します。

1 アプリケーション・プログラムを記述し、Embedded SQL 文と変数宣言を埋め込みます。

2 アプリケーションを *.pco* ファイルに保存します。

3 アプリケーションをプリコンパイルします。重大なエラーが発生しなければ、プリコンパイラはアプリケーション・プログラムを含むファイルを生成します。このファイルには、拡張子だけを変えて元のソース・ファイルと同じ名前が付けられます。この拡張子は、使用している COBOL コンパイラの要件に応じて異なります。詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

4 新しく生成されたソース・コードを標準の COBOL プログラムと同じようにコンパイルします。

5 必要に応じて、コンパイルされたコードを必要なライブラリとリンクします。

6　ストアド・プロシージャを生成するためのプリコンパイラ・オプションを指定した場合は、isql を使用して生成されたスクリプトを実行して Adaptive Server Enterprise にロードします。

7　標準の COBOL プログラムと同じようにアプリケーション・プログラムを実行します。

プリコンパイラによるアプリケーションの処理

Embedded SQL プリコンパイラは、Embedded SQL 文を COBOL のデータ宣言文と呼び出し文に変換します。プリコンパイル後に作成されるソース・プログラムは、普通の COBOL プログラムと同様にコンパイルできます。

プリコンパイラは、アプリケーションを 2 段階で処理します。1 段階目の処理は、Embedded SQL 文と変数宣言を解析し、構文をチェックして、エラーを検出した場合にはメッセージを表示します。重大なエラーが検出されなければ、次のような 2 段階目の処理に移ります。

- “SQL--” で始まるプリコンパイラ変数の宣言を加えます。混乱を避けるため、自分で作成する変数には “SQL” で始まる名前を付けないでください。
- 元の Embedded SQL 文のテキストをコメントに変換します。
- プリコンパイル・コマンド・ラインにストアド・プロシージャの生成および呼び出しオプションを付けた場合は、ストアド・プロシージャを生成し、呼び出します。
- Embedded SQL 文をランタイム・ルーチンに対する呼び出しに変換します。
- プリコンパイラは 3 種類のファイルを作成します。「ターゲット・ファイル」、オプションの「リスティング・ファイル」、オプションの「**isql** スクリプト・ファイル」の 3 つです。

注意　プリコンパイラのコマンド・ライン・オプションの詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

複数の Embedded SQL ソース・ファイル

Embedded SQL を使用したアプリケーションが、複数のソース・ファイルで構成される場合には、次の規則が適用されます。

- 接続名はユニークで、アプリケーション全体でグローバル
- カーソル名は接続ごとでユニーク
- 準備文名は接続に対してグローバル
- 動的記述子はアプリケーションに対してグローバル

プリコンパイラの生成ファイル

「ターゲット・ファイル」は、元の入力ファイルに似ていますが、すべての SQL 文が実行時呼び出しに変換されています。

リスティング・ファイルには、入力ファイルおよび入力ファイルの元の文に加えて、付加的な情報、警告、またはエラー・メッセージなどが含まれています。

isql スクリプト・ファイルには、プリコンパイラが生成したストアド・プロシージャが含まれています。ストアド・プロシージャは、Transact-SQL で記述されています。

グループ要素参照

Embedded SQL COBOL プリコンパイラは、exec sql 文のホスト変数の COBOL 言語構造構文をサポートしています。たとえば、基本構造データ項目 C を含む構造体 B を含む構造体 A の場合、A.B.C は C OF B OF A と同じです。

要素とピリオド (.) の間には空白スペースを使用することができます。C OF A .B のような、2 つの構文の混合は無効です。次に、グループ要素参照の例を示します。

```
 EXEC SQL BEGIN DECLARE SECTION END-EXEC .

      01    AU-IDPIC X(15).
      01    GROUP1.
      05    GROUP2.
            10  LNAME PIC X(40).
            10  FNAME PIC X(40).
            10  PHONE PIC X(15).

EXEC SQL END DECLARE SECTION END-EXEC.
                     ...
EXEC SQL USE pubs2 END-EXEC.

MOVE "724-80-9391" TO AU-ID.
EXEC SQL SELECT INTO :GROUP1. GROUP2.LNAME,
                              :GROUP2.FNAME, :PHONE
                   au_lname, au_fname, phone
                   FROM authors
                   WHERE au_id = :AU-ID END-EXEC.
DISPLAY "LAST  NAME = ", LNAME.
DISPLAY "FIRST NAME = ", FNAME.
DISPLAY "PHONE #    = ", PHONE.

* This SELECT does the same thing. You can use :GROUP1.GROUP2
* which refers to the entire structure, but partially qualified
* names such as :LNAME OF GROUP1 do not work.

EXEC SQL SELECT INTO :GROUP1. GROUP2
             au_lname, au_fname, phone
                   FROM authors
                    WHERE au_id = :AU-ID END-EXEC.

DISPLAY "---------------------------------------".
DISPLAY "GROUP LISTING FROM ENTIRE STRUCTURES".
DISPLAY "---------------------------------------".
       DISPLAY "LAST  NAME = ", LNAME.
       DISPLAY "FIRST NAME = ", FNAME.
       DISPLAY "PHONE #    = ", PHONE.

                   ...
```

第 2 章

一般情報

この章では、Embedded SQL の一般的な情報について説明します。

Embedded SQL プログラムの 5 つのタスク

Embedded SQL プログラムはホスト言語のコードを含めることに加え、さらに 5 つのタスクを実行します。正しくプリコンパイル、コンパイル、実行するには、Embedded SQL プログラムはこれら 5 つのタスクをすべて実行しなければなりません。これら 5 つのタスクについては、あとの章で説明します。

1 SQLCA、SQLCODE、または SQLSTATE を使用して SQL 通信を確立します。

SQL 通信領域 (SQLCA、SQLCODE、または SQLSTATE) を設定して、アプリケーション・プログラムと Adaptive Server Enterprise 間の通信パスを提供します。これらの構造体には、Adaptive Server Enterprise と Client-Library が生成するエラー・メッセージ、警告メッセージ、情報メッセージのコードが入っています。「第 3 章 Adaptive Server Enterprise との通信」を参照してください。

2 変数を宣言します。

Embedded SQL 文で使用されるホスト変数をプリコンパイラに示します。「第 4 章 変数の使い方」を参照してください。

3 Adaptive Server Enterprise に接続します。

アプリケーションを Adaptive Server Enterprise に接続します。「第 5 章 Adaptive Server Enterprise との接続」を参照してください。

4 Transact-SQL 文を Adaptive Server Enterprise に送信します。

Transact-SQL 文を Adaptive Server Enterprise に送信して、データの定義と操作を行います。「第 6 章 Transact-SQL 文の使い方」を参照してください。

5 エラーを処理してコードを返します。

SQLCA、SQLCODE、または SQLSTATE を使用して Client-Library と Adaptive Server Enterprise が返したエラーの処理とレポートを行います。「第 8 章 エラーの処理」を参照してください。

簡単な Embedded SQL プログラム

次に、簡単な Embedded SQL プログラムを示します。ここでは、このプログラムをすべて理解する必要はありません。プログラムを示した目的は、Embedded SQL プログラムの構成を説明するためです。詳細は、あとの章で説明します。

```
 IDENTIFICATION DIVISION.
  PROGRAM-ID. EXAMPLE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

 * Communicating with Adaptive Server Enterprise -
Chapter 3
      exec sql include sqlca end-exec.

 * Declaring variables  -  Chapter 4
 exec sql begin declare section end-exec
 01  MY-ID            PIC X(30).
 01  MYPASS           PIC X(30).
 01  MYSERVER         PIC X(30).
 exec sql end declare section end-exec.

 PROCEDURE DIVISION.
 MAIN-SECTION.
 PARA-1.

 * Initializing error-handling routines - Chapter 8

 exec sql whenever sqlerror perform ERR-PARA
 through ERR-PARA-END end-exec.
* Connecting to Adaptive Server Enterprise - Chapter 5

     DISPLAY "PLEASE ENTER USER-ID".
     ACCEPT MY-ID.
     DISPLAY "PLEASE ENTER PASSWORD".
     ACCEPT MYPASS.
     DISPLAY "SERVER TO USE?".
     ACCEPT MYSERVER.
     exec sql connect :MY-ID identified by :MYPASS
     using :MYSERVER end-exec.
*Issuing Transact-SQL statements  -  Chapter 6
 exec sql update alltypes set account = account * 2 end-exec.
exec sql commit work end-exec.
```

```
*Closing connection to the server  -  Chapter 5

 exec sql disconnect default end-exec.
 STOP RUN.

Error-handling routine  -  Chapter 8

  ERR-PARA.
      DISPLAY     " ERROR CODE "    SQLCODE
        " ERROR MESSAGE:"          SQLERRMC.
  ERR-PARA-END.
END PROGRAM.
```

Embedded SQL の一般的な規則

Embedded SQL 文には、次のような規則が適用されます。

- Embedded SQL 文は、次のキーワードで始めます。

  exec sql

- Embedded SQL のカラム 7 には継続文字、カラム 8 ～ カラム 72 にはトークンが必要です。文の最初には exec sql を入れます。
- 生成された宣言セクションを正しく揃えて、コンパイラの警告を避けるには、exec sql begin declare section 文を、データ宣言用の正しいカラムに揃える必要があります。
- Embedded SQL のキーワードには、大文字と小文字の区別がありません。exec sql、EXEC SQL、Exec Sql などのほか、大文字と小文字を自由に混在させることができます。このマニュアルでは、Embedded SQL のキーワードは次のように小文字で統一して表記しています。次に例を示します。

  ```
  exec sql commit work end-exec.
  ```

- Embedded SQL 文はすべてキーワード end-exec で終わります。プログラムの構文または論理で必要な場合は、end-exec の後にピリオドを入れます。たとえば、次のコードでは、COBOL の段落をピリオドで終了しなければならないため、end-exec の後にピリオドが必要です。

  ```
  PARA-1.
       IF SQLCODE = 0
           exec sql commit work end-exec.
   PARA-2.
  ```

次の例では、COBOL が if と else の間のピリオドを認めないため、最初の end-exec の後にピリオドがありません。

```
IF SQLCODE NOT = 0
   exec sql rollback transaction disconnect
   end-exec
 ELSE
   exec sql commit work end-exec.
```

- Embedded SQL 文は、数行にわたって記述できます。end-exec は、文のコードの最終行の末尾、または最終行の次の新しい行に入れてください。

文の位置

アプリケーション・プログラム内で COBOL の文が有効な場所であれば、どこにでも Embedded SQL 文を使用できます。ただし、Embedded SQL 文は、プログラムの DATA DIVISION の WORKING-STORAGE SECTION が定義されるまで作成できません。したがって、FILE SECTION などには、Embedded SQL 文を入れることができません。

コメント

Embedded SQL や COBOL の文の内部にコメントを入れる場合、次の 3 つの規約のうちのいずれかに従います。

Transact-SQL の規約は、次のとおりです。

```
/* comments */
```

COBOL の規約は、次のとおりです。

* (in column 7)

ANSI の規約は、次のとおりです。

```
-- comments
```

SQL 文の外部で書かれるコメントは、COBOL の規約に準拠します。

識別子

識別子は、アプリケーション内でプロシージャ名またはデータ名として使用されます。識別子を複数行に分割することはできません。

引用符

Embedded SQL 文中のリテラル文字列は、一重引用符または二重引用符で囲みます。文字列を二重引用符で始めたときは、二重引用符で終了します。文字列を一重引用符で始めたときは、一重引用符で終了します。

予約語

COBOL、Transact-SQL、または Embedded SQL の予約語は、言語本来の意味以外に使用しないでください。

Embedded SQL のキーワードは、大文字でも小文字でも、両者が混在していてもかまいません。このマニュアルでは、Embedded SQL のキーワードを小文字で示します。

変数の命名規約

Embedded SQL の変数は、COBOL の命名規約に準拠します。一対の引用符内には、変数名を入れないでください。変数名が実際の値で置き換えられる際に、自動的に適切な引用符が挿入されます。プリコンパイラがアプリケーションを解析する際に、プリコンパイラは変数に対する宣言を追加します。これらの宣言は “SQL--” で始まります。混乱を避けるために、“SQL” で始まる変数名は使用しないようにしてください。

スコープの規則

Embedded SQL 文と、プリコンパイラが生成する文のスコープは、ホスト言語のスコープ規則に準拠します。ただし、whenever 文とカーソル名は例外です。

文のバッチ処理

Transact-SQL と同じように、複数の SQL 文は、1 つの exec sql 文にまとめてバッチ処理できます。アプリケーションが実行されるたびに同じ Transact-SQL を実行しなければならないときは、バッチ処理を行えば便利で効率もよくなります。

たとえば、あるアプリケーションでは、起動時にテンポラリ・テーブルとテンポラリ・インデックスを作成するとします。これらの文は、1 つのバッチで送信できます。文のバッチ処理の規則については、『ASE リファレンス・マニュアル』を参照してください。

次のような制約が、文のバッチ処理に適用されます。

- バッチ内の文からプログラムに結果を返すことはできません。つまり、バッチに select 文は使えません。
- バッチ内のすべての文は、有効な Transact-SQL 文でなければなりません。declare cursor や prepare といった Embedded SQL 文は、バッチ内では使えません。
- Transact-SQL バッチとまったく同じ規則が Embedded SQL バッチにも適用されます。たとえば、use database 文は Embedded SQL バッチでは使えません。

Embedded SQL の構文

表 2-1 に、Embedded SQL 文の有効な構文を示します。

表 *2-1: Embedded SQL* の構文

begin declare section	drop trigger
begin tran	drop view
begin work	dump database
checkpoint	dump tran
close *cursor_name*	end declare section
commit tran	exec *procedure_name*
commit work	execute name
connect	execute immediate
create database	fetch *cursor_name*
create default	grant
create table	include sqlca または *file*
create index	insert
create unique index	open *cursor_name*
create clustered index	prepare *statement_name*
create nonclustered index	revoke
create unique clustered index	rollback tran
create unique nonclustered index	rollback work
create proc	select
create rule	set
create trigger	truncate
create view	update
declare cursor	use
delete	whenever *condition action*
disconnect	
drop table	
drop default	
drop index	
drop proc	
drop rule	

第 3 章

Adaptive Server Enterprise との通信

この章では、アプリケーション・プログラムがどのようにして Adaptive Server Enterprise からステータス情報を受け取るかについて説明します。次のトピックで構成されています。

通信パスを作成し、Adaptive Server Enterprise からアプリケーションへの通信で使用されるシステム変数を宣言するには、次のうちのどれかを作成してください。

- SQLCODE を持つ SQL 通信領域 (SQLCA)
- スタンドアロンの SQLCODE long integer
- SQLSTATE 文字配列

SQLCODE、SQLCA、SQLSTATE は、Adaptive Server Enterprise からアプリケーションへの通信時に使用するシステム変数です。

Adaptive Server Enterprise は、Embedded SQL 文を実行するたびに、リターン・コードを SQLCA、SQLCODE、または SQLSTATE 内に格納します。アプリケーション・プログラムは、その変数にアクセスして、文の成功または失敗を調べることができます。

注意 プリコンパイラは、自動的に SQLCA、SQLCODE、SQLSTATE 変数を設定します。これらの変数はデータベースへのランタイム・アクセスに非常に重要であり、ユーザがこれを初期化または変更する必要はありません。

エラーの検出と処理、複数のエラー・メッセージやその他のリターン・コードに関する詳細については、「第 8 章 エラーの処理」を参照してください。

スコープの規則：SQLCA、SQLCODE、および SQLSTATE

アプリケーション・プログラムで COBOL の変数を宣言できる場所であれば、どこでも SQLCA を宣言できます。この構造体のスコープは、COBOL のスコープ規則に従います。

使用しているファイル内で SQLCA、SQLCODE、または SQLSTATE を宣言している場合、各変数はそのファイルにあるすべての実行可能な Embedded SQL 文のスコープ内になければなりません。プリコンパイラは各 Embedded SQL 文に対して、これらのステータス変数のそれぞれを設定するコードを生成します。そのため、変数がスコープ内にない場合は、生成されたコードはコンパイルされません。

SQLCA の宣言

警告！ SQLCODE や SQLCA よりも SQLSTATE を使用する方が望ましいのですが、このバージョンのプリコンパイラでは SQLCODE のみをサポートしています。今後のバージョンでは SQLCA および SQLSTATE の両方を完全にサポートする予定です。

使用するアプリケーション・プログラムの WORKING-STORAGE SECTION で SQLCA を宣言してください。次に、SQLCA の宣言の構文を示します。

```
exec sql include sqlca [is external] [is global]
    end-exec.
```

複数の SQLCA

1 つのファイルには複数の COBOL プログラムを含むことができるので、SQLCA が複数ある場合があります。ただし、各 SQLCA は、別々の WORKING-STORAGE SECTION に配置される必要があります。

SQLCA 変数

プリコンパイラは include sqlca 文を検出すると、アプリケーション・プログラムに SQLCA 構造体の宣言を挿入します。SQLCA とはプリコンパイラが定める「システム変数」を持つデータ構造体であり、各システム変数は独立してアクセスされます。

SQLCA 変数は、直前に実行した Embedded SQL 文のステータスに関する情報をアプリケーション・プログラムへ渡します。

表 3-1 では、Adaptive Server Enterprise が生成するステータス情報、リターン・コード、エラー・コード、エラー・メッセージを保持する SQLCA 変数について説明します。

表 3-1: *Adaptive Server Enterprise SQLCA* 変数

変数	データ型	説明
SQLCAID	PIC X(8)	“SQLCA” を含む文字列
SQLCABC	PIC S9(9) COMP	SQLCA の長さ
SQLCODE	PIC S9(9) COMP	直前に実行した SQL 文のリターン・コードを含む。リターン・コードの定義については、表 3-2 (18 ページ) の SQLCODE 値を参照。
SQLWARN0 から *SQLWARN7*	PIC X(1)	警告フラグ。個々のフラグは警告が発行されたかどうかを示す。“W” は警告あり、スペースは警告なしを意味する。SQLWARN フラグについては第 8 章を参照。
SQLERRMC	PIC X(256)	エラー・メッセージ
SQLERRML	PIC S9(9) COMP	エラー・メッセージの長さ
SQLERRP	PIC X(8)	エラー／警告を検出したプロシージャ
SQLERRD	PIC S9(9) COMP OCCURS 6 TIMES	エラー／警告の詳細。SQLERRD(3) は影響を受けたローの数。

SQLCA 変数へのアクセス

前の項で挙げた SQLCA 変数は、エラーとリターン・コードについてより多くの情報を提供し、アプリケーションのデバッグと通常の処理に役立ちます。

警告！ SQLCODE は SQLCA 構造体内のフィールドであるため、SQLCODE と SQLCA の両方を SQLCODE として定義しないでください。

SQLCA 内の SQLCODE

Adaptive Server Enterprise は実行ごとに SQLCODE を更新するので、アプリケーションは文の実行ごとに SQLCODE をテストしなければなりません。SQLCODE をテストするには、通常、「第 8 章 エラーの処理」で説明する whenever 文を使用します。

次に、SQLCODE を使用する例を示します。

```
IF SQLCODE = 100
 PERFORM END-DATA-PARA.
```

または

```
DISPLAY "SQL status code is" SQLCODE.
```

スタンドアロン領域としての SQLCODE の宣言

注意 SQLCODE や SQLCA よりも SQLSTATE を使用する方が望ましいのですが、このバージョンのプリコンパイラでは SQLCODE のみをサポートしています。今後のバージョンでは SQLCA および SQLSTATE の両方を完全にサポートする予定です。

SQLCA を作成する代わりに、SQLCODE を独立して使用する方法があります。これには、直前に実行した SQL 文のリターン・コードが含まれています。SQLCODE をスタンドアロン領域として宣言する利点は、コードを実行する速度が向上する点です。SQLCA が持つ他の情報を見る必要がなく、リターン・コードだけを知りたい場合は、SQLCODE を使用することもできます。

しかし、SQLCODE は Embedded SQL の前のバージョンとの互換性を保つだけの機能なので、SQLCODE の実行速度の方が高速ですが、SQLCODE ではなく、SQLSTATE の使用をおすすめします。

警告！ SQLCODE を宣言セクションで宣言しないでください。

次は、スタンドアロン領域として SQLCODE を宣言する例です。

```
01  SQLCODE S9(9)    COMP.
 exec sql open cursor pub_id   end-exec.

PARAGRAPH-1:
    exec sql fetch pub_id into :PUB_NAME  end-exec.
 IF SQLCODE = 0 GOTO PARAGRAPH-1.
```

SQLCODE が示すエラーのデバッグの詳細については、「第 8 章 エラーの処理」を参照してください。

表 3-2 に SQLCODE の値を示します。

表 *3-2: SQLCODE* の値

値	説明
0	文の実行が成功した。
-n	エラーが発生した。Server-Library または Client-Library のエラー・メッセージを参照。*-n* は、エラーまたは例外の数を表す。
+100	データが存在しない、fetch のあとにローが残っていない、または update、delete、insert の探索条件に合うローが存在しない。

SQLSTATE の使用

警告！ SQLCODE や SQLCA よりも SQLSTATE を使用する方が望ましいのですが、このバージョンのプリコンパイラでは SQLCODE のみをサポートしています。今後のバージョンでは SQLCA および SQLSTATE の両方を完全にサポートする予定です。

SQLSTATE はステータス・パラメータです。そのコードは、直前に実行された文のステータスを示します。つまり、その文が正常に完了したか、またはその文の実行中にエラーが発生したかを示します。

次に、SQLSTATE の宣言の例を示します。

```
WORKING-STORAGE SECTION.
01 SQLSTATE PIC x(5)
 . . .
exec sql whenever sqlerror perform ERR-PARA
 end-exec
 . . .
ERR-PARA.
    IF  sqlstate = "ZD000" or
         sqlstate = "ZE000" or
         sqlstate = "ZF000" or
         sqlstate = "ZG000" or
         sqlstate = "ZH000"
         DISPLAY "Unexpected results were ignored"
     ELSE
        IF sqlstate = "08001" or sqlstate = "08000"
              DISPLAY "Connection failed-quitting"
              STOP RUN
         ELSE
             DISPLAY "A non-results, non-connect
             – error occurred"
        END_IF
    END_IF
```

表 3-3 に、SQLSTATE の値を示します。

表 *3-3: SQLSTATE* の値

値	説明
00XXX	正常に実行された
01XXX	警告
02XXX	データが存在しない。影響を受けたローがない。
その他の値	エラー

SQLSTATE コードおよびエラー・メッセージの取得

SQLSTATE は、1 つ以上のエラー・メッセージ、警告メッセージ、またはその両方のリストが含まれます。メッセージには、情報、警告、重大なエラー、致命的エラーのメッセージがあります。Open Client Client-Library および Open Server Server Library は、SQLSTATE メッセージを生成します。SQLSTATE コードおよびエラー・メッセージの全リストについては、適切なマニュアルを参照してください。

プリコンパイラが生成する SQLSTATE メッセージの表については、「付録 A プリコンパイラの警告とエラー・メッセージ」を参照してください。

第 4 章

変数の使い方

この章では、アプリケーションと Adaptive Server Enterprise 間でデータを渡す、次の 2 種類の変数について説明します。

- ホスト変数：Embedded SQL 文で使用する COBOL 言語の変数で、Adaptive Server Enterprise から取得したデータや Adaptive Server Enterprise に送信するデータを格納します。
- インジケータ変数:ホスト変数に関連する null データやデータのトランケートを示す変数です。

変数の宣言

「第 3 章 Adaptive Server Enterprise との通信」で説明したように、アプリケーション・プログラムに SQLCA、SQLCODE、SQLSTATE が含まれていると、プリコンパイラが自動的にシステム変数を設定します。ただし、Embedded SQL 文を使用する前に、declare セクションでホスト変数とインジケータ変数を明示的に宣言してください。

警告！ プリコンパイラによって作成される一部の変数は、すべて “SQL--” で始まっています。自分で作成する変数には、先頭に “SQL” を付けないでください。エラー・メッセージや不正確なデータを受け取ることがあります。

---

declare セクションで COPY 文は使用できません。declare セクションの構文は、次のとおりです。

```
exec sql begin declare section end-exec
    declarations ...
exec sql end declare section end-exec.
```

ホスト変数の宣言は、COBOL 言語のデータ宣言の規則に従ってください。プログラム内の declare セクションの数は無制限なので、すべての変数を同じ declare セクションで宣言する必要はありません。

注意 11.1 以降のバージョンでは、PIC 句への更新をサポートしません。

変数を宣言するには、ピクチャおよび usage 句も指定してください。有効なピクチャおよび usage 句については、「COBOL データ型と Adaptive Server Enterprise データ型の比較」(32 ページ) の項を参照してください。

次に、declare セクションの例を示します。

```
exec sql begin declare section end-exec
 01  E-NAME      PIC X(30).
 01  E-TYPE      PIC X(3).
 01  TINY-INT    PIC S9(2) COMP.
 01  SHORT-INT   PIC S9(4) COMP.
 01  MONEY-DATA  CS-MONEY.
 exec sql end declare section end-exec.
```

文字配列の宣言

プリコンパイラは、「複雑な定義」、つまり構造体と配列をサポートしています。構造体をネストできますが、構造体の配列は作成できません。

プリコンパイラは、あらゆるデータ型の 1 次元配列を認識します。また、次の例のように、プリコンパイラは文字の 2 次元配列も認識します。

```
01 NUMSALES PIC S9(9) OCCURS 25 TIMES.
exec sql begin declare section end-exec.
 01 DAYS-OF-THE-WEEK PIC X(31) OCCURS 7 TIMES.
 exec sql end declare section end-exec.
```

配列の詳細については、「配列の使い方」(28 ページ) を参照してください。

ホスト変数の使い方

ホスト変数を使用すると、Adaptive Server Enterprise とアプリケーション・プログラム間で値を転送できます。

ホスト変数は、アプリケーション・プログラムの Embedded SQL の declare セクションで宣言します。宣言しないと SQL 文で変数を使用できません。

Embedded SQL 文の中でホスト変数を使うときは、変数の前にコロンを付けます。プログラム内のその他の箇所でこの変数を使うときは、コロンは使わないでください。Embedded SQL 文の中で複数のホスト変数を続けて使用するときは、それぞれの変数の間をカンマで区切るか、SQL 文の文法規則に従ってください。

次の例は、ホスト変数の使い方を示しています。*PAR-1*、*PAR-2*、および *PAR-3* をホスト変数として宣言し、myproc プロシージャに対するパラメータとして使用しています。

```
exec sql begin declare section end-exec
 01  PAR-1              PIC X(10).
 01  PAR-2              PIC X(10).
 01  PAR-3              PIC X(10).
 exec sql end declare section end-exec
 exec sql exec myproc :PAR-1, :PAR-2, :PAR-3 end-exec.
```

ホスト変数には、次の 4 とおりの使い方があります。

- SQL 文およびプロシージャの入力変数
- 結果変数
- SQL プロシージャの呼び出しによるステータス変数
- SQL 文およびプロシージャの出力変数

関数に関係なく、「変数の宣言」(21 ページ) に記載されているあらゆるホスト変数を宣言してください。次に、ホスト変数の使い方を説明します。

ホスト入力変数

これらの変数は、情報を Adaptive Server Enterprise に渡します。アプリケーション・プログラムは、これらの変数に値を代入します。ここに格納されたデータは、ストアド・プロシージャ、where 句のある select 文、values 句のある insert 文、set 句のある update 文などの実行文に使用されます。

次の例では、*TITLE-ID1*、*TITLE-ID2*、*PUB-ID* 変数を入力変数として使用しています。

```
exec sql begin declare section end-exec
 01  TITLE-ID1       PIC X(6).
 01  TITLE-ID2       PIC X(6).
 01  PUB-ID          PIC X(4).
 exec sql end declare section end-exec

 exec sql delete from titles
      where title_id = :TITLE-ID1 end-exec.
 exec sql update titles set pub_id = :PUB-ID
      where title_id = :TITLE-ID2 end-exec.
```

ホスト結果変数

これらの変数は、select 文と fetch 文の結果を受け取ります。

次の例では、*TITLE-ID* 変数を結果変数として使用しています。

```
exec sql begin declare section end-exec
 01  TITLE-ID           PIC X(6).
 exec sql end declare section end-exec

 exec sql select title_id into :TITLE-ID from titles
      where pub_id = "0736"
      and type = "business" end-exec.
```

ホスト・ステータス変数

これらの変数は、ストアド・プロシージャのリターン・ステータス値を受け取ります。ステータス変数は、ストアド・プロシージャが正常に終了したかどうか、また失敗した場合はその理由を示します。Adaptive Server Enterprise のデータ型から smallint に変換できる変数を使用してください。

次の例では、*RET-CODE* 変数を「ステータス変数」として使用しています。

```
exec sql begin declare section end-exec
 01  RET-CODE   PIC S9(4) COMP.
 exec sql end declare section end-exec.
        . . .
 exec sql exec :RET-CODE = update_pubs end-exec.
 IF RET-CODE NOT = 0
 exec sql rollback transaction end-exec.
```

ホスト出力変数

ホスト出力変数は、ストアド・プロシージャから受け取ったデータをアプリケーション・プログラムへ渡します。ストアド・プロシージャが out を使用して宣言したパラメータ値を返すときには、ホスト出力変数を使用します。「ストアド・プロシージャの使用」(57 ページ) を参照してください。

次の例では、変数 *PAR1* と *PAR2* を出力変数として使用しています。

```
exec sql exec a_proc :PAR1 out, :PAR2 out end-exec.
```

インジケータ変数の使い方

インジケータ変数をホスト変数と対応付けると、データベースの値が null の場合を示すことができます。スペースと、オプションの indicator キーワードを使って、各「インジケータ変数」を対応するホスト変数と区切ります。インジケータ変数は、それぞれ対応するホスト変数の直後に置いてください。

インジケータ変数がないと、Embedded SQL は NULL 値を示すことができません。

インジケータ変数およびサーバの制約

Embedded SQL は一般的なインタフェースであり、Adaptive Server Enterprise の他にも、さまざまなサーバで実行できます。

このため、Embedded SQL は特定のサーバの制約を適用または反映することはありません。たとえば、Embedded SQL で text および image ストアド・プロシージャ・パラメータは使用できますが、Adaptive Server Enterprise では使用できません。

Embedded SQL のアプリケーションを記述する場合、アプリケーションの対象となるサーバの制約に注意してください。あるサーバについてどのような制約があるか不明なときは、そのサーバのマニュアルを参照してください。

インジケータ変数を使用したホスト変数

Embedded SQL 文のあるアプリケーション・プログラムでインジケータ変数を使用する前に、declare セクションでホスト変数とインジケータ変数を宣言します。

declare セクションでインジケータ変数を次のうちの 1 つとして宣言してください。

PIC S9(4) COMP
DISPLAY SIGN LEADING (オプションで、SEPARATE)
DISPLAY SIGN TRAILING (オプションで、SEPARATE)
COMP-3
COMP-4
COMP-5
BINARY

Embedded SQL 文でこの変数を使うときは、インジケータ変数の前にコロンを付けてください。インジケータ変数をホスト変数と対応付ける構文は、次のとおりです。

```
:host_variable [[indicator] :indicator_variable]
```

インジケータ変数とホスト変数の対応関係は、1 つの exec sql 文でのみ有効です。

Adaptive Server Enterprise がインジケータ変数を設定するのは、ホスト変数に値を代入したときだけです。したがって、インジケータ変数を宣言したあとで、別の文で別のホスト変数とともに再使用できます。

インジケータ変数は、出力変数、結果変数、および入力変数とともに使用できます。出力変数や結果変数とともに使用する場合、Embedded SQL は、この変数を対応するホスト変数の NULL ステータスを示すように設定します。入力変数とともに使用する場合は、インジケータ変数の値を Adaptive Server Enterprise に送出される前の「入力変数」の null ステータスを示すように設定してください。

注意 インジケータ変数は、出力変数、結果変数、および入力変数とともに使用できます。

ホスト出力変数と結果変数に対応するインジケータ変数の使い方

インジケータ変数を出力変数または「結果変数」に対応付けると、Client-Library は、自動的にこの変数を表 4-1 に示すいずれかの値に設定します。

表 *4-1:* 出力変数または結果変数と併用したインジケータ変数の値

値	意味
-1	対応する Adaptive Server Enterprise のデータベース・カラムに null 値がある。
0	ホスト変数に null 以外の値が代入された。
>0	ホスト変数のデータ変換中にオーバフローが発生した。ホスト変数には、トランケートされたデータが代入される。正の数は、トランケートされる前の値の長さをバイト単位で表している。

次の例は、インジケータ変数 *INDIC-V* と結果変数 *PUB-NAME* の対応関係を示しています。

```
exec sql begin declare section end-exec
  01  INDIC-V     PIC S9(4) COMP.
  01  PUB-ID      PIC X(4).
  01  PUB-NAME    PIC X(20).
 exec sql end declare section end-exec

    exec sql select pub_name into :PUB-NAME :INDIC-V
         from publishers where pub_id = :PUB-ID
             end-exec.

     if INDIC-V = -1
         display "No Publisher name"
     else
         display "Publisher Name is: " PUB-NAME.
```

インジケータ変数を使用したホスト入力変数

インジケータ変数を入力変数に対応付けるときは、表 4-2 の値を参考にインジケータ変数を明示的に設定してください。

表 *4-2:* 入力変数と併用したインジケータ変数の値

値	意味
-1	対応する入力を NULL 値として扱う。
0	ホスト変数の値をカラムに代入する。

入力値が null かどうかを調べてインジケータ変数に -1 を代入するには、ホスト言語コードを指定してください。これによって、null 値についての情報が Client-Library に通知されます。インジケータ変数を -1 に設定すると、ホスト変数の実際の値に関係なく null が使用されます。

次の例は、インジケータ変数と入力変数の対応関係を示しています。*R-INDIC* が -1 に設定されているため、データベースの royalty カラムは null 値に設定されます。*R-INDIC* の値を変更すると、royalty の値も変化します。

```
exec sql begin declare section end-exec
 01   R-INDIC   PIC S9(4) COMP.
 01   R-VAR     PIC X(10).
 exec sql end declare section end-exec.

     MOVE -1 TO R-INDIC.
     exec sql update titles
             set royalty = :R-VAR :R-INDIC
         where pub_id="0736" end-exec.
```

ホスト変数の規約

ホスト変数名は、COBOL の命名規約に準拠していなければなりません。

同じロケーションで Transact-SQL 文の Transact-SQL のリテラルを使用できるならば、Embedded SQL 文でホスト変数を使用できます。

ホスト変数は、プリコンパイラの有効なデータ型にしてください。ホスト変数のデータ型は、返されるデータベース・カラム値のデータ型と互換性があるものにします。詳細については、表 4-3 (34 ページ) と表 4-4 (35 ページ) を参照してください。

ホスト言語の予約語と Embedded SQL のキーワードは、変数名として使えません。

ホスト変数は、「動的 SQL」で指定される場合を除いて、Embedded SQL のキーワードやデータベース・オブジェクトを表すことはできません。「第 4 章 変数の使い方」 を参照してください。

ホスト変数が SQL 文の文字列を表す場合は、引用符で囲まないでください。

次に示す例は、プリコンパイラが必要に応じて値の前後に引用符を挿入するので、正しくありません。引用符は入力しないでください。

```
exec sql select pub_id from publishers
     where pub_id like ":PUB-ID"

end-exec
```

次に正しい例を示します。

```
exec sql select pub_id from publishers
     where pub_id like :PUB-ID
end-exec
```

配列の使い方

配列は、関連するデータの集まりを 1 つの変数にしたものです。配列は、select 文や fetch 文の into 句の出力変数として使用できます。次に例を示します。

```
01 author-array.
 10 author-name    PIC X(30)   occurs 100 times.
exec sql
 select au_lname
 from authors
 into :au_array
end-exec.
```

注意 単一の項目は、配列のどこにでもフェッチできます。しかし、複数ローは、配列の先頭にしかフェッチできません。

---

select と fetch into の使い方については、第 6 章の「配列を使用した複数ローの選択」(45 ページ) を参照してください。

複数の配列

1 つの SQL 文で複数の配列を使用する場合、配列はすべてサイズが同じでなければなりません。サイズが異なっていると、エラー・メッセージが表示されます。

スコープの規則

プリコンパイラは、ネストされた COBOL プログラムおよび COBOL における変数のスコープ規則をサポートしています。ホスト変数は is global 句および is external 句を使用できます。次にネストされる例を示します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  outer.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  xyz.
 OBJECT-COMPUTER.  xyz.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
    exec sql begin declare section end-exec.
     01 global-var is global pic x(10).
     01 not-global-var pic x(10).
     01 shared-var is external pic x(10).
     exec sql end declare section end-exec.
procedure division.
 p0.
      . . .
IDENTIFICATION DIVISION.
 PROGRAM-ID.  inner.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  xyz.
 OBJECT-COMPUTER.  xyz.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
procedure division.
p0.
       . . .
* This is legal because global-var was
 * declared using is global
    exec sql
         select au_lname into :global-var
             where au_id = "998-72-3567"
    end-exec.
* This is not legal because not-global-var was
  * not declared using is global
    exec sql
         select au_lname into :not-global-var
             where au_id = "998-72-3567"
     end-exec.
       . . .
```

```
end program inner.
end program outer.
IDENTIFICATION DIVISION.
 PROGRAM-ID.  nonest.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  xyz.
 OBJECT-COMPUTER.  xyz.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
    exec sql begin declare section end-exec.
         01 local-var pic x(10).
         01 shared-var is external pic x(10).
     exec sql end declare section end-exec.
procedure division.
p0.
       . . .
* This is legal.
     exec sql
         select au_lname into :local-var
         where au_id = "998-72-3567"
     end-exec.
* So is this.
    exec sql
         select au_lname into :shared-var
         where au_id = "998-72-3567"
     end-exec.
    . . .
end program nonest.
```

データ型

COBOL の「ベニア層」とは、プリコンパイルされたアプリケーションによって、Open Client の Client-Library とともに使用されるライブラリのことをいいます。プリコンパイラによって生成される COBOL コードは、このベニア層にある関数を呼び出します。各関数は、Client-Library の特定の関数を呼び出します。ベニア層は、COBOL と Client-Library との通信を可能にする変換などの操作を実行します。また、ベニア層は、COBOL ホスト変数と Adaptive Server Enterprise データ型の間の変換も行います。

ESQL/COBOL ベニア層には、静的と動的共有の 2 種類があります。次の表に、32 ビットと 64 ビットのすべてのプラットフォームでリリースされている動的共有ベニア層ライブラリを示します。

プラットフォーム	ライブラリ名	リエントラント・バージョン
HP-UX PA-RISC 32 ビット版	*libsybcobct.sl*	*libsybcobct_r.sl*
HP-UX PA-RISC 64 ビット版	*libsybcobct64.sl*	*libsybcobct_r64.sl*
ESQL/COBOL をサポートしているその他の 32 ビット・プラットフォーム	*libsybcobct.so*	*libsybcobct_r.so*
ESQL/COBOL をサポートしているその他の 64 ビット・プラットフォーム	*libsybcobct64.so*	*libsybcobct_r64.so*

既存の静的なバージョンの ESQL/COBOL ベニア層ライブラリは、*libsybcobct.a* です。

データ項目には、基本データ項目とグループ・データ項目の 2 つの種類があります。以降の項では、これらの各データ項目について説明します。

基本データ項目

「基本データ項目」とは、個別のパーツに分割できない完全な項目のことをいいます。基本データ項目は、ホスト変数として使用できます。

次に、基本データ項目の例を示します。

```
01   MYSTR  PIC X(26).
```

MYSTR は、基本データ項目であるため、ホスト変数 (*:MYSTR*) として使用できます。

グループ・データ項目

複数の基本データ項目が結合して関連項目のグループを形成する場合、これらの項目を「グループ・データ項目」と呼びます。グループ・データ項目は、ホスト変数として使用できます。declare セクションでグループ・データ項目を宣言してください。

次にグループ項目の例を示します。

```
01 AUTH-REC.
10 AUTH-NAME                    PIC X(25).
10 STATE                        PIC X(25).
10 TOTAL-SALES PIC S9(9) COMP SYNC
```

次に、データ項目がホスト変数であるグループ項目へ、項目を選択して入れる例を示します。

```
exec sql select au_lname, salary, tot_sales
 from table into :AUTH-REC end-exec
```

上の例の結果は、次のコードと同じです。

```
exec sql select au_lname, salary, tot_sales
 from table into :AUTH-NAME, :SALARY, :TOTAL-SALES
```

次の例でも同じ結果になります。

```
exec sql select au_lname, salary, tot_sales
 from table into :AUTH-NAME OF AUTH-REC,
 :SALARY OF AUTH-REC, :TOTAL-SALES OF AUTH-REC
```

Embedded SQL/COBOL は、exec sql 文のホスト変数の C 言語構造構文もサポートしています。たとえば、前述の例は、次のように書き直すことができます。

```
exec sql select au_lname, salary, tot_sales
 from table into :AUTH-REC.AUTH-NAME,
 :AUTH-REC.SALARY, :AUTH-REC.TOTAL-SALES
```

SYNC は、グループ・データ項目内で宣言された COMP、COMP-4、COMP-5、BINARY データ項目とともに使用します。

特別なデータ項目

CS_MONEY、CS-TEXT、CS-IMAGE などの特別な Sybase のデータ型は、次に示すように宣言します。

```
01 MYTEXT PIC x(100) USAGE IS CS-TEXT.
```

COBOL データ型と Adaptive Server Enterprise データ型の比較

ホスト変数データ型は対応するデータベース・カラムのデータ型と互換性を持たせてください。したがって、アプリケーション・プログラムを記述する前に、データベース・カラムのデータ型を確認してください。

データ型には次の規則が適用されます。

- “To: COBOL Datatype” カラムにあるホスト変数を入力または出力として使用する場合、適切な変換が自動的に行われます。
- インジケータ変数は、COMP、COMP-3、COMP-4、COMP-5、BINARY、または DISPLAY の変形を使用してください。この場合、S9(4) または同等のピクチャ文字列が必要です。

- PIC S9(1-9) COMP ではどのような値でも使用できます。10 進数トランケーションが 起きる場合、トランケーション・メッセージは返されません。その代わりに、特に数字のトランケーションを示す、SQLCA または SQLSTATE エラー・メッセージが返されます。

  たとえば、PIC S9(4) に “1234” の値を選択した場合、値が指定されたバイトに合っているので、トランケーション・メッセージは返されません。これに対して、PIC S9(3) に “1234567” の値を選択した場合、値が指定されたバイトに合っていないので、トランケーション・メッセージが返されます。

データ型の変換

プリコンパイラは、自動的にホスト変数のデータ型を Adaptive Server Enterprise のテーブル・カラムのデータ型と比較します。Adaptive Server Enterprise のデータ型とホスト言語データ型に互換性はあるが同一ではない場合、COBOL ベニア層が別の型に変換します。プリコンパイラがデータを別のデータ型に変換できる場合、データ型には互換性があります。データ型に互換性がない場合は、実行時に変換エラーが発生し、SQLCODE または SQLSTATE が負の数に設定されます。

長いデータ型を短いデータ型に変換する場合 (長い整数を PIC S9(4) COMP に変換する場合など) は、データをトランケートする可能性が常にあるため、注意してください。トランケーションが起きた場合、SQLWARN1 が設定されます。

注意　Adaptive Server Enterprise のデータを、カンマや 10 進数の文字などの編集中の文字がある COBOL の数値フィールドにフェッチしないでください。その代わりに comp や display sign leading separate などの編集されていないフィールドにデータをフェッチして、編集されたフィールドにデータを移動してください。

結果変数のデータ型の変換

表 4-3 は、結果変数に対するデータ変換が有効かどうかを示します。黒点は変換できることを示していますが、「ホスト変数」のデータ型を選択するときに注意しないと、エラーが発生します。

表 *4-3:* 結果変数のデータ型の変換

	変換先：COBOL データ型				
変換元：Adaptive Server Enterprise データ型	**S9(1—9) COMP、COMP-4、COMP-5、BINARY**	**CS-DATE、CS-TIME、CS-DATETIME、CS-DATETIME4**	**PIC X(n)**	**S9(m)V9(n) DSLS、DSL、DSTS、DST、COMP、COMP-3**	**CS-MONEY、CS-MONEY4**
char	•	•	•	•	•
varchar	•	•	•	•	•
bit	•		•	•	
tinyint	•		•	•	•
smallint	•		•	•	•
int	•		•	•	•
bigint	•		•	•	•
ubigint	•		•	•	•
uint	•		•	•	•
usmallint	•		•	•	•
float	•		•	•	•
money	•		•	•	•
money4	•		•	•	•
numeric	•		•	•	•
real	•		•	•	•
date		•			
time		•			
datetime		•	•		
datetime4		•	•		

キー：DSL = Display Sign Leading
DSLS = Display Sign Leading Separate
DST = Display Sign Trailing
DSTS = Display Sign Trailing Separate

入力変数のデータ型の変換

表 4-4 は、入力変数に対するデータ変換が有効かどうかを示します。黒点は、変換できることを示しています。変換できないホスト変数のデータ型を選択すると、トランケートなどのエラーが発生します。

表 *4-4:* 入力変数のデータ型の変換

	変換先：Adaptive Server Enterprise データ型												
変換元：COBOL データ型	**varchar**	**money**	**date、time**	**int、smallint**	**bigint**	**ubigint**	**uint**	**usmallint**	**bit**	**float**	**char**	**numeric**	**real、float**
S9(1—9) COMP、COMP-4、COMP-5、BINARY		•		•	•	•	•	•		•		•	•
CS-DATE、CS-TIME、CS-DATETIME、CS-DATETIME4			•										
PIC X(n)	•		•								•		
S9(m)V9(n) DSLS、DSL、DSTS、DST、COMP-3		•		•	•	•	•	•	•	•		•	•
CS-MONEY、CS-MONEY4	•	•		•	•	•	•	•	•	•	•	•	•

キー：DSL = Display Sign Leading
DSLS = Display Sign Leading Separate
DST = Display Sign Trailing
DSTS = Display Sign Trailing Separate

第 5 章

Adaptive Server Enterprise との接続

この章では、Embedded SQL プログラムを Adaptive Server Enterprise と接続する方法、およびサーバ、ユーザ名、パスワードの指定方法について説明します。この章では、次の項目について説明します。

サーバへの接続

アプリケーション・プログラムと Adaptive Server Enterprise を接続するには、connect 文を使用します。アプリケーションが C と COBOL の両方の言語を使用している場合、最初の connect 文は COBOL プログラムから発行されなければなりません。

connect 文の構文は次のとおりです。

```
exec sql connect :user [identified by :password]
 [at :connection_name] [using :server]
 [label_name label_name label_value label_value...]
 end-exec
```

以降の各項では、connect 文の各引数について、1 つずつ説明します。connect 文に必須なのは *user* 引数のみです。それ以外の引数はオプションです。

user

user はホスト変数または引用符付き文字列で、Adaptive Server Enterprise ユーザ名を示します。ユーザ名は指定されているサーバで有効である必要があります。

password

password はホスト変数または引用符付き文字列で、指定されたユーザ名に対応するパスワードを示します。この引数は、Adaptive Server Enterprise へアクセスするときにパスワードが要求される場合にのみ必要です。password 引数が null 値に設定されている場合は、ユーザがパスワードを指定する必要はありません。

connection_name

connection_name は、Adaptive Server Enterprise 接続をユニークに識別します。二重引用符付きリテラルおよび引用符のないリテラルを使用することもできます。アプリケーション・プログラム内では接続をいくつでも作成でき、そのうちの 1 つは名前を付けなくてもかまいません。*connection_name* の最大サイズは、255 文字です。

connect 文中で *connection_name* を使用すると、それ以降同じ接続を指定した Embedded SQL 文は、connect 文で指定したサーバを自動的に使用します。connect 文でサーバを指定していない場合には、「デフォルト・サーバ」が使用されます。デフォルト・サーバの決定方法については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

注意 現在のサーバ接続を変更するには、「現在の接続の変更」(39 ページ) で説明されている set connection 文を使用してください。

Embedded SQL 文は、connect 文で指定された *connection_name* だけを参照します。アプリケーション・プログラムが使用するサーバ 1 つにつき、少なくとも 1 つの connect 文が必要です。

server

server はホスト変数または引用符付き文字列で、サーバ名を示します。*server* には、サーバをユニークにかつ完全に識別する文字列を使用してください。

connect の使用例

次の例は、UNIX のフォーマットを使用して SYBASE というサーバに接続します。

```
exec sql begin declare section end-exec
01      USER       PIC X(16)   VALUE "myname"
01      PASSWD     PIC X(16)   VALUE "abcdefg".
01      SERV-NAME  PIC X(16).
01      MY-SERVER  PIC X(512).
exec sql end declare section end-exec.
    MOVE "SYBASE" TO SERV-NAME.
exec sql connect :USER identified by :PASSWD
     using :SERV-NAME end-exec.
```

現在の接続の変更

現在の接続を変更するには、set connection 文を使用します。文の構文は、次のとおりです。

```
exec sql set connection {connection_name | default}
```

“default” は、名前の付いていない接続があればそれを指します。

次は、現在の接続を変更する例を示します。

```
exec sql connect "ME" at connect1 using "SERVER1" end-exec
exec sql connect "ME" at connect2 using "SERVER2" end-exec
exec sql set connection connect1 end-exec
exec-sql select user_id()into :MYID end-exec
```

複数の接続の確立

効率を考えると、Embedded SQL アプリケーションには、複数のアクティブな Adaptive Server Enterprise 接続が必要な場合があります。次に例を示します。

- 複数の Adaptive Server Enterprise のログイン名を必要とするアプリケーションでは、ログイン・アカウント名ごとに 1 つずつ接続を持つことができます。
- 複数のサーバに接続することによって、アプリケーションは複数のサーバ上にあるデータへ同時にアクセスできます。

単一のアプリケーションが、1 つのサーバに対して複数の接続を持つことも、複数のサーバに対して複数の接続を持つことも可能です。アプリケーションに複数の接続を指定するには、connect 文の at *connection_name* 句を使用してください。

1 つの接続をオープンして、次にもう 1 つ別の新しい接続 (名前のある接続または名前のない接続) をオープンする場合は、新しい接続が現在の接続になります。

注意 プリコンパイラを使用して適切な SQL 文のストアド・プロシージャを作成する場合、プリコンパイラは、すべてのサーバ上のすべてのストアド・プロシージャ用のファイルを、各 Embedded SQL ファイルに 1 つずつ作成します。このファイルは、ユーザが適切なサーバにロードできます。サーバは他のサーバ用のプロシージャが読み込めないという警告やエラーを出しますが、これは無視してください。各サーバにとって適切なストアド・プロシージャが、そのサーバ上に正しくロードされます。使用するすべてのサーバにストアド・プロシージャをロードしてください。そうしないと、クエリは失敗します。

接続名の指定

表 5-1 は、接続名の付け方を示します。

表 *5-1:* 接続名の付け方

使用されている句	使用されていない句	接続名
at *connection_name*		*connection_name*
using *server_name*	at	*server_name*
なし		“DEFAULT” 接続の実際の名前

at 句では無効な文

次の文は、at 句では無効になります。

- connect
- begin declare section
- end declare section
- include file
- include sqlca
- set connection
- whenever

Adaptive Server Enterprise 接続の使い方

名前の付いていないデフォルトの接続を使用するとき以外は、Embedded SQL 文を実行するには、常に接続名を指定します。アプリケーション・プログラムが接続を 1 つしか使用しないときは、接続に名前を付けなくてもかまいません。また、at 句を省略できます。

複数の接続を使用する場合の構文は、次のとおりです。

```
exec sql [at connection_name] sql_statement
     end-exec
```

sql_statement は Transact-SQL 文です。

次の例は、2 つの接続を異なるサーバに対して確立し、使用する方法を示します。

```
exec sql begin declare section end-exec
 01  USER            PIC X(16) VALUE "myname".
 01  PASSWD          PIC X(16) VALUE "mypass".
 01  AU-NAME         PIC X(20).
 01  A-VALUE         PIC S9(9) COMP.
 01  A-TEST          PIC S9(9) COMP.
 01  SERVER-1        PIC X(16).
 01  SERVER-2        PIC X(16).
 exec sql end declare section end-exec.
      . . .
      MOVE "sybase1" TO SERVER-1.
      MOVE "sybase2" TO SERVER-2.

      exec sql connect :USER identified by :PASSWD
         using :SERVER-1 end-exec.
      exec sql connect :USER identified by :PASSWD
         at connection-2 using :SERVER-2 end-exec.

 * This statement uses the current connection
 * (connection-2)
     exec sql select royalty into :A-VALUE from pubs
           where author = :AU-NAME end-exec.

 * This statement uses connection "SERVER-1"
     IF A-VALUE = A-TEST
     exec sql at SERVER-1 update titles
          set column = :A-VALUE * 2
          where author = :AU-NAME end-exec.
```

サーバとの接続の切断

アプリケーション・プログラムが確立した接続は、明示的にクローズするか、プログラムが終了するまでオープンされたままになります。disconnect 文を使用して、アプリケーション・プログラムと Adaptive Server Enterprise との接続をクローズします。

disconnect 文の構文は、次のとおりです。

```
exec sql disconnect {connection_name | current |
     default | all} end-exec
```

パラメータの意味は次のとおりです。

- current は、現在の接続を指定します。
- default は、名前の付いていないデフォルト接続を指定します。
- all は、現在オープンしているすべての接続を指定します。

disconnect 文は、次の処理を行います。

1 トランザクションをロールバックします。このとき、セーブポイントが設定されていても無視します。

2 接続をクローズします。

3 テーブルなどのテンポラリ・オブジェクトをすべて削除します。

4 オープンしているカーソルをすべてクローズします。

5 現在のトランザクション用に設定されたロックを解放します。

6 サーバのデータベースへのアクセスを終了します。

disconnect は、現在のトランザクションを暗黙的には commit しません。

警告！ プログラムを終了する前に、オープンしている各接続に対して exec sql disconnect または exec sql disconnect all 文を実行してください。設定の仕様によっては、切断されずにクライアントが終了することが、Adaptive Server Enterprise には通知されないことがあります。この場合は、アプリケーションによって保持されていたリソースが解放されません。

第 6 章

Transact-SQL 文の使い方

この章では、Embedded SQL とホスト変数による Transact-SQL 文の使い方について説明します。また、「ストアド・プロシージャ」の使い方についても説明します。ストアド・プロシージャとは、Adaptive Server Enterprise に保管されている SQL 文の集まりです。ストアド・プロシージャは、コンパイルしてからデータベースに保存されるので、起動時に再コンパイルすることなく高速に実行されます。

Embedded SQL における Transact-SQL 文

exec sql 構文

Embedded SQL 文は、exec sql というキーワードで始めて、end-exec というキーワードで終了してください。Embedded SQL 文の構文は、次に示すとおりです。

```
exec sql [at connection_name] sql_statement end-exec
```

パラメータの意味は次のとおりです。

- *connection_name* は、文に対する接続を指定します。接続の説明については、「第 5 章 Adaptive Server Enterprise との接続」を参照してください。at というキーワードは Transact-SQL 文および disconnect 文について有効です。
- *sql_statement* は、1 つ以上の Transact-SQL 文です。

無効な文

次に挙げる Transact-SQL 文を除いて、Embedded SQL 中では、すべての Transact-SQL 文が有効です。

- print
- raiserror
- readtext
- writetext

Embedded SQL における Transact-SQL 文の相違点

ほとんどの Transact-SQL 文は、Embedded SQL で使用される場合にその機能性や構文を保持しますが、select 文、update 文、delete 文などのデータ操作言語文 (DML 文) は、Embedded SQL においては多少異なります。

- 次の 4 項目は、select 文の into 句に特有のものです。
  - into 句は、スカラ・ホスト変数に単一のデータ・ローを割り当てることができます。この句は、単一のデータ・ローを返す select 文に対してのみ有効です。select を使用して複数のローを選択した場合、負の SQLCODE となり、最初のローだけが返されます。
  - into 句内の変数が配列の場合は、select を使用して複数のローを選択できます。配列が保持できる数よりも多くのローを select した場合には、SQLCODE <0 の例外が発生し、余分なローは失われます。
  - select は、カーソルや配列への選択を使わなければ、ホスト変数に複数のデータ・ローを返すことはできません。
  - update 文と delete 文では、探索条件 where current of *cursor_name* を使用できます。

ローの選択

select 文で使用できる最大カラム数は 1024 です。select 文の構文の完全なリストについては、『ASE リファレンス・マニュアル』を参照してください。

1 つのローの選択

カーソルや配列を使用しない場合、select 文はデータ中の単一のローだけを返すことができます。Embedded SQL で複数のデータ・ローを返すには、カーソルまたは配列を使う必要があります。

Embedded SQL では、select 文には into 句が必要です。into 句は値を割り当てるホスト変数のリストを指定します。

注意　Embedded SQL プリコンパイラの現在のバージョンでは、テーブルを指定する into 句をサポートしていません。

Embedded SQL の select 文の構文は、次のとおりです。

```
exec sql [at connect_name]
     select [all | distinct] select_list into
     :host_variable[[indicator]:indicator_variable]
     [, :host_variable
     [[indicator]:indicator_variable]...]
 end-exec
```

select 文の句の詳細については、『ASE リファレンス・マニュアル』を参照してください。

pubs2 データベース中の authors テーブルにアクセスし、*au_id* の値をホスト変数 *ID* に割り当てる select 文を発行する構文の例は、次のようになります。

```
exec sql select au_id into :ID from authors
     where au_lname = "Stringer"
 end-exec
```

配列を使用した複数ローの選択

配列を使用すると、複数のローを返すことができます。配列に対しては、配列への選択とフェッチという 2 つの動作があります。

配列への *select into* の使用

返されるローの最大数がわかっている場合は、「select into 配列」という方法を使用します。select into 文が配列の保持できるローよりも多くのローを返した場合、その文は、最小の配列が保持できるローの最大数を返します。

次に、配列を選択する例を示します。

```
exec sql begin declare section end-exec
 01 TITLEID-ARRAY PIC X(6) OCCURS 100 TIMES.
 exec sql end declare section end-exec
 ...
 exec sql select title_id into :titleid-array
     from titles end-exec.
```

配列へのインジケータの使用

配列フェッチとともにインジケータを使用するには、*host_variable* 配列と同じ長さを持つインジケータの配列を宣言し、そのインジケータをホスト変数に関連付けるための構文を使用してください。

例

```
exec sql begin declare section end-exec
 01 ITEM-NUMBERS S9(9) OCCURS 100 TIMES.
 01 I-ITEM-NUMBERS S9(4) OCCURS 100 TIMES.
 exec sql end declare section end-exec
 ...
 exec sql select it_n from item.info
     into :item-numbers :i-item-numbers end-exec.
 ...
```

インジケータ変数としての配列と構造体

多数のカラムを持つテーブルの場合は、SQL 文で参照されるホスト変数のセットとして、配列と構造体を使用できます。この機能が正しく動作するには、インジケータ配列要素またはインジケータ構造体要素を PIC S9(4) 句と COMP-5 句で宣言する必要があります。ESQL/C と同様に ESQL/COBOL でも、構造体と配列をインジケータ変数として使用すると、アプリケーションにおける各 Embedded SQL 文の null 入力可能なカラムごとに単一インジケータ変数をコーディングする時間のかかるプロセスを省くことができます。

例

例 1 インジケータ配列を宣言し、そのインジケータ配列にクエリを実行する例を示します。

```
* Declare variables
....
01 HOST-STRUCTURE-M1.
   03 M-TITLE PIC X(64).
   03 M-NOTES PIC X(200).
   03 M-PUBNAME PIC X(40).
   03 M-PUBCITY PIC X(20).
   03 M-PUBSTATE PIC X(2).

01 INDICATOR-TABLE.
   03 I-NOTES-ARR PIC S9(4) COMP-5 OCCURS 5 TIMES.
....


* Execute query
....
EXEC SQL
SELECT substring(title, 1, 64), notes, pub_name,
            city, state
```

```
     INTO :HOST-STRUCTURE-M1:I-NOTES-ARR
     FROM titles, publishers
     WHERE titles.pub_id = publishers.pub_id
     AND title_id = :USER-TITLEID
END-EXEC.
....
```

例 2 インジケータ構造体を宣言し、そのインジケータ構造体にクエリを実行する例を示します。

```
* Declare variables
....
01 HOST-STRUCTURE-M1.
   03 M-TITLE PIC X(64).
   03 M-NOTES PIC X(200).
   03 M-PUBNAME PIC X(40).
   03 M-PUBCITY PIC X(20).
   03 M-PUBSTATE PIC X(2).

01 INDICATOR-STRUCTURE-I1.
   03 I-TITLE PIC S9(4) COMP-5.
   03 I-NOTES PIC S9(4) COMP-5.
   03 I-PUBNAME PIC S9(4) COMP-5.
   03 I-PUBCITY PIC S9(4) COMP-5.
   03 I-PUBSTATE PIC S9(4) COMP-5.
....

* Execute query
....
EXEC SQL
SELECT substring(title, 1, 64), notes, pub_name, city,
       state
    INTO :HOST-STRUCTURE-M1:INDICATOR-STRUCTURE-I1
    FROM titles, publishers
    WHERE titles.pub_id = publishers.pub_id
    AND title_id = :USER-TITLEID
END-EXEC.
```

使用法

構造体と配列をインジケータ変数として使用する場合は、次の点に留意してください。

- インジケータ配列またはインジケータ構造体の要素数は、ホスト変数構造体の要素数と一致する必要があります。両者が一致しないと、cobpre または cobpre64 が発生し、処理が停止しコードは生成されません。
- SELECT リストのカラムが、INTO リストの順序、データ型、選択した構造体名と一致している必要があります。一致しないと、ct_bind() 実行時エラーが発生し、処理が停止します。

エラー・メッセージ

表 6-1 に、この機能についてホスト変数とインジケータ変数の不一致エラーを処理する際に作成される Embedded SQL の内部エラー・メッセージを示します。

表 6-1: 新しい内部エラー・メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_INVTYPE_V	インジケータ変数の不正な型が、構造体で見つかりました。	致命的なエラー	ホスト変数宣言とインジケータ宣言で同じインジケータ変数が使用されていることを確認する。
M_INVTYPE_VI	インジケータ構造体とホスト変数構造体の構造体要素の数が一致しません。	致命的なエラー	インジケータ構造体とホスト変数構造体で同数の要素を宣言する。
M_INVTYPE_VII	インジケータ配列とホスト変数構造体の要素の数が一致しません。	致命的なエラー	インジケータ配列とホスト変数構造体で同数の要素を宣言する。

制限事項

ホスト変数構造体を持つ単一ホスト変数または単一インジケータ変数を、インジケータ配列またはインジケータ構造体と混合することはできません。

fetch into：バッチ配列

fetch は、指定された数のローを現在アクティブな集合から返します。fetch は、実行されるたびにあとに続くバッチのローを返します。たとえば、現在アクティブな集合に 150 のローがあり、60 のローを選択してフェッチする場合、最初の fetch は、最初の 60 ローを返します。次の fetch は、それに続く 60 ローを返します。3 番目の fetch は、最後の 30 ローを返します。

注意 フェッチされたローの総数を知る方法については、「SQLCA 変数」(16 ページ) で説明しているように、SQLCA の *SQLERRD* 変数を参照してください。

次に、配列を選択する例を示します。

```
exec sql begin declare section end-exec
     TITLEID-ARRAY PIC X(6) occurs 100 times.
exec sql end declare section end-exec
...
exec sql
select title_id into :titleid_array
             from titles
end-exec
IF (SQLERRD OF SQLCA LESS THAN 50)
     DISPLAY "No of title_ids is less than 50");
ENDIF.
```

カーソルと配列

配列に返されるローの数がわからない場合は、「fetch into 配列」という方法を使用します。カーソルを宣言し、オープンしてから、fetch を使って ローのグループを取得してください。fetch into で配列が保持できるロー数よりも多くのローを返そうとした場合、その文は最小の配列が保持できる最大ロー数を返し、SQLCODE はエラーまたは例外を示す負の値を表示します。

カーソルの使用

カーソルは、複数のデータ・ローをホスト・プログラムに渡すときに、一度に 1 つのローを選択します。カーソルは、データの最初のローである「現在のロー」を示し、そのローをホスト・プログラムに渡します。次の fetch 文によってカーソルは次のローへ進み、今度はこのローが現在のローになります。この処理は、要求されたすべてのローがホスト・プログラムに渡されるまで続きます。

カーソルは、select 文が複数のデータ・ローを返す場合に使用してください。Client-Library は Adaptive Server Enterprise の返した複数のローを追跡し、アプリケーション用にバッファリングします。カーソルを使用してデータを取得するには fetch 文を使用します。

カーソルのメカニズムは、次に挙げる文から構成されます。

- declare
- open
- fetch
- update と delete where current of
- close

カーソルのスコープ規則

カーソル宣言のスコープは、それが宣言されているファイルになります。カーソルの open 文は、そのカーソルが宣言されているファイル内で使用してください。カーソルをオープンした場合、スコープは、そのカーソルがオープンされた接続になります。

同じカーソル名を複数の接続に対してオープンできます。カーソルのフェッチ、更新、削除、クローズ操作は、カーソルが宣言された以外のファイルで行うことができます。ただし、これらの操作が、カーソルがオープンされたのと同じ接続に対して実行される場合にかぎります。

カーソル名はプログラム内でユニークでなければなりません。実行時にアプリケーションが、同一の名前を持つ 2 つのカーソルを宣言しようとすると、アプリケーションは次のエラー・メッセージを示して失敗します。

```
There is already another cursor with the name 'XXX'.
```

カーソルの宣言

declare cursor 文は、実行文ではなく宣言です。そのため、ファイルのどこにでも置くことができますが、SQLCODE、SQLSTATE、SQLCA は、この文の後では設定されません。

カーソルは、データから複数のローを返す各 select 文に対して宣言します。カーソルは、使用する前に宣言しておいてください。declare セクション内では、カーソルは宣言できません。

カーソルを宣言する構文は、次のとおりです。

```
exec sql declare cursor_name cursor
     for select_statement end-exec.
```

パラメータの意味は次のとおりです。

- *cursor_name* には、カーソルの名前を指定します。ユニークな名前を最大 255 文字で指定する必要があります。また、英字または記号 # か _ で始めてください。
- select_statement は、データから複数のローを返す select 文です。select の構文の詳細については、『ASE リファレンス・マニュアル』を参照してください。ただし、into 句または compute 句は使用できません。

例：カーソルの宣言

次に、カーソルを宣言する例を示します。

```
exec sql declare C1 cursor for
 select type, price from titles
 where type like :WK-TYPE end-exec
```

上の例で C1 は、type カラムと price カラムで返されるローに対するカーソルとして宣言されています。プリコンパイラは、declare cursor 文に対するコードは生成しません。プリコンパイラは、単にカーソルに対応する select 文を格納します。

カーソルをオープンすると、select 文または declare cursor 文中のプロシージャが実行されます。そこでデータがフェッチされると、結果がホスト変数にコピーされます。

注意 各カーソルの open 文と declare 文は、同じファイルで指定してください。declare 文中のホスト変数は、open 文を定義したスコープと同じスコープになるようにしてください。ただし、一度カーソルをオープンすれば、どのファイルの中でもそのカーソルの fetch および update/delete where current of を実行できます。

スクロール可能カーソルの宣言

スクロール可能カーソルを宣言する構文は、次のとおりです。

```
exec sql declare  cursor_name  [cursor sensitivity] [cursor
scrollability] cursor
     for  select_statement ;
```

パラメータの意味は次のとおりです。

- *cursor_name* には、カーソルの名前を指定します。ユニークな名前を最大 255 文字で指定する必要があります。また、英字または記号 “#” か “_” で始めてください。
- *cursor sensitivity* は、カーソルの変更反映の可否を指定します。オプションは次のとおりです。
  - semi_sensitive − declare 文で semi_sensitive を指定すると、スクロール可能であることが暗黙的に設定されます。カーソルは、semi_sensitive (半反映型) のスクロール可能な読み取り専用カーソルになります。
  - insensitive − declare 文で insensitive を指定すると、カーソルは非反映型になります。スクロールの可能性は、declare 部分に SCROLL を指定することにより決定されます。SCROLL を省略するか、NOSCROLL を指定すると、カーソルには insensitive だけが設定され、非スクロール可能カーソルになります。このカーソルは読み込み専用でもあります。

  cursor sensitivity を指定しない場合、カーソルは非スクロール可能な読み込み専用カーソルになります。
- *cursor scrollability* には、カーソルがスクロール可能かどうかを指定します。オプションは次のとおりです。
  - scroll − declare 文で scroll を指定し、変更反映の可否を指定しない場合、カーソルは非反映型でスクロール可能になります。このカーソルは読み込み専用でもあります。
  - no scroll − SCROLL オプションを省略するか、NOSCROLL を指定した場合は、読み込み専用の非スクロール可能カーソルになります。カーソルの動作については、上の *cursor sensitivity* の説明を参照してください。

  cursor scrollability を指定しない場合、カーソルは非スクロール可能な読み込み専用カーソルになります。
- select_statement は、データから複数のローを返す select 文です。select の構文の詳細については、『ASE リファレンス・マニュアル』を参照してください。ただし、into 句または compute 句は使用できません。

カーソルのオープン

選択されたローの内容を取得するには、最初にカーソルをオープンする必要があります。open 文は、declare 文のカーソルに対応する select 文を実行します。

カーソルをオープンするときの open 文の構文は次のとおりです。

```
exec sql open cursor_name [ROW_COUNT = size] end-exec.
```

注意 配列がホスト変数として使用される場合、および複数行の検索が必要な場合は、ROW_COUNT を指定する必要があります。

カーソルを宣言すれば、select 文を発行したいときにいつでもカーソルを開くことができます。open 文を実行すると、Embedded SQL が、declare cursor 文の where 句で参照されたすべてのホスト変数の値を置き換えます。

オープンできるカーソルの数は、現在のセッションのリソース要求に依存します。Adaptive Server Enterprise は、オープンできるカーソルの数を制限しません。ただし、現在オープンしているカーソルをオープンすることはできません。このような場合には、エラー・メッセージが表示されます。

アプリケーションの実行中に、必要なだけカーソルをオープンできます。しかし、再びオープンする場合には、その前にクローズしなければなりません。現在のカーソルの結果セットからすべてのローを取得しなくても、別のカーソルの結果セットからローを取得できます。

カーソルを使用したデータのフェッチ

カーソルを使用してデータを取得し、そのデータをホスト変数に割り当てるには fetch 文を使用します。fetch 文の構文は次のとおりです。

```
exec sql [at connect_name] fetch cursor_name
 into :host_variable
 [[ indicator]: indicator_variable ]
 [,: host_variable
 [[ indicator]: indicator_variable ]...];
```

ただし、結果ローの各カラムに対して 1 つの *host_variable* が必要です。

各ホスト変数の前にはコロンを付け、次のホスト変数との間にカンマを入れてください。fetch 文にリストされたホスト変数は、select 文が取得する Adaptive Server Enterprise の値と一致している必要があります。つまり、変数の数は戻り値の数と同じで、同じ順序で並び、互換性のあるデータ型でなければなりません。

indicator_variable は、前の declare セクションで宣言された 2 バイトの符号付き整数です。Adaptive Server Enterprise から取得された値が null の場合、実行時のシステムは対応するインジケータ変数を -1 に設定します。それ以外の場合、インジケータを 0 に設定します。

fetch 文が取得するデータは、カーソルの位置に依存します。カーソルは、「現在のロー」を指します。fetch 文は、常に現在のローを返します。最初の fetch は最初のローを取得し、その値を示されたホスト変数にコピーします。fetch を行うごとにカーソルは、次の結果ローに進みます。

通常、fetch 文は、select 文で返されたすべての値がホスト変数に割り当てられるように、ループの中に置きます。一般的に使用されるループは次のとおりです。

```
   exec sql
        whenever sqlerror perform err-para thru err-para-end
    end-exec.
        exec sql
        whenever not found go to read-end
        end-exec.

* 0 is never equal to 1, so the perform will run
* until the whenever NOT FOUND clause causes
* a jump to READ-END

     PERFORM READ-PARA UNTIL 0 = 1.

 READ-END.
       . . .

 READ-PARA.
     exec sql fetch cursor_name into host-variable-list
     end-exec.
         . . .
 OTHER-PARA.
     . . .
```

上のループは、すべてのローが返されるか、エラーが発生するまで続きます。どちらの場合も、SQLCODE または SQLSTATE がループの終了した理由を示します。これらは、フェッチのたびに whenever 文によってチェックされます。エラー処理ルーチンを使用すれば、どちらかの条件が発生したとき何らかのアクションが実行されるようにできます。詳細については、「第 8 章 エラーの処理」を参照してください。

スクロール可能カーソルを使用したデータのフェッチ

カーソルを使用してデータを取得し、そのデータをホスト変数に割り当てるには fetch 文を使用します。fetch 文の構文は次のとおりです。

```
exec sql [at connect_name] fetch [fetch
orientation]cursor_name
 into : host_variable
 [[ indicator]: indicator_variable ]
 [,: host_variable
 [[ indicator]: indicator_variable ]...];
```

結果ローの各カラムに対して 1 つの *host_variable* が必要です。

各ホスト変数の前にはコロンを付け、次のホスト変数との間にカンマを入れてください。fetch 文にリストされたホスト変数は、select 文が取得する Adaptive Server Enterprise の値と一致している必要があります。つまり、変数の数は戻り値の数と同じで、同じ順序で並び、互換性のあるデータ型でなければなりません。

fetch orientation には、カーソルがスクロール可能である場合のローのフェッチ方向を指定します。オプションは次のとおりです。NEXT、PRIOR、FIRST、LAST、ABSOLUTE *fetch_offset*、RELATIVE *fetch_offset* です。フェッチ方向を指定しない場合のデフォルトは next です。フェッチ方向を指定するには、カーソルがスクロール可能でなければなりません。

fetch 文が取得するデータは、カーソルの位置に依存します。fetch 文は、通常、カーソルをオープンするときに指定された ROW_COUNT に応じて、カーソル結果セットから単一または複数のローを取得します。カーソルがスクロール可能でない場合、fetch は、結果セットの次のローを取得します。カーソルがスクロール可能な場合、フェッチされるローの位置は、fetch 文に含まれるコマンドによって指定されます。

スクロール可能カーソルの宣言とローのフェッチの例

スクロール可能カーソルを宣言し、無作為にローをフェッチするには、カーソルの宣言で変更反映の可否とスクロール可能性を指定した後、フェッチ時にフェッチ方向を指定します。次の例は、insensitive (非反映型) スクロール可能カーソルを宣言し、無作為にローをフェッチする方法を示しています。

```
exec sql declare c1 insensitive scroll cursor for
 select title_id, royalty, ytd_sales from authors
 where royalty < 25;
exec sql open c1;
```

この例では、カーソル宣言に scroll と insensitive が指定されています。フェッチ方向は fetch 時に指定可能であり、結果セットからフェッチする必要があるローを示します。

カーソルをスクロール可能と宣言してオープンしたら、フェッチ時に FETCH の方向を指定して、結果セットから取得するローを指定します。

次の fetch 例では、結果セットから先頭ローの指定されたカラムをフェッチします。

```
exec sql fetch first from c1 into :title,:roy,:sale;
```

次の fetch 例では、結果セットから前のローの指定されたカラムをフェッチします。

```
exec sql fetch prior from c1 into :title,:roy,:sale;
```

次の fetch 例では、結果セットからロー 20 の指定されたカラムをフェッチします。

```
exec sql fetch absolute 20 from c1 into :title,:roy,:sale;
```

フェッチ文で有効なローが返されたかどうかを判別するには、*sqlcode* または *sqlstate* を使用します。スクロール可能カーソルでは、カーソルが結果セット境界の外 (先頭ローの前や最終ローの後ろなど) に位置している場合、フェッチされるローが 0 行になる可能性があります。このような状況では、0 行のローのフェッチはエラーではありません。

カーソルを使用したローの更新と削除

カーソルの現在のローを更新または削除するには、update または delete 文内に探索条件として where current of *cursor_name* を指定します。

カーソルを使ってローを更新するには、更新に使われる結果カラムが、更新可能になっていなければなりません。max(colname) のような SQL 式の結果は使えません。つまり、結果カラムと更新されるデータベース・カラムに有効な対応がなければなりません。

次は、カーソルを使用してローを更新する方法の例です。

```
        exec sql declare c1 cursor for
              select title_id, royalty, ytd_sales
              from titles
              where royalty < 12
              end-exec

         exec sql open C1 end-exec

         PERFORM READ-PARA UNTIL SQLCODE = 100.
         exec sql close C1 end-exec.
         STOP RUN.
     READ-PARA.
         exec sql fetch C1 into :TITLE-ID, :ROYALTY,
              :SALES end-exec.
         IF SALES > 10000
           exec sql update titles
                 set royalty = :roy + 2
              where current of C1 end-exec.
```

Embedded SQL での update と delete 文の構文は、where current of *cursor_name* による探索条件を含めて、Transact-SQL と同じです。

テーブルの更新プロトコルの決定とロックの詳細については、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。

カーソルのクローズ

オープンされたカーソルをクローズするには、close 文を使用します。close 文の構文は次のとおりです。

```
exec sql [at connection] close cursor_name end-exec
```

クローズしたカーソルを再使用するには、open 文を発行します。再びカーソルをオープンしたとき、そのカーソルは最初のローを指します。オープンしていないカーソルに対して、close 文を発行しないでください。発行した場合は、エラーが起こります。

カーソルの例

次の例では、2 つのカーソルをネストする方法を示します。カーソル C2 は、カーソル C1 から *TITLE-ID* にフェッチされた値に依存します。

このプログラムでは、*TITLE-ID* の値を、宣言時ではなくオープン時に取得します。

```
          ...
     exec sql declare C1 cursor for
         select title_id, title, royalty from titles
     end-exec

     exec sql declare C2 cursor for
         select au_lname, au_fname, from authors
         where au_id in
             (select au_id from titleauthor
              where title_id = :TITLE-ID)
     end-exec

     exec sql open C1 end-exec.

     PERFORM READ-TITLE UNTIL SQLCODE = 100.

READ-END.
 . . .

READ-TITLE.
     exec sql fetch C1 into
         :TITLE-ID, :TITLE, :ROYALTY end-exec.
     IF SQLCODE NOT = 100
          MOVE ROYALTY TO DISP-ROY
          DISPLAY "Title ID: " TITLE-ID
         ", Royalty:" DISP-ROY
          IF ROYALTY > 10
             exec sql open C2 end-exec
             PERFORM READ-AUTH UNTIL SQLCODE = 100
             exec sql close C2 end-exec.
```

```
READ-AUTH.
    exec sql fetch C2 into :AU-LNAME, :AU-FNAME
    end-exec
    IF SQLCODE NOT = 100
         DISPLAY "  AUTHOR: " AU-LNAME " "
           AU-FNAME.
```

カーソルを使用した別の例については、オンライン・プログラム例を参照してください。オンライン・サンプルへのアクセスの詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

ストアド・プロシージャの使用

ストアド・プロシージャには、2 つの種類があります。ユーザ定義のストアド・プロシージャとプリコンパイラが生成するストアド・プロシージャです。どちらも Adaptive Server Enterprise がクエリをあらかじめ最適化するので、個々に独立した文よりも高速に実行されます。ユーザがユーザ定義のストアド・プロシージャを生成し、プリコンパイラがストアド・プロシージャを生成します。

ユーザ定義のストアド・プロシージャ

Embedded SQL を使用して、データ・ローを返す select 文を含むストアド・プロシージャを実行できます。ストアド・プロシージャは、プログラムに出力パラメータとリターン・ステータス変数を使って結果を返します。

ストアド・プロシージャのパラメータは、入力、出力、または入出力の両方として使用できます。ストアド・プロシージャの詳細については、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。

構文

有効なストアド・プロシージャ名として含むことができるのは、大文字または小文字の英字、および $、_、# です。

ストアド・プロシージャに use 文を含めることはできません。

ストアド・プロシージャを実行するには、次の構文を使用します。

```
exec sql [at connection_name]
  exec [:status_variable = status_value] procedure_name
 [([[@parameter_name =]parameter_value [out[put]]],...)]
  [into :hostvar_1 [:indicator_1]
  [, hostvar_n [indicator_n, ...]]]
  [with recompile]
 end-exec
```

パラメータの意味は、次のとおりです。

- *status_variable* では、Adaptive Server Enterprise リターン・ステータス値またはリターン・コードを返すことができます。これらは、ストアド・プロシージャが正常に終了したこと、または失敗の理由を示します。負のステータス値は、Adaptive Server Enterprise 用に予約されています。ストアド・プロシージャのリターン・ステータス値のリストについては、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。
- *status_value* は、ストアド・プロシージャのリターン・ステータス変数 *status_variable* の値です。
- *procedure_name* は、実行するストアド・プロシージャの名前です。
- *parameter_name* は、ストアド・プロシージャの変数の名前です。*@parameter_name* フォーマットを使用し、位置または名前のどちらかを使用して、パラメータを渡すことができます。1 つのパラメータの名前を指定した場合は、すべてに名前を指定してください。『ASE Transact SQL ユーザーズ・ガイド』を参照してください。
- *parameter_value* はリテラル定数またはホスト変数であり、その値がストアド・プロシージャに渡されます。ホスト変数の場合、インジケータを対応させることができます。この変数は対応するキーワードを持たないことに注意してください。
- output は、ストアド・プロシージャがパラメータ値を返すことを示します。ストアド・プロシージャ内の対応するパラメータも、output キーワードを使用して作成されている必要があります。
- into*:hostvar_1* を指定すると、ストアド・プロシージャから返されるロー・データは、指定されたホスト変数 (*hostvar_1* ～ *hostvar_n*) に保管されます。それぞれのホスト変数にはインジケータ変数を指定できます。
- *indicator_n* は、前の declare セクションで宣言された 2 バイトのホスト変数です。対応する *hostvar_n* の値が null の場合、ロー・データの取得時にインジケータ変数は -1 に設定されます。トランケーションが発生すると、インジケータ変数は結果カラムの実際の長さに設定されます。トランケーションがなければ、インジケータ変数は 0 になります。
- with recompile を指定すると、ストアド・プロシージャが実行されるたびに Adaptive Server Enterprise はこのプロシージャのための新しいクエリ・プランを作成します。

注意 Embedded SQL 内では、ストアド・プロシージャを実行するのに exec キーワードが必要です。exec の代わりに execute は使えません。

ストアド・プロシージャの例

次に、ストアド・プロシージャへの呼び出しの例を示します。ここで、*RET-CODE* はステータス変数、a_proc はストアド・プロシージャ、*PAR-1* は入力パラメータ、*PAR-2* は出力パラメータです。

```
exec sql begin declare section end-exec
  01  PAR-1              PIC S9(9) COMP.
  01  PAR-2              PIC S9(9) COMP.
  01  RET-CODE           PIC S9(4) COMP.
 exec sql end declare section end-exec
     . . .
     exec sql exec :RET-CODE=a_proc :PAR-1,
                   :PAR-2 out end-exec.
```

次の例は、データ・ローを取得するストアド・プロシージャの使用方法を示します。ストアド・プロシージャの名前は “*get_publishers*” です。

```
exec sql begin declare section end-exec.
  01  PUB-ID             PIC X(4).
  01  NAME               PIC X(45).
  01  CITY               PIC X(25).
  01  STATE              PIC X(2).
  01  RET-CODE           PIC S9(9).
 exec sql end declare section end-exec.
    . . .
 exec sql exec :RET-CODE = get_publishers :PUB-ID
                    into :NAME :CITY :STATE END-EXEC.
```

exec 文の詳細については、「第 10 章 Open Client/Server 設定ファイル」を参照してください。

規約

ストアド・プロシージャ中のパラメータのデータ型は、COBOL ホスト変数と互換性がなければなりません。Client-Library は、ある決まった組み合わせだけしか変換しません。互換データ型の互換性については、「第 4 章 変数の使い方」にある表を参照してください。

プリコンパイラが生成するストアド・プロシージャ

オプションのコマンド・ライン・スイッチを設定すると、プリコンパイラは、プログラム中の Transact-SQL 文の実行を最適化できるストアド・プロシージャを自動的に生成します。

プリコンパイラのコマンド・ライン・オプション・スイッチのリストについては、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

プリコンパイラが生成するストアド・プロシージャを起動するには、次の手順に従ってください。

1 適切なコマンド・ライン・スイッチを設定して、プリコンパイラが自動的に最適化する Transact-SQL 文に対するストアド・プロシージャを生成します。

プリコンパイラは、isql ファイルを生成します。このファイルには、ストアド・プロシージャを生成する文が含まれています。

2 対話型 SQL (isql プログラム) を使用して、そのファイルを実行します。

Adaptive Server Enterprise にストアド・プロシージャをロードします。また、プリコンパイラは、ストアド・プロシージャの呼び出しを出力ファイルに生成します。

デフォルトでは、プリコンパイラが生成するストアド・プロシージャの名前は、ソース・プログラムの名前から拡張子を除いた名前になります。ストアド・プロシージャは連続した番号を持ち、ファイル名と番号はセミコロン (;) によって区切られます。

たとえば、ソース・プログラムの名前が *test1.pco* のとき、そのストアド・プロシージャの名前は *test1;1* から順に *test1;n* となります。ここで *n* は、ソース・プログラムの最後のストアド・プロシージャの番号です。

オプションとして、ストアド・プロシージャの名前を変更するコマンド・ライン・フラグがあります。このフラグを使用すると、変更したアプリケーションをテストする際に、生成済みのストアド・プロシージャを削除する必要がありません。そのアプリケーションのテストが終わったあとにフラグを設定せずにプリコンパイルすれば、ストアド・プロシージャをインストールできます。

注意 declare cursor 文を発行する場合は、select 句だけがストアド・プロシージャとして保存されます。アプリケーションに構文エラーがあると、プリコンパイラはファイルとストアド・プロシージャのどちらも作成しません。

文のグループ化

文を実行する場合には、バッチまたはトランザクションによってその文をグループ化できます。

バッチによる文のグループ化

バッチとは、ひとまとまりで実行させる文の集合です。プリコンパイラは、exec sql キーワードと end-exec キーワードの間にあるすべての Transact-SQL 文をバッチ・モードで実行します。

プリコンパイラはストアド・プロシージャを保存しますが、再度実行できるようにバッチを保存することはできません。バッチは、現在の実行にのみ有効です。

プリコンパイラは、結果セットを返さないバッチ・モード文だけをサポートしています。

```
exec sql insert into TABLE1 values (:val1)
     insert into TABLE2 values (:val2)
     insert into TABLE3 values (:val3)
 end-exec.
```

上記の 3 つの insert 文は 1 つのグループとして処理されます。これで個別に処理するよりも効率的になります。バッチ内でエラー処理を行うには、get diagnostics メソッドを使用します。詳細については、「get diagnostics の使い方」(91 ページ) を参照してください。

これらの文は結果を返さないので、バッチ内で使用できます。詳細については、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。

トランザクションによる文のグループ化

「トランザクション」とは、1 文または複数文から構成される、実行の 1 つの単位をいいます。トランザクション内の文は 1 つのグループとして実行されるので、それらの文すべてが実行されるか、1 つも実行されないかのどちらかになります。

プリコンパイラは、デフォルトの ANSI/ISO とオプションの Transact-SQL という 2 つのトランザクション・モードをサポートします。Transact-SQL トランザクション・モードでは、begin transaction 文が前に置かれていないと、それぞれの文は暗黙的にコミットされます。

Transact-SQL モードはシステム・リソースを少ししか使用しないのに対して、デフォルトの ANSI/ISO トランザクション・モードはシステム応答時間に著しく影響します。使用するアプリケーションにとって適切なモード選択の詳細については、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。

プリコンパイラ・オプションを使用して、アプリケーションがオープンする接続に対する「トランザクション・モード」を決定できます。詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

Transact-SQL トランザクション・モード

このオプションのトランザクション・モードでは、Embedded SQL の構文は Transact-SQL で使用される構文と同じです。begin transaction 文が明示的にトランザクションを開始します。

Embedded SQL のトランザクション文の構文は、次のとおりです。

```
exec sql [at connect_name]
     begin transaction [transaction_name] end-exec

exec sql [at connect_name]
     save transaction [savepoint_name] end-exec
exec sql [at connect_name] commit transaction
     [transaction_name] end-exec
exec sql [at connect_name] rollback transaction
     [savepoint_name | transaction_name] end-exec
```

注意 disconnect は、すべてのオープン・トランザクションをロールバックします。disconnect 文の詳細については、「第 5 章 Adaptive Server Enterprise との接続」を参照してください。

ある接続で begin transaction を発行した場合は、同じ接続で save transaction、commit transaction、または roll back transaction のいずれかを発行してください。そうしないと、エラーが発生します。

デフォルトの ANSI/ISO トランザクション・モード

ANSI/ISO SQL では、save transaction または begin transaction 文がサポートされていません。その代わりに、アプリケーション・プログラムが次に挙げる文のどれかを実行すると、トランザクションは暗黙的に開始されます。

- delete
- insert
- select
- update
- open
- exec

commit work 文または rollback work 文を発行すると、そのトランザクションは明示的に終了します。ANSI/ISO フォーマットの commit 文および rollback 文を使用する必要があります。

構文は次のとおりです。

```
exec sql commit [work] end-exec
exec sql rollback [work] end-exec
```

拡張トランザクション

「拡張トランザクション」は、複数の Embedded SQL 文を含む作業の単位です。Transact-SQL トランザクション・モードでは、拡張トランザクション文を begin transaction および commit transaction 文で囲んでください。

デフォルトの ANSI モードでは、常に拡張トランザクションを実行していることになります。commit work 文を発行した場合は、現在の拡張トランザクションが終了し、別の拡張トランザクションが開始されます。詳細については、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。

注意 allow ddl in tran データベース・オプションが設定されていないかぎり、ANSI モードの拡張トランザクションで Transact-SQL 文として alter database、create database、create index、create table、create view、disk init、grant、load database、load transaction、revoke、truncate table、および update statistics を使用しないでください。

ファイルとディレクトリの取り込み

include 文は、ファイルの検索とコピーがプリコンパイル時に起こるという点を除けば、COBOL COPY コマンドと基本的に同じです。include は、プリコンパイル時に、ディレクトリ内のファイル、またはプリコンパイル文に指定されたディレクトリを検索します。各環境でプリコンパイル文と COBOL コンパイラを使用する方法については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

Embedded SQL の include 文を使用すると、COBOL の COPY コマンドを使用した場合と同様に、共通データ定義などのソース・コード・ファイルを各自のアプリケーションに追加できます。したがって、次の例は有効です。

```
exec sql include "myfile" end-exec.
```

プリコンパイラは、include 文を COBOL の COPY コマンドに変更して、ファイル名を引用符で囲みます。

また、プリコンパイラのコマンド・オプションを設定して、インクルード・ファイルのディレクトリを指定することもできます。プリコンパイル時に、プリコンパイラは、COBOL のコンパイル・コマンドに指定されたパスを検索します。このオプションを使用してディレクトリを指定すると、プリコンパイラは、そのディレクトリをファイル名に追加して、パス名全体を引用符で囲みます。これにより、ファイルのパスは、ターゲット・プログラムにハードコード化されます。詳細については、『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 7 章

動的 SQL の使い方

この章では、動的 SQL について説明します。動的 SQL とは、Embedded SQL アプリケーションのユーザが、アプリケーションの実行中に SQL 文を入力できるようにする高度な手法です。通常は静的 SQL で十分ですが、動的 SQL を使用すると、実行時にさまざまな SQL 文を柔軟に構築できます。

動的 SQL は、一連の Embedded SQL 文からなり、これを使用してオンライン・アプリケーションのユーザは実行時に対話的にデータベースへアクセスできます。

次の条件のいずれかが実行時までわからないときに、動的 SQL を使用します。

- ユーザが実行する SQL 文
- カラム、インデックス、テーブルの参照
- ホスト変数の数またはデータ型

動的 SQL は ANSI および ISO SQL2 標準の一部です。動的 SQL は対話型アプリケーションの実行に役立ちます。

動的 SQL を使用するとき

アプリケーションが少数の SQL 文のみを実行する場合は、プログラム内に埋め込むことができます。しかし、アプリケーションがさまざまなタイプの SQL 文を実行する場合には、SQL 文を構築し、それらを動的にバインドして実行すると便利です。

次のような状況では、動的 SQL の利点を活用できます。たとえば、アプリケーション・プログラムが本の売り上げに関する書店のデータベースを検索するとします。見込み顧客は、価格、主題、装丁の種類、ページ数、発行日付、言語などのいろいろな基準を適用できます。

顧客が「10 ドルから 20 ドルまでのビジネスに関するノンフィクションの本が欲しい」と要求しているとします。この要求は、Transact-SQL 文で簡単に表現できます。

```
select * from titles where
 type = "business"
 and price between $10 and $20
```

すべての顧客の検索条件を予想することは不可能です。そのため、動的 SQL を使わないと、Embedded SQL プログラムでは、1 つの「クエリ」を使用して該当する本のリストを生成するのは困難です。

動的 SQL を使えば、書店はそれぞれの顧客に対して別々の where 句の探索条件でクエリを入力できます。書店は、発行日付、本のカテゴリー、その他のデータに基づいて要求を変えたり、表示されるカラムを変えたりできます。次に例を示します。

```
select * from titles
     where type = ?
     and price between ?and ?
```

疑問符 (“?”) は、動的パラメータのマーカで、ユーザが検索値を入力できる場所を表します。

動的 SQL プロトコル

注意 実行時まで文が完成されないので、プリコンパイラは、動的 SQL 文に対するストアド・プロシージャを生成しません。実行時に、Adaptive Server Enterprise が、一時的なストアド・プロシージャとして tempdb データベースに記録します。tempdb データベースにはユーザ名 "guest" が指定されている必要があり、この "guest" アカウントは create procedure パーミッションを持っている必要があります。そうでない場合は、これらのテンポラリ・ストアド・プロシージャの 1 つを実行しようとすると、「サーバのユーザ id *user_id* は、データベース *database_name* の中では有効なユーザではありません。」というエラー・メッセージが表示されます。*user_id* はユーザのユーザ ID であり、*database_name* はユーザのデータベースの名前です。

動的 SQL の prepare 文は、サーバに実際の SQL 文を送ります。この文には、任意のデータ定義言語 (DDL)、データ操作言語 (DML) の文、または create procedure 以外の Transact-SQL 文のいずれかを使用できます。

動的 SQL は、次のように動作します。

1　入力データを SQL 文に変換します。

2　SQL 文が動的に実行できるかどうかを確認します。

3　SQL 文を Adaptive Server Enterprise に送り、実行の準備をします。Adaptive Server Enterprise は、受け取った文をコンパイルし、一時的なストアド・プロシージャとして保存します (メソッド 2、3、4 のとき)。

4　すべての入力パラメータまたは記述子をバインドします (メソッド 2、3、4 のとき)。

5　文を実行します。

select リストが変化する select 文に対しては、返されるデータ項目とローの参照に記述子が使用されます (メソッド 2 または 4 のとき)。

6　出力パラメータまたは記述子をバインドします (メソッド 2、3、4 のとき)。

7　結果を取得します (メソッド 2、3、4 のとき)。

8　Adaptive Server Enterprise のストアド・プロシージャを再度アクティブにして、文を削除します (メソッド 2、3、4 のとき)。

9　Adaptive Server Enterprise と Client-Library からのすべてのエラーと警告条件を処理します。

メソッド 1：execute immediate の使い方

ホスト変数またはリテラル文字列に格納された Transact-SQL 文を Adaptive Server Enterprise に送るには、execute immediate を使用してください。文は、結果を返しません。このメソッドでは、select 文は実行できません。

動的に入力された文は、セッション中にユーザが起動した回数だけ実行されます。このメソッドでは、次の動作が実行されます。

1　Embedded SQL プログラムが Adaptive Server Enterprise にテキストを渡します。

2　Adaptive Server Enterprise は、文が動的に実行可能で、ローを返さないことを確認します。

3　Adaptive Server Enterprise が文をコンパイルして実行します。

execute immediate を使用すると、ユーザは Transact-SQL 文のすべてまたは一部を入力できます。

execute immediate の構文は、次のとおりです。

```
exec sql [at connection_name] execute immediate
     {:host_variable | "string"} end-exec
```

各パラメータの意味は、次のとおりです。

- *host_variable* は、declare セクションで定義された文字列変数です。execute immediate を呼び出す前に、ホスト変数は、文法的に正しい完全な Transact-SQL 文に設定されていなければなりません。
- *string* は、*host_variable* の代わりに使われるリテラル Transact-SQL 文の文字列です。

Embedded SQL は、*host_variable* または文字列を処理したり、チェックしたりせずに Adaptive Server Enterprise に送ります。文が結果を返すか、または失敗した場合は、エラーが発生します。文の実行後に SQLCODE の値をテストしたり、エラー・ハンドラを設定するのに、whenever 文を使用したりできます。Embedded SQL プログラムでのエラー処理の詳細については、「第 8 章 エラーの処理」を参照してください。

メソッド 1：例

次の 2 つの例では、メソッド 1 の execute immediate の実際の使い方を説明します。最初の例では、ユーザに文の入力を求めて、その文を実行します。

```
exec sql begin declare section end-exec
01  CMD-1              PIC X(50).
01  SRC-COND           PIC X(50).
01  SQLSTR1            PIC X(200).
exec sql end declare section end-exec

DISPLAY "ENTER statement".
ACCEPT SQLSTR1.
exec sql execute immediate :SQLSTR1 end-exec.
```

次の例は、ユーザに探索条件の入力を求めて、更新する titles テーブルのローを指定します。そして、その探索条件を update 文に連結して、完全な文を Adaptive Server Enterprise に送ります。

```
MOVE "UPDATE titles SET price = price*1.10 WHERE "
     TO CMD-1.
DISPLAY "ENTER SEARCH CONDITION:".
ACCEPT SRC-COND.
STRING CMD-1 delimited by size SRC-COND DELIMITED BY
SIZE INTO SQLSTR1.
exec sql execute immediate :SQLSTR1 end-exec.
```

メソッド 2：prepare および execute の使い方

メソッド 2 の prepare および execute は、次のいずれかの場合に使用します。

- 取得するデータがないことが確かで、文を複数回実行する場合。
- select 文が単一のローを返す場合。このメソッドでは、カーソルと select 文を対応させることはできません。

この処理は、単一ロー select とも呼ばれます。複数のローを取得する場合は、メソッド 3 またはメソッド 4 を使ってください。

このメソッドは、prepare と execute を使用して、文を Adaptive Server Enterprise に送る前に COBOL 変数から Transact-SQL 文にデータを置換します。Transact-SQL 文は、文字バッファに COBOL 変数のどの値を置換すべきかを表す動的パラメータ・マーカとともに格納されます。

このような文が準備されるため、Adaptive Server Enterprise は文をコンパイルして、一時的なストアド・プロシージャとして保存します。そして、セッションの間、必要なだけ文を繰り返し実行します。

prepare 文は、バッファを文の名前と対応させて、文を実行する準備をします。execute 文は、COBOL 変数のリストからバッファに値を置換し、完全な文を Adaptive Server Enterprise に送ります。このメソッドで、すべての文を実行できます。

prepare

prepare 文の構文は次のとおりです。

```
exec sql [at connection_name] prepare
     statement_name from {:host_variable | "string"}
 end-exec
```

各パラメータの意味は、次のとおりです。

- *statement_name* は、最大 255 文字までの名前で文を識別します。これは COBOL 変数やリテラル文字列ではありません。これは、プリコンパイラが、execute 文を prepare 文に対応させるのに使用する記号名です。
- *host_variable* は、動的パラメータ・マーカです。

  標準 Embedded SQL 文では、動的パラメータ・マーカの前にはコロンを付けてください。
- *string* は、リテラル文字列で、*host_variable* の代わりに使用されます。

execute

execute 文の構文は次のとおりです。

```
exec sql [at connection_name] execute statement_name
     [into {host_var_list | sql descriptor
         descriptor_name | descriptor sqlda_name }]
     [using {host_var_list | sql descriptor
         descriptor_name | descriptor sqlda_name}]
 end-exec
```

各パラメータの意味は、次のとおりです。

- *statement_name* は prepare 文で割り当てられた名前です。into は、単一ロー select に使用されます。
- into は、単一ロー select に使われます。
- using は、*host_var_list* にある変数の動的パラメータ・マーカと置換される COBOL 変数または記述子を指定します。変数は、declare セクションで定義する必要があり、リストされている順番で置換されます。この句は文が動的パラメータ・マーカを使用する変数を含んでいるときにだけ必要です。
- *descriptor_name* は、動的 SQL 文の動的パラメータ・マーカの記述を保持しているメモリの領域を表します。
- *host_var_list* は、クエリ内でパラメータ・マーカ (“?”) に置換するホスト変数のリストです。
- *sqlda_name* は、SQLDA の名前です。

メソッド 2：例

次の例では、メソッド 2 の prepare および execute の実際の使い方を示します。この例では、titles テーブルのどのローを更新するかを決める where 句の入力が要求されます。たとえば、“1.1” と入力すれば、価格が 10 パーセントずつ増加します。

```
01  CUST-TYPE   PIC X.
     88  BIG-CUSTOMER   VALUE "B".
     88  OTHER-CUSTOMER VALUE "O".
 . . .
 exec sql begin declare section end-exec
 01  MULTIPLIER         PIC S9(2) COMP.
 01  CMD-1              PIC X(50).
 01  SRC-COND           PIC X(50).
 01  SQLSTR1            PIC X(200).
 exec sql end declare section end-exec

     MOVE "UPDATE titles SET
         " price = price + (price * ? / 100)
         WHERE "
         TO CMD-1.
```

```
        DISPLAY "ENTER SEARCH CONDITION:".
        ACCEPT SRC-COND.
        STRING CMD-1 SRC-COND DELIMITED BY SIZE
            INTO SQLSTR1.

        exec sql prepare statement1 from :SQLSTR1
        end-exec.

        IF BIG-CUSTOMER
            MOVE 10 TO MULTIPLIER
        ELSE
            MOVE 25 TO MULTIPLIER.

    exec sql execute statement1 using :MULTIPLIER
    end-exec.
```

メソッド 3：カーソルによる prepare および fetch の使い方

メソッド 3 は、カーソル文とともに prepare 文を使用して select 文の結果を返します。このメソッドは、select リストが固定しており、複数のローを返す select 文に対して使います。つまり、前もってアプリケーションが返される select カラム・リスト属性の数とデータ型を確認している場合に使います。結果を取り込むためにホスト変数を予測して定義する必要があります。

メソッド 3 の場合、文を実行するには、declare、open、fetch、close cursor 文を含めてください。このメソッドは、文が複数のローを返すので必要です。準備文の識別子と指定したカーソル名は対応しています。update および delete where current of というカーソル文も使用できます。

メソッド 2 のように、Transact-SQL select 文は、最初に文字ホスト変数または文字列に格納されます。これには、入力変数からの値をどこに置換するかを表す動的パラメータ・マーカを含めることができます。文には、prepare、declare、および open 文中でその文を識別するための名前が与えられています。

メソッド 3 には、5 つの手順が必要です。

1　prepare

2　declare

3　open

4　fetch (オプションとして update および delete)

5　close

次の項では、これらの手順について説明します。

prepare

prepare 文は、メソッド 2 で使われたものと同じです。詳細については、「prepare」(69 ページ) を参照してください。

declare

declare 文は、カーソルに対する標準の declare 文と似ています。ただし動的 SQL では、カーソルは select 文に対してではなく、準備された *statement_name* に対して宣言され、すべての入力ホスト変数は、declare 文ではなく open 文で参照されます。

動的 declare 文は、宣言というより実行文です。そのため、コード内で実行文が有効である場所に置かれなければならず、アプリケーションは宣言を実行したあとで、ステータス・コード (SQLCODE、SQLCA、または SQLSTATE) を確認する必要があります。

declare 文の動的 SQL の構文は、次のとおりです。

```
exec sql [at connection_name] declare cursor_name
     cursor for statement_name end-exec
```

各パラメータの意味は、次のとおりです。

- at *connection_name* は、カーソルが使用する Adaptive Server Enterprise 接続を指定します。
- *cursor_name* は、open、fetch、close 文で使用されるカーソルを指定します。
- *statement_name* は、prepare 文で指定される名前で、実行される select 文を表します。

open

open 文は、文バッファのすべての入力変数を置換し、その結果を実行するために Adaptive Server Enterprise に送ります。open 文の構文は次のとおりです。

```
exec sql [at connection_name] open cursor_name
    [using {host_variable_list |
     sql descriptor descriptor_name | descriptor sqlda_name}]
 end-exec
```

各パラメータの意味は、次のとおりです。

- *cursor_name* は、declare 文でカーソルに与えられた名前です。
- *host_variable_list* は、動的パラメータ・マーカに対する値を含むホスト変数の名前です。
- *descriptor_name* は、動的パラメータ・マーカに対する値を含む記述子の名前です。
- *sqlda_name* は、SQLDA の名前です。

fetch および *close*

カーソルがオープンしたあとに、結果セットがアプリケーションに返されます。データは、アプリケーション・プログラムのホスト変数にフェッチおよびロードされます。オプションとして、データの更新や削除もできます。fetch と close 文は、静的 Embedded SQL と同じです。

fetch 文の構文は次のとおりです。

```
exec sql [at connection_name] fetch cursor_name
    into :host_variable
         [[indicator]:indicator_variable]
     [,:host_variable
     [[indicator]:indicator_variable]...]
 end-exec
```

各パラメータの意味は、次のとおりです。

- *cursor_name* は、declare 文でカーソルに与えられた名前です。
- 結果ローの各カラムに対して 1 つの COBOL *host_variable* が必要です。変数は、declare セクションで定義してください。データ型にはカーソルで返される結果との互換性が必要です。

close 文の構文は次のとおりです。

```
exec sql [at connection_name] close cursor_name
end-exec
```

cursor_name は、declare 文でカーソルに割り当てられた名前です。

メソッド 3：例

次は、prepare および fetch を使用する例で、ユーザに select 文の order by 句の入力を求めます。

```
exec sql begin declare section end-exec
 01  AGE                PIC S9(2) COMP.
 01  R-AGE              PIC S9(2).
 01  ROYALTY            PIC S9(9) COMP.
 01  TITLE              PIC X(25).
 01  MANAGER            PIC X(25).
 01  SQLSTR2            PIC X(100).
 01  I-TITLE            PIC S9(4) COMP.
 01  I-AGE              PIC S9(4) COMP.
 exec sql end declare section end-exec

 01  DSP-AGE            PIC  9(2).
 01  DSP-ROYALTY        PIC -ZZZ,ZZZ,ZZZ.

 PROCEDURE DIVISION.

     MOVE 60 TO R-AGE.

MOVE "select age, royalty, title, manager from
 -          " inprogr where age !=?" TO SQLSTR2
 MOVE 0 TO I-AGE.
     exec sql prepare statement2 from :SQLSTR2
         end-exec.
     exec sql declare C1 cursor for statement2
         end-exec
     exec sql whenever not found goto NOT-FOUND
         end-exec
     exec sql open C1 using :R-AGE indicator :I-AGE
         end-exec.

 RET-LOOP.
     MOVE 0 TO I-TITLE.

     exec sql fetch C1 into
         :AGE, :ROYALTY,
         :TITLE indicator :I-TITLE,
         :MANAGER end-exec.

     MOVE AGE TO DSP-AGE.
     MOVE ROYALTY TO DSP-ROYALTY.
     IF I-TITLE = -1
         MOVE "Null" TO TITLE.

     DISPLAY "Age = " DSP-AGE
             " Royalty = " DSP-ROYALTY
             " Title = " TITLE
             " Manager = " MANAGER.
     DISPLAY " ".
     GO TO RET-LOOP.

 NOT-FOUND.
     exec sql close C1 end-exec.
```

メソッド 4：システム記述子による prepare および fetch の使い方

メソッド 4 では、select リストが変化する select 文を使用できます。つまり、アプリケーションを作成するときに、select 文が返す項目の型や数がわからなくてもよいということです。

どれだけの数の変数が必要であるか、また変数はどのようなデータ型でなければならないかが不明なため、前もってホスト変数を定義できないときにメソッド 4 を使用します。

メソッド 4：動的記述子

「動的記述子」とは、動的 SQL 文で使用される変数の記述を保持するデータ構造体です。動的記述子には、SQL 記述子と SQLDA 構造体の 2 種類があります。この 2 種類の動的記述子については、このあとの項を参照してください。

カーソルは、オープンされるときにそのカーソルに対応する入力記述子を持つことができます。入力記述子には、動的 SQL 文のパラメータ・マーカに代入される値が含まれます。

ユーザがパラメータの数と、それぞれのパラメータについてのデータ型、長さ、精度、位取り、インジケータ、データなどを含む適切な情報を使用して入力記述子を指定したあとに、カーソルがオープンされます。

出力記述子は fetch 文と対応していて、fetch 文を実行した結果生成されるデータを保持しています。Adaptive Server Enterprise は、データ型と返される実際のデータを含むデータ項目の属性を読み込みます。SQL 記述子を使用する場合は、get descriptor 文を使用してホスト変数にデータをコピーしてください。

動的 SQL のメソッド 4 は、次の手順に従って実行されます。

1　実行するための文を準備します。

2　カーソルと文を対応付けます。

3　入力パラメータまたは記述子を定義およびバインドします。

- 入力記述子を使っているときに、その記述子に割り付けます。
- 入力ホスト変数を使っているときに、そのパラメータに文またはカーソルを対応付けます。

4　適当な入力パラメータまたは記述子を使用してカーソルをオープンします。

5　入力記述子と異なる場合は出力記述子を割り付け、出力記述子を文へバインドします。

6　fetch cursor と出力記述子を使用してデータを取得します。

7　「動的記述子」からホスト・プログラム変数へデータをコピーします。SQLDA を使用している場合、この手順は適用されません。データは手順 6 でコピーされます。

8　カーソルをクローズします。

9　動的記述子の割り付けを解除します。

10　文 (最終的には、ストアド・プロシージャ) を削除します。

動的記述子の文

SQL 文に記述子を対応させる文と、その SQL 文と対応するカーソルに記述子を対応させる文があります。次の表では、メソッド 4 の場合の動的 SQL 文を簡潔に説明します。

文	説明
allocate descriptor	SQL 記述子を割り付けるように Client-Library に通知する。
describe input	prepare 文の動的パラメータ・マーカについての情報を得る。
set descriptor	動的記述子のデータの挿入または更新を行う。
get descriptor	記述子に保管されたロー情報またはパラメータ情報をホスト変数に移動し、これによってアプリケーション・プログラムがその情報を使用できるようにする。
execute	準備された文を実行する。
open cursor	記述子をカーソルと対応させて、そのカーソルをオープンする。
describe output	準備された動的 SQL 文の select リスト・カラムについての情報を得る。
fetch cursor	動的に宣言されたカーソルに対してデータのローを取得する。
deallocate descriptor	動的記述子の割り付けを解除する。

上記の文の詳細については、「第 9 章 Embedded SQL 文：リファレンス・ページ」を参照してください。

SQL 記述子の概要

SQL 記述子とは、prepare 文で準備された動的 SQL 文で使用される変数の記述を保管するメモリの領域です。SQL 記述子には、データ属性についての次のような情報が含まれる場合があります。

- *precision* － 整数。
- *scale* － 整数。
- *nullable* － カラムが null を含むことができる場合は 1 (*cs_true*)、含むことができない場合は 0 (*cs_false*)。get descriptor 文でのみ有効。
- *indicator* － 動的パラメータ・マーカに対応したインジケータ・パラメータの値。
- *name* － 動的パラメータ・マーカの名前。get descriptor 文でのみ有効。

- *data* − 項目番号で指定された動的パラメータ・マーカの値。*indicator* の値が -1 の場合、*data* の値は未定義である。
- *count* − 記述子で記述された動的パラメータ・マーカの数。
- *type* − 動的パラメータ・マーカまたはホスト変数のデータ型。
- *returned_length* − 出力カラム内のデータの実際の長さ。

「第 9 章 Embedded SQL 文：リファレンス・ページ」の set descriptor および get descriptor コマンドの説明を参照してください。

メソッド 4：例

次の例は、動的パラメータ・マーカおよび SQL 記述子とともに prepare および fetch を使用する例です。

```
exec sql begin declare section end-exec.
     01 COLTYPE      IS GLOBAL PIC S9(9) COMP.
      01 INDEX-COLCNT IS GLOBAL PIC S9(9) COMP.
      01 INT-BUFF     IS GLOBAL PIC S9(9) COMP.
      01 CHAR-BUFF    IS GLOBAL PIC X(255).
      01 MISC-BUFF    IS GLOBAL PIC X(255).
      01 TYPE         IS GLOBAL PIC X(255).
      01 TITLE        IS GLOBAL PIC X(255).
      01 COLNAME      IS GLOBAL PIC X(255).
      01 SALES        IS GLOBAL PIC S9(9) COMP.
      01 DESCNT       IS GLOBAL PIC S9(9) COMP.
      01 OCCUR        IS GLOBAL PIC S9(9) COMP.
      01 CNT          IS GLOBAL PIC S9(9) COMP.
      01 CONDCNT      IS GLOBAL PIC S9(9) COMP.
      01 DIAG-CNT     IS GLOBAL PIC S9(9) COMP.
      01 NUM-MSGS     IS GLOBAL PIC S9(9) COMP.
      01 USER-ID      IS GLOBAL PIC X(30).
      01 PASS         IS GLOBAL PIC X(30).
      01 SERVER-NAME  IS GLOBAL PIC X(30).
      01 STR1         IS GLOBAL PIC X(1024).
      01 STR2         IS GLOBAL PIC X(1024).
      01 STR3         IS GLOBAL PIC X(1024).
      01 STR4         IS GLOBAL PIC X(1024).
 exec sql end declare section end-exec.
...
PROCEDURE DIVISION.
  P0.
DISPLAY "Dynamic sql Method 4".
 DISPLAY "Enter in a Select statement to retrieve
 any kind "
 DISPLAY "of information from the pubs database:".
 accept str4.
 DISPLAY "Enter in the larger of the columns to be "
 DISPLAY "retrieved or the number "
```

```
 DISPLAY "of ? in the SQL statement:".
 ACCEPT occur.
exec sql prepare S4 from :str4 end-exec
exec sql declare c2 cursor for s4 end-exec
exec sql describe input s4 using sql descriptor dinout end-exec
     call "filldesc".
exec sql open c2 using sql descriptor dinout
 end-exec
PERFORM UNTIL SQLCODE = 100 OR SQLCODE < 0
exec sql fetch c2 into sql descriptor dinout end-exec
PERFORM "prtdesc".
 END-PERFORM.
exec sql close c2 end-exec
exec sql deallocate descriptor dinout end-exec
exec sql deallocate prepare s4 end-exec
DISPLAY "Dynamic SQL Method 4 completed".
     goback.
END PROGRAM dyn-m4.
IDENTIFICATION DIVISION.
 PROGRAM-ID. prtdesc is common.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. xyz.
 OBJECT-COMPUTER. xyz.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
P0.
exec sql get descriptor dinout :descnt = count
 end-exec
DISPLAY "Column name Column data".
DISPLAY "----------- --------------------------"
DISPLAY "---------------".
PERFORM VARYING CNT FROM 1 BY 1 UNTIL cnt > descnt
* get each column attribute
 exec sql get descriptor dinout
    VALUE :index-colcnt :coltype = TYPE end-exec
         IF coltype = 1
* character type
exec sql get descriptor dinout VALUE :index-colcnt
    :colname = NAME, :char-buff = data end-exec
DISPLAY colname char-buff.
ELSE IF coltype = 4
 * integer type
exec sql get descriptor dinout
 VALUE :index-colcnt :colname = NAME, :int-buff = DATA end-exec
DISPLAY colname int-buff.
     else
* other types
 exec sql get descriptor dinout
 VALUE :index-colcnt
```

```
 :colname = NAME, :misc-buff = DATA end-exec
DISPLAY colname misc-buff
    end-perform.
     goback.
 END PROGRAM prtdesc.
...
PROCEDURE DIVISION.
 P0.
exec sql get descriptor dinout :descnt = count
 end-exec
     PERFORM varying cnt from 1 by 1 UNTIL cnt >
     descnt
     DISPLAY "Enter in the data type of the " cnt "
     ?".
     accept &coltype.
     IF coltype = 1
 * character type
          DISPLAY "Enter in the value of the data:".
        ACCEPT char-buff.
         exec sql set descriptor dinout
                VALUE :cnt TYPE = 1,
                LENGTH = 255, DATA = :char-buff
  end-exec
       ELSE IF coltype = 4
  * integer type
          DISPLAY "Enter in the value of the data:".
          ACCEPT int-buff.
          exec sql set descriptor dinout
                VALUE :cnt TYPE = :coltype,
                DATA = :int-buff END-EXEC
         ELSE
         DISPLAY "non-supported column type.".
         END-IF.
         END-PERFORM
     GOBACK
    END PROGRAM filldesc.
 ...
```

SQLDA について

SQLDA は、SQL 記述子のように動的 SQL 文の準備文で使用される変数を記述するホスト言語の構造体です。ただし SQL 記述子とは異なり、SQLDA はユーザがアクセスできるフィールドを持つパブリック・データ構造体です。SQLDA を使用する文は、SQL 記述子を使用する同等の文よりも実行が速い場合があります。

SQLDA 構造体は SQL 標準の一部ではありません。Embedded SQL の実装方法が異なると、SQLDA 構造体の定義も異なります。Embedded SQL バージョン 11.1 以降では、Sybase が定義する SQLDA をサポートしていますが、他のベンダが定義する SQLDA データ型はサポートしていません。

Embedded SQL では、1 つのプログラムが作成できる SQLDA 構造体の数を制限しません。

表 7-1 は、SQLDA 構造体のフィールドの説明です。

表 *7-1: SQLDA* 構造体のフィールド

フィールド	データ型	説明
SD-SQLN	PIC S9(9) COMP	sd_column 配列のサイズ。
SD-SQLD	PIC S9(9) COMP	記述されるクエリのカラム数。つまり記述される文がクエリでない場合は、0 である。fetch 文、open 文、execute 文の場合、このフィールドは sd_column で記述されるホスト変数の数を示す。また describe input 文の場合、このフィールドは動的パラメータ・マーカの数を示す。
SD-DATAFMT OF SD-COLUMN	データ・フォーマット構造体	このカラムと対応する Client-Library の CS_DATAFMT 構造体。『Open Client Client-Library/C リファレンス・マニュアル』の ct_bind、ct_param、ct_describe の説明を参照。
SD-SQLDATA OF SD-COLUMN	PIC S9(9) COMP または PIC S9(18) COMP	fetch 文、open 文、execute 文の場合は、その文のホスト変数のアドレスを保管する。このフィールドは、describe 文または prepare 文では使用されない。
SD-SQLIND OF SD-COLUMN	PIC S9(4) COMP	fetch 文、open 文、execute 文の場合、このフィールドは記述されるカラムのインジケータ変数として機能する。そのカラムの値が null である場合、このフィールドは -1 に設定される。このフィールドは、describe 文または prepare 文では使用されない。SYBSETSQLDA を使用してこのフィールドを設定する (「SYBSETSQLDA の使用」(81 ページ) を参照)。
SD-SQLLEN OF SD-COLUMN	PIC S9(9) COMP	このカラムと対応する Client-Library CS_DATAFMT 構造体の実際の大きさ。
SD-SQLMORE OF SD-COLUMN	PIC S9(9) COMP または PIC S9(18) COMP	Sybase によって予約されている。

SYBSETSQLDA の使用

SQLDA フィールドの定義は COBOL 宣言と厳密には対応しないため、COBOL に慣れたユーザがその用語を使用できるように、SYBSETSQLDA 関数が用意されています。SYBSETSQLDA を使用すると、Sybase スタイルの SQLDA のフィールドを設定できます。これは、指定の SQLDA の ITEM-NUMBER SQLDA-SQLDATA フィールドを設定し、特定のバッファを指します。さらに、datafmt フィールドを適切に設定します。

構文

```
01 SQLDA-NAME.
< rest of sqlda declaration >
01 ITEM-NUMBER PIC S9(9) COMP.
 01 DATA-BUFFER < picture >.
 01 PICTURE-TYPE PIC S9(9) COMP.
 01 M PIC S9(9) COMP.
 01 N PIC S9(9) COMP.
 01 USAGE-TYPE PIC S9(9) COMP.
 01 SIGN-TYPE PIC S9(9) COMP
 CALL "SYBSETSQLDA" USING SQLDA-NAME ITEM-NUMBER
 DATA-BUFFER PICTURE-TYPE M N USAGE-TYPE SIGN-TYPE
```

各パラメータの意味は、次のとおりです。

- SQLDA-NAME は情報を設定するための SQLDA です。
- ITEM-NUMBER は、情報を設定する対象の項目です。
- DATA-BUFFER は、データを持つホスト変数です。
- PICTURE-TYPE は、データが持つ picture 句の種類です。使用可能な値については、表 7-2 を参照してください。
- M は、表に説明されている picture 句の “m” の値です。picture がない場合は 0 です。
- N は、前述の picture 句の “n” の値です。picture がない場合は 0 です。
- SIGN-TYPE は、データの定義に使用される sign 句です。使用可能な値については、表 7-2 を参照してください。
- USAGE-TYPE は、データの定義に使用される usage 句です。使用可能な値については、表 7-2 を参照してください。

表 *7-2: SYBSETSQLDA* の値

引数	値	意味
USAGE-TYPE	SYB-BINARY-USAGE	USAGE は BINARY
USAGE-TYPE	SYB-COMP-USAGE	USAGE は COMP
USAGE-TYPE	SYB-COMP1-USAGE	USAGE は COMP-1
USAGE-TYPE	SYB-COMP2-USAGE	USAGE は COMP-2
USAGE-TYPE	SYB-COMP3-USAGE	USAGE は COMP-3
USAGE-TYPE	SYB-COMP4-USAGE	USAGE は COMP-4
USAGE-TYPE	SYB-COMP5-USAGE	USAGE は COMP-5
USAGE-TYPE	SYB-COMP6-USAGE	USAGE は COMP-6
USAGE-TYPE	SYB-COMPX-USAGE	USAGE は COMP-X
USAGE-TYPE	SYB-DISPLAY-USAGE	USAGE は DISPLAY
USAGE-TYPE	SYB-POINTER-USAGE	USAGE は POINTER
USAGE-TYPE	SYB-INDEX-USAGE	USAGE は INDEX
USAGE-TYPE	SYB-MONEY-USAGE	USAGE は CS-MONEY
USAGE-TYPE	SYB-MONEY4-USAGE	USAGE は CS-MONEY4
USAGE-TYPE	SYB-DATE-USAGE	USAGE は CS-DATE
USAGE-TYPE	SYB-TIME-USAGE	USAGE は CS-TIME
USAGE-TYPE	SYB-DATETIME-USAGE	USAGE は CS-DATETIME
USAGE-TYPE	SYB-DATETIME4-USAGE	USAGE は CS-DATETIME4
USAGE-TYPE	SYB-NO-USAGE	usage 句なし
PICTURE-TYPE	SYB-NO-PIC	picture 句なし
PICTURE-TYPE	SYB-SNINES-PIC	PIC S9(m)
PICTURE-TYPE	SYB-NINES-PIC	PIC 9(m)
PICTURE-TYPE	SYB-SVNINES-PIC	PIC S9(m)V9(n) または SV9(n)
PICTURE-TYPE	SYB-VNINES-PIC	PIC 9(m)V9(n) または V9(n)
PICTURE-TYPE	SYB-X-PIC	PIC X(m)
SIGN-TYPE	SYB-NO-SIGN	sign 句なし (符号のない PIC 句ではない)
SIGN-TYPE	SYB-LEADING-SEPARATE-SIGN	SIGN LEADING SEPARATE
SIGN-TYPE	SYB-TRAILING-SEPARATE-SIGN	SIGN TRAILING SEPARATE
SIGN-TYPE	SYB-LEADING-SIGN	SIGN LEADING
SIGN-TYPE	SYB-TRAILING-SIGN	SIGN TRAILING

戻り値

戻り値はありません。

メソッド 4：SQLDA の使用例

次は、動的パラメータ・マーカおよび SQL 記述子とともに prepare および fetch を使用する例です。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  unittest.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  xyz.
OBJECT-COMPUTER.  xyz.
DATA DIVISION.
WORKING-STORAGE SECTION.
exec sql begin declare section end-exec
 01 uid pic x(10).
 01 pass pic x(10).
exec sql end declare section end-exec
 01 input-descriptor.
         09 SD-SQLN PIC S9(4) COMP.
         09 SD-SQLD PIC S9(4) COMP.
         09 SD-COLUMN OCCURS 3 TIMES.
          19 SD-DATAFMT.
                29 SQL--NM PIC X(132).
                29 SQL--NMLEN PIC S9(9) COMP.
              29 SQL--DATATYPE PIC S9(9) COMP.
              29 SQL--FORMAT PIC S9(9) COMP.
                  29 SQL--MAXLENGTH PIC S9(9) COMP.
                  29 SQL--SCALE PIC S9(9) COMP.
                         29 SQL--PRECISION PIC S9(9) COMP.
                  29 SQL--STTUS PIC S9(9) COMP.
              29 SQL--COUNT PIC S9(9) COMP.
                       29 SQL--USERTYPE PIC S9(9) COMP.
                  29 SQL--LOCALE PIC S9(9) COMP.
         19 SD-SQLDATA PIC S9(9) COMP.
         19 SD-SQLIND PIC S9(4) COMP.
         19 SD-SQLLEN PIC S9(9) COMP.
         19 SD-SQLMORE PIC S9(9) COMP.
 01 output-descriptor.
         09 SD-SQLN PIC S9(4) COMP.
         09 SD-SQLD PIC S9(4) COMP.
         09 SD-COLUMN OCCURS 3 TIMES.
          19 SD-DATAFMT.
              29 SQL--NM PIC X(132).
              29 SQL--NMLEN PIC S9(9) COMP.
              29 SQL--DATATYPE PIC S9(9) COMP.
              29 SQL--FORMAT PIC S9(9) COMP.
                  29 SQL--MAXLENGTH PIC S9(9) COMP.
                  29 SQL--SCALE PIC S9(9) COMP.
                         29 SQL--PRECISION PIC S9(9) COMP.
                  29 SQL--STTUS PIC S9(9) COMP.
```

```
                    29 SQL--COUNT PIC S9(9) COMP.
                              29 SQL--USERTYPE PIC S9(9) COMP.
                           29 SQL--LOCALE PIC S9(9) COMP.
                19 SD-SQLDATA PIC S9(9) COMP.
                19 SD-SQLIND PIC S9(4) COMP.
                19 SD-SQLLEN PIC S9(9) COMP.
                19 SD-SQLMORE PIC S9(9) COMP.
    01 conversion-tester pic s9(4) comp-3.
    01 charvar pic x(20).
    01 temp-int-1 pic s9(9) comp.
    01 temp-int-2 pic s9(9) comp.
    01 temp-int-3 pic s9(9) comp.
    01 temp-int-4 pic s9(9) comp.
    01 SQLCODE pic s9(9) comp.
    01 retcode pic s9(9) comp.
PROCEDURE DIVISION.
P0.
     MOVE "sa" TO uid.
     move" "to pass.
     exec sql connect :uid identified by :pass end-exec.
* setup
     exec sql whenever sqlwarning perform err-paraend-exec.
      exec sql drop table example end-exec.
      exec sql create table example (fruit char(30),
            number int)end-exec.
      exec sql insert example values ('tangerine', 1) end-exec.
      exec sql insert example values ('pomegranate', 2) end-exec.
      exec sql insert example values ('banana', 3) end-exec.
* test functionality using execute
      exec sql prepare statement from
          "select fruit from example where number = ?" end-exec.
      exec sql describe input statement using descriptor
           input-descriptor end-exec.
      if sd-sqld of input-descriptor not equal 1
             or  sql--datatype of sd-datafmt of sd-column  of
             input-descriptor (1) not equal cs-int-type
            display "failed on first describe input"
            move cs-fail to p-retcode
      end-if.
      move 1 to temp-int-1.
      move 4 to temp-int-2.
      move 0 to temp-int-3.
      call "SYBSETSQLDA" using retcode input-descriptor
             temp-int-1 conversion-tester syb-snines-pic
             temp-int-2 temp-int-3 syb-comp3-usage syb-no-sign .
      move 2 to conversion-tester.
      exec sql describe output statement using descriptor
      output-descriptor end-exec.
```

```
if sd-sqld of output-descriptor not equal
       or  sql--datatype of sd-datafmt of sd-column  of
       output-descriptor (1) not equal cs-char-type
       display "failed on first describe output"
      move cs-fail to p-retcode
end-if.
move 1 to temp-int-1.
move 20 to temp-int-2.
move 0 to temp-int-3.
call "SYBSETSQLDA" using retcode output-descriptor
      temp-int-1 charvar syb-x-pic temp-int-2
      temp-int-3 syb-no-usage syb-no-sign .
exec sql execute statement into descriptor
      output-descriptor using descriptor
      input-descriptor end-exec.
display "Expected pomegranate, got "charvar.
exec sql deallocate prepare statement end-exec.
exec sql prepare statement from
      "select number from example where fruit = ?" end-exec.
exec sql declare c cursor for statement end-exec.
exec sql describe input statement using descriptor
      input-descriptor end-exec.
move 1 to temp-int-1.
move 20 to temp-int-2.
move 0 to temp-int-3.
call "SYBSETSQLDA" using retcode input-descriptor
      temp-int-1 charvar syb-x-pic temp-int-2
      temp-int-3 syb-no-usage syb-no-sign .
move "banana" to charvar.
exec sql open c using descriptor input-descriptor end-exec.
exec sql describe output statement using descripto
      output-descriptor end-exec.
move 1 to temp-int-1.
move 20 to temp-int-2.
move 0 to temp-int-3.
call "SYBSETSQLDA" using retcode output-descriptor
     temp-int-1 charvar syb-x-pic temp-int-2 temp-int-3
     syb-no-usage syb-no-sign .
exec sql fetch c into descriptor output-descriptor
      end-exec.
display "Expected 3, got "charvar.
exec sql commit work end-exec.
end program unittest.
```

第 8 章

エラーの処理

この章では、Embedded SQL プログラムの実行中に発生するエラーを検出し、修正する方法について説明します。また、警告およびエラーの処理に使用する whenever および get diagnostics 文についてや、警告およびエラーと関係する SQLCA 変数についても説明します。

Embedded SQL アプリケーションの稼働中、アプリケーションの操作を妨げるイベントが起こることもあります。次にそのようなイベントの例を示します。

- Adaptive Server Enterprise がアクセスできなくなる
- ユーザが間違ったパスワードを入力する
- ユーザがデータベース・オブジェクトへのアクセス権を持っていない
- データベース・オブジェクトが削除されている
- カラムのデータ型が変更されている
- クエリが予期しない null 値を返す
- 動的 SQL 文に構文エラーがある

ユーザが、警告およびエラー処理コードを記述しておくと、これらのイベントを前もって予測でき、上記の状況をうまく復旧できます。

エラーのテスト

Adaptive Server Enterprise に送られた SQL 文の実行の成否を示すために、Embedded SQL は、*SQLCODE* 変数にリターン・コードを返します。Embedded SQL 文のあとで *SQLCODE* の値をテストするか、whenever 文を使用してユーザ用のテスト・コードを生成するようにプリコンパイラに指示できます。whenever 文については、この章で後述します。

SQLCODE の使用

表 8-1 に、*SQLCODE* に入る可能性のある値を示します。

表 *8-1: SQLCODE* の戻り値

値	意味
0	警告またはエラーの発生なし。
<0	エラーが発生した。SQLCA 変数には、エラーを診断するための情報が入る。
100	文は正常に実行されたが、最後の文からローが返されない。これは、カーソルからローをフェッチするループを使用しているときに役立つ。SQLCODE が 100 になると、ループおよびフェッチされたすべてのローが終了する。この手法については、「第 6 章 Transact-SQL 文の使い方」を参照。

警告状態のテスト

SQLCODE が文の実行が成功したことを示していても、警告状態が発生する場合があります。8 つの文字配列である SQLCA.SQLWARN は、このような警告状態を示します。各 SQLWARN 配列要素または “ フラグ ” には、スペース文字 (ブランク) または文字 “W” が格納されます。各フラグ内の “W” は、警告状態が発生したことを示します。警告状態の種類はフラグによって異なります。

表 8-2 は、各フラグでのスペース文字または文字 “W” の意味を示します。

表 *8-2: SQLWARN* フラグ

フラグ	説明
SQLWARN1	ブランクの場合、いかなるときでも警告状態ではなく、他のすべての SQLWARN フラグもブランクになる。SQLWARN1 に “W” が設定されている場合、1 つ以上の警告状態が発生し、他に少なくとも 1 つのフラグに “W” が設定されている。
SQLWARN2	“W” が設定されている場合、fetch 文で指定した文字列変数が文の結果データを格納するには短すぎ、そのため結果データがトランケートされた。トランケートされたデータの元の長さを受信するインジケータ変数を指定していない。
SQLWARN3	“W” が設定されている場合、Adaptive Server Enterprise に送信した入力が、式や null 値を使用できないテーブルへの入力値など、無効なコンテキスト内に null 値を含んでいた。
SQLWARN4	select 文の結果セット内のカラム数が、文の into 句内のホスト変数の数を超えていた。
SQLWARN5	Sybase によって予約されている。
SQLWARN6	この文を実行しようとしたとき、Adaptive Server が変換エラーを生成した。
SQLWARN7	Sybase によって予約されている。
SQLWARN8	Sybase によって予約されている。

警告のテストは、SQL 文の実行が成功したことを確認してから行ってください。ユーザ用のテスト・コードを生成するようにプリコンパイラに指示するには、whenever 文を使用してください。これについては、次の項で説明します。

whenever 文を使用したエラーのトラップ

Embedded SQL の whenever 文を使用して、エラーおよび警告状態をトラップします。この文には、Adaptive Server Enterprise に送られた Embedded SQL 文の結果に対応した動作を指定します。

whenever 文は、実行文ではありません。代わりに、プリコンパイラに COBOL コードを生成するように指示して、プログラムの各 Embedded SQL 実行文のあとで指定した条件をテストします。

whenever 文の構文は、次のとおりです。

```
exec sql whenever {sqlwarning | sqlerror |
 not found ]
 {continue | goto label |
 program call [using param . . .]) |
 perform paragraph_1 [through paragraph_2] |
 stop};
```

whenever の条件のテスト

whenever 文では、次のような 3 つの条件をテストできます。

- sqlwarning
- sqlerror
- not found

各条件に対する whenever 文を記述していない場合、プリコンパイラは警告メッセージを作成します。ユーザが、自分でコードを記述してエラーおよび警告をチェックする場合は、各条件に対応する whenever...continue 句を記述してプリコンパイラの警告を出さないようにします。こうすると、プリコンパイラはエラーおよび警告を無視します。

verbose オプションを使用してプリコンパイルすると、プリコンパイラは、connect 文の一部として ct_debug() 関数呼び出しを生成します。これによって、Client-Library はアプリケーションが実行している画面に情報、警告、およびエラー・メッセージを表示します。whenever は、これらのメッセージを無効にしません。『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

Embedded SQL 文の実行後、SQLCODE および SQLWARN1 の値によって条件のいずれかが成立しているかどうかを判定します。表 8-3 に、whenever が条件の検出に使用する基準を示します。

表 *8-3: whenever* 文の基準

条件	基準
sqlwarning	SQLCODE = 0 および SQLWARN1 = W
sqlerror	SQLCODE < 0
not found	SQLCODE = 100

whenever 文の動作を変更するには、その条件について新しく whenever 文を記述します。whenever は、その同じ条件に対する whenever 文が次に現れるまで、それ以降の Embedded SQL 文のすべてに適用されます。

whenever 文は、アプリケーション・プログラムの論理を無視します。たとえば、ループの最後に whenever を置いた場合は、ループ中でその whenever 文より前の部分に対しては、何の影響も与えません。

whenever の動作

whenever 文は、次の 5 つの動作のうちの 1 つを指定します。

表 *8-4: whenever* の動作

動作	説明
continue	SQL 文が指定した条件を戻しても、特別の動作は実行しない。通常の処理を続ける。
goto	アプリケーション・プログラム中のエラー処理プロシージャへ分岐する。goto を記述するときは “goto” または “go to” のどちらでも使用できる。後ろには、有効なパラグラフ名を置く。パラグラフ名がプログラムで定義されていない場合、プリコンパイラはエラーを検出しないが、COBOL コンパイラが検出する。
call	別の COBOL プログラムを呼び出す。オプションで変数を渡すこともできる。
perform	SQL 文が指定した条件になったときに実行するパラグラフを、最低でも 1 つは命名する。COBOL perform 文のフォーマット 1、2、3、4 を perform 句で使用できる。パラグラフ名を使用する場合、パラグラフは whenever 条件が適用するセクションになくてはならない。
stop	SQL 文が指定条件を戻したとき、プログラムを終了する。

```
      . . .
exec SQL whenever sqlerror perform ERR-PARA
         thru ERR-PARA-END
end-exec
       . . .
exec SQL select au_lname from authors
        into :AU-LNAME
        where au_id = :AU-ID
        end-exec
       . . .
exec SQL update authors set au_lname = :AU-LNAME
        where au_id = :AU-ID
end-exec
       . . .
```

get diagnostics の使い方

get diagnostics 文は、Client-Library からエラー・メッセージ、警告メッセージ、および情報メッセージを取得します。この文は whenever 文と似ていますが、検出したエラーの詳細を取得できるよう拡張されているため、さらに強力なものとなっています。

whenever 文内でアプリケーションを指定して別のアプリケーションまたはパラグラフへジャンプしたり呼び出したりする場合は、プロシージャ・コード内で次のように get diagnostics 文を指定します。

```
err-handler.
 exec sql get diagnostics :num-msgs = number
 end-exec.
 perform varying condcnt from 0 by 1
 until condcnt greater or equal num-msgs
 exec sql get diagnostics exception :condcnt
     :sqlca = sqlca_info end-exec
     display "sqlcode is " sqlcode
     display "message text is " sqlerrmc
 end-perform.
```

警告およびエラーを処理するルーチンの書き方

Embedded SQL アプリケーションでエラーおよび警告を処理するのによい手法は、エラーや警告を処理するカスタム・プロシージャを記述し、whenever...perform 文を使用してそのプロシージャをインストールすることです。

次に、警告およびエラー処理ルーチンの例を示します。記述量を少なくするため、どちらのルーチンでも、通常は指定する必要のある特定の条件を省略しています。warn_para では SQLWARN1 のコードを省略し、err_para では Client-Library のエラーとオペレーティング・システムのエラーを処理するコードを省略しています。

```
* Declare the sqlca.  *
 exec sql include sqlca end-exec
 exec sql whenever sqlerror call "ERR-PARA"
     end-exec
 exec sql whenever sqlwarning call
     "WARN-PARA" end-exec
 exec sql whenever not found continue end-exec

 WARN-PARA.
 *    Displays error codes and numbers from the sqlca
 *    and exits with an ERREXIT status.

    DISPLAY "Warning code is " SQLCODE.
     DISPLAY "Warning message is " SQLERRMC.

    IF SQLWARN2 EQUAL "W"
         DISPLAY "Data has been truncated.".
     IF SQLWARN3 EQUAL "W"
         DISPLAY "A null value was eliminated from
-         "    the argument set of a function.".
     IF SQLWARN4 EQUAL "W"
         DISPLAY "An into clause had too many or too
-         "    few host variables.".
     IF SQLWARN5 EQUAL "W"
         DISPLAY "A dynamic update or delete was
-         "    lacking a where clause.".
     IF SQLWARN6 EQUAL "W"
         DISPLAY "A server conversion or truncation
-         "    error occurred.".
 WARN-PARA-END.
     EXIT.

 ERR-PARA.
 * Print the error code, the error message, and the
 * line number of the command that caused the
 * error.

    DISPLAY "Error code is " SQLCODE.
     DISPLAY "Error message is " SQLERRMC.
     STOP RUN.
```

プリコンパイラが検出するエラー

Embedded SQL プリコンパイラは、プリコンパイル時に Embedded SQL のエラーを検出します。プリコンパイラは、セミコロンが見つからなかったり、宣言されていないホスト変数が SQL 文で使われているなどの構文エラーを検出します。これらは、重大なエラーであるため、適切なエラー・メッセージが生成されます。

プリコンパイラで Transact-SQL 構文エラーをチェックすることもできます。Adaptive Server Enterprise は、適切なプリコンパイラ・コマンド・オプションが設定されている場合は、プリコンパイル時に Transact-SQL 文を解析します。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』のプリコンパイラのリファレンス・ページを参照してください。

プリコンパイラは、Embedded SQL 文のホスト変数を動的パラメータ・マーカ (“?”) に置き換えます。ホスト変数をパラメータ・マーカに置き換えると、構文エラーの原因になることがあります (ルールまたはトリガがそのパラメータを許可しない場合など)。

プリコンパイラは、次の例のようにテーブルを作成してそこからデータを選択する場合にはエラーを検出しません。エラーは、ホスト変数のデータ型が取得したカラムに合わないということです。プリコンパイラが文を解析する時点では、テーブルはまだ存在しないため、プリコンパイラはエラーを検出しません。

```
exec sql begin declare section end-exec
         01  VAR1           PIC S9(9) COMP.
         02  VAR2           PIC X(20).
         exec sql end declare section end-exec

         exec sql create table T1
           (col1 int, col2 varchar(20)) end-exec
         ...

         exec sql select * from T1 into
        :VAR2, :VAR1 end-exec.
```

エラーは実行時に検出され、レポートされることに注意してください。

第 9 章

Embedded SQL 文：リファレンス・ページ

この章は、Transact-SQL にはない Embedded SQL 文、または Transact-SQL の場合とは機能が異なる Embedded SQL 文のそれぞれのリファレンス・ページから構成されています。Embedded SQL で有効なその他のすべての Transact-SQL 文の詳細については、『ASE Transact-SQL ユーザーズ・ガイド』を参照してください。

print、raiserror、readtext、writetext 以外のすべての Transact-SQL 文を Embedded SQL で使用できます。ただし、いくつかの文の構文はこの章で説明されているように異なっています。

この章のリファレンス・ページはアルファベット順になっています。それぞれの文のリファレンス・ページの内容は次のとおりです。

- その文の機能についての簡単な説明
- その文の構文の説明
- その文のキーワードとオプションについての説明
- その文の適切な使用方法についてのコメント
- 関連する文がある場合にはそのリスト
- その文の簡単な使用例

allocate descriptor

説明　SQL 記述子を割り付けます。

構文

exec sql allocate descriptor *descriptor_name*
 [with max [*host_variable* | *integer_literal*]]
 end-exec

パラメータ

descriptor_name
準備文内の動的パラメータ・マーカについての情報が入っている SQL 記述子の名前です。

with max
SQL 記述子内のカラムの最大数です。

host_variable
declare セクション内で定義された整数ホスト変数です。

integer_literal
SQL 記述子をいくつ使用するかを表す数値です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
                01    COLTYPE           PIC S9(9) COMP.
                01    NUMCOLS           PIC S9(9) COMP.
                01    COLNUM            PIC S9(9) COMP.
    EXEC SQL END DECLARE SECTION END-EXEC.

               ...

    EXEC SQL ALLOCATE DESCRIPTOR big_desc WITH MAX 1000 END-EXEC.

    EXEC SQL PREPARE dynstmt FROM "select * from huge_table" END-EXEC.

   *  Assume that the select returns only 1 row.
    EXEC SQL EXECUTE dynstmt INTO SQL DESCRIPTOR big_desc END-EXEC.

    EXEC SQL GET DESCRIPTOR big_desc :NUMCOLS = COUNT END-EXEC.

    MOVE 1 TO COLNUM.
    PERFORM GET-DESC-LOOP UNTIL COLNUM  > NUMCOLS.

    EXEC SQL DEALLOCATE DESCRIPTOR big_desc END-EXEC.
    EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
        ...

    GET-DESC-LOOP.
        EXEC SQL GET DESCRIPTOR big_desc VALUE
                :COLNUM :COLTYPE = TYPE END-EXEC.
        DISPLAY "COLUMN ",COLNUM," IS OF TYPE ", COLTYPE.
        ADD 1 TO COLNUM.
```

使用法

- allocate descriptor コマンドは、Adaptive Server Enterprise が割り付ける項目記述子領域の数を指定します。
- 割り付けできる SQL 記述子の数には制限がありません。
- SQL 記述子が割り付けられるとき、そのフィールドは未定義です。
- すでに割り付けられている SQL 記述子を割り付けようとすると、エラーが発生します。
- with max 句に値を指定していない場合は、1 つの項目記述子が割り当てられます。
- SQL 記述子が割り付けられるとき、そのフィールドのそれぞれの値は未定義です。

参照

deallocate descriptor、get descriptor、set descriptor

begin declare section

説明

Embedded SQL ソース・ファイルで使用されるホスト言語変数を宣言する declare セクションを開始します。

構文

```
exec sql begin declare section end-exec
 host_variable_declaration.
 ...
exec sql end declare section end-exec
```

パラメータ

host_variable_declaration

1 つ以上のホスト言語変数の宣言です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
                01 TITLE PIC X(80).
                01 VAR1PIC S9(9) COMP.
                01 VAR2 PIC X(100).
 EXEC SQL END DECLARE SECTION END-EXEC.
```

使用法

- declare セクションの終了には、Embedded SQL の end declare section 文を使用します。
- ソース・ファイルに指定できる declare セクションの数には制限がありません。
- 変数を宣言できる場所であればどこにでも declare セクションを指定できます。変数を宣言する declare セクションは、その変数を参照する文よりも前に指定してください。
- declare セクションでの変数の宣言は、ホスト言語の規則に準拠する必要があります。

- declare セクションではネストされた構造体を使用できます。ただし、構造体の配列は使用できません。
- declare セクションで指定できる Embedded SQL の include 文の数には制限がありません。
- declare セクションにある Embedded SQL の include 文を処理する場合、Embedded SQL プリコンパイラは、プリコンパイルされるファイルに直接入力されたようにインクルード・ファイルの内容を処理します。

参照　exec sql include "filename"

begin transaction

説明　非連鎖トランザクションの始めにマークを付けます。

構文

```
exec sql [at connection_name]
 begin {transaction | tran} [transaction_name]
 end-exec
```

パラメータ

transaction | tran
　キーワードは transaction と tran のどちらでも使用できます。

transaction_name
　このトランザクションに割り当てられる名前です。この名前は、Transact-SQL 識別子の規則に準拠しなければなりません。

例

```
      *
       * Use explicit transactions to synchronize tables on
       * two servers.
       *

          EXEC SQL BEGIN DECLARE SECTION END-EXEC.
               01     TITLE-ID     PIC X(6).
               01     NUM-SOLD     PIX S9(9) COMP.
          EXEC SQL END DECLARE SECTION END-EXEC.

                ...

          EXEC SQL WHENEVER SQLERROR PERFORM ABORT-TRAN END-EXEC.

             EXEC SQL CONNECT :UID IDENTIFIED BY :PASS
                                  AT connect1  END-EXEC.
             EXEC SQL CONNECT :UID IDENTIFIED BY :PASS
                                  AT connect2 END-EXEC.

          PERFORM TRY-UPDATE.
```

```
TRY-UPDATE.
EXEC SQL AT connect1 BEGIN TRANSACTION END-EXEC.
EXEC SQL AT connect2 BEGIN TRANSACTION END-EXEC.

EXEC SQL AT connect1 SELECT  sum(qty) INTO :NUM-SOLD
      FROM salesdetail
      WHERE title_id = :TITLE-ID END-EXEC.

EXEC SQL AT connect2 UPDATE current_sales
      SET num_sold = :NUM-SOLD
      WHERE title_id = :TITLE-ID END-EXEC.

EXEC SQL AT connect2 COMMIT TRANSACTION END-EXEC.
EXEC SQL AT connect1 COMMIT TRANSACTION END-EXEC.

IF SQLCODE <> 0
     DISPLAY "OOPS!Should have used 2-phase commit".


ABORT-TRAN.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
   DISPLAY "Error code is " SQLCODE.
   DISPLAY "Error message is " SQLERRMC.
EXEC SQL AT connect2 ROLLBACK TRANSACTION END-EXEC.
EXEC SQL AT connect1 ROLLBACK TRANSACTION END-EXEC.
PERFORM TRY-UPDATE.
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の begin transaction 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- begin transaction 文は非連鎖トランザクション・モードでのみ有効です。連鎖トランザクション・モードでは、トランザクションの開始点に明示的にマークを付けることはできません。
- トランザクションをネストする場合、最も外側の begin transaction 文とその文に対応する commit transaction 文または rollback transaction 文だけにトランザクション名を割り当てます。
- データベース・オプション ddl in tran を設定しないかぎり、Adaptive Server Enterprise では、非連鎖トランザクションの中で create database、create table、create index、create view、drop、select into *table_name*、grant、revoke、alter database、alter table、truncate table、update statistics、load database、load transaction、および disk init の各文を使用することはできません。
- 1 つのトランザクションには、そのトランザクションの開始時に使用している、現在の接続で実行される文だけが含まれています。
- リモート・プロシージャは、それらのプロシージャが含まれているトランザクションとは関係なく実行されます。

参照

commit transaction、commit work、rollback transaction、rollback work

close

説明　オープンしているカーソルをクローズします。

構文　exec sql [at *connection_name*] close *cursor_name*
end-exec

パラメータ　*cursor_name*
クローズするカーソルの名前です。この名前はカーソルを宣言したときに割り当てた名前です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     LNAME     PIC X(40).
      01     FNAME   PIC X(20).
      01     PHONE   PIC X(12).
 EXEC SQL END DECLARE SECTION END-EXEC.

           ...

 EXEC SQL DECLARE authorlist CURSOR FOR
       SELECT au_lname, au_fname, phone
       FROM authors END-EXEC.

 EXEC SQL OPEN authorlist END-EXEC.
 PERFORM FETCH-LOOP UNTIL SQLCODE = 100.

 EXEC SQL CLOSE authorlist END-EXEC,
           ...


 FETCH-LOOP.
      EXEC SQL FETCH authorlist INTO
           :LNAME, :FNAME, :PHONE END-EXEC.
      DISPLAY LNAME, FNAME, PHONE.
```

使用法

- close 文はオープンしているカーソルをクローズします。フェッチされなかったローはキャンセルされます。
- クローズしたカーソルを再度オープンすると、対応するクエリが再実行されて、結果セットの最初のローの前にカーソル・ポインタが置かれます。
- カーソルは再度オープンする前にクローズしなければなりません。
- オープンしていないカーソルをクローズしようとすると、実行時エラーが発生します。
- commit transaction 文、rollback transaction 文、commit work 文、rollback work 文はカーソルを自動的にクローズしますが、プリコンパイラのオプションを使用してその動作を無効にすることもできます。

- カーソルを一度クローズしてから再度オープンすると、そのカーソルがローを取得するテーブルに対して行われたすべての変更を、プログラム側で認識できます。

参照 declare cursor、fetch、open、prepare

commit

説明 トランザクション中のデータベースへの変更を保存し、トランザクションを終了します。

構文
exec sql [at *connection_name*]
commit [transaction | tran | work]
[*transaction_name*] end-exec

パラメータ transaction | trans | work

rollback 文では、キーワード transaction、trans、work は交換可能ですが work だけは ANSI 準拠です。

transaction_name

トランザクションに割り当てられた名前です。

例

例 *1*

```
    * Using unchained transaction mode to
         * synchronize tables on two servers.
         *
          EXEC SQL BEGIN DECLARE SECTION END-EXEC.
               01     TITLE-ID     PIC X(7).
               01     NUM-SOLD       PIC S9(9).
          EXEC SQL END DECLARE SECTION END-EXEC.

                ...

             EXEC SQL CONNECT :UID IDENTIFIED BY :PASS
                                  AT connect1  END-EXEC.
             EXEC SQL CONNECT :UID IDENTIFIED BY :PASS
                                  AT connect2 END-EXEC.

                ...

          PERFORM TRY-UPDATE.

          TRY-UPDATE.
          EXEC SQL AT connect1 BEGIN TRANSACTION END-EXEC.
          EXEC SQL AT connect2 BEGIN TRANSACTION END-EXEC.
```

```
EXEC SQL AT connect1 SELECT sum(qty) INTO :NUM-SOLD
      FROM salesdetail
      WHERE title_id = :TITLE-ID END-EXEC.

EXEC SQL AT connect2 UPDATE current_sales
      SET num_sold = :NUM-SOLD
      WHERE title_id = :TITLE-ID END-EXEC.

EXEC SQL AT connect2 COMMIT TRANSACTION END-EXEC.

EXEC SQL AT connect1 COMMIT TRANSACTION END-EXEC.

IF SQLCODE <> 0
     DISPLAY "Oops!Should have used 2-phase commit".
```

例 *2*

```
* Using chained transaction mode to synchronize
* tables on two servers.


EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01     TITLE-ID     PIC X(7).
     01     NUM-SOLD     PIX S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.

      ...

EXEC SQL WHENEVER SQLERROR PERFORM ABORT-TRAN END-EXEC.

PERFORM TRY-UPDATE.

TRY-UPDATE.
EXEC SQL AT connect1 SELECT sum(qty) INTO :NUM-SOLD
      FROM salesdetail
      WHERE title_id = :TITLE-ID END-EXEC.

EXEC SQL AT connect2 UPDATE current_sales
      SET num_sold = :NUM-SOLD
      WHERE title_id = :TITLE-ID END-EXEC.

EXEC SQL AT connect2 COMMIT WORK END-EXEC.
EXEC SQL AT connect1 COMMIT WORK END-EXEC.

IF SQLCODE <> 0
     DISPLAY "OOPS!Should have used 2-phase commit".
```

```
ABORT-TRAN.
   DISPLAY "ERROR!ABORTING TRAN".
   DISPLAY "Error code is " SQLCODE.
   DISPLAY "Error message is " SQLERRMC.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
EXEC SQL AT connect2 ROLLBACK WORK END-EXEC.
EXEC SQL AT connect1 ROLLBACK WORK END-EXEC.
PERFORM TRY-UPDATE.
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の commit 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- トランザクション名は、識別子についての Transact-SQL の規則に準拠している必要があります。トランザクション名は Transact-SQL の拡張機能です。したがって、ANSI 標準に準拠するキーワード work とともに使用することはできません。
- トランザクションをネストする場合、最も外側の begin transaction 文とその文に対応する commit transaction 文または rollback transaction 文だけにトランザクション名を割り当てます。

参照

begin transaction、commit work、rollback transaction、rollback work

connect

説明

Adaptive Server Enterprise との接続を確立します。

構文

exec sql connect *user_name*
[identified by *password*] [at *connection_name*]
[using *server_name*] [labelname *label_name* labelvalue *label_value* …] end-exec

パラメータ

user_name

Adaptive Server Enterprise にログインするときに使用するユーザ名です。

password

Adaptive Server Enterprise にログインするときに使用するパスワードです。

connection_name

Adaptive Server Enterprise 接続をユニークに識別するのに使用する名前です。

server_name

接続している Adaptive Server Enterprise のサーバ名です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01         UID             PIC X(32).
      01         PASS            PIC X(32).
      01         SERVER          PIC X(100).
 EXEC SQL END DECLARE SECTION END-EXEC.


 DISPLAY "UID NAME?".
 ACCEPT UID.
 DISPLAY "PASSWORD ?".
 ACCEPT PASS.
 DISPLAY "SERVER TO CONNECT TO ?".
 ACCEPT SERVER.

 EXEC SQL CONNECT :UID IDENTIFIED BY :PASS
           USING :SERVER END-EXEC.
```

使用法

- すべての Embedded SQL プログラムで、connect 文は allocate descriptor 文以外のすべての実行 SQL 文よりも前に実行される必要があります。
- *label_name* 句と *label_value* 句を使用する場合、これらの句が connect 文の最後の句になるようにしてください。
- C 言語と COBOL 言語の両方を使用するプログラムでは、最初の connect 文は COBOL プログラムから発行しなければなりません。
- プログラムが複数の接続を持っている場合、名前を指定しないで使用できる接続は 1 つだけです。
- 特定の名前付き接続に指示する at *connection_name* 句が Embedded SQL 文に存在しないと、その文は現在の接続上で実行されます。
- null パスワードを指定するには、identified by 句を省略するか、空の文字列を使用してください。
- connect 文が Adaptive Server Enterprise を指定しないと、DSQUERY 環境変数または論理名によって指定されたサーバを使用します。DSQUERY が定義されていない場合、デフォルトのサーバは SYBASE になります。
- Client-Library は、SYBASE 環境変数または論理名によって指定されたディレクトリ内にある interfaces ファイル内のサーバ名を探します。
- Embedded SQL プログラムが終了するか、または disconnect 文を発行すると、Adaptive Server Enterprise 接続が終了します。
- 名前付きまたは名前なしに関わらず新しい接続をオープンすると、新しい接続が現在の接続になります。
- 複数の Adaptive Server Enterprise ログイン名が必要なプログラムは、ログイン・アカウントごとに接続を持つことができます。

- 2 つ以上のサーバに接続することによって、プログラムは、異なるサーバに格納されたデータに同時にアクセスできます。
- 1 つのプログラムは 1 台のサーバへの複数接続、または、異なるサーバへの複数接続を持つことができます。
- 表 9-1 は、接続名の付け方を示します。

表 *9-1:* 接続名の付け方

使用されている句	使用されていない句	接続名
at *connection_name*		*connection_name*
using *server_name*	at	*server_name*
なし		DEFAULT

参照 at connection_name、exec sql、disconnect、set connection

deallocate cursor

説明 静的 SQL 文または動的 SQL 文のカーソルの割り付けを解除します。

構文 exec sql [at *connection_name*] deallocate cursor *cursor_name* end-exec

パラメータ *cursor_name*

割り付けを解除するカーソルの名前です。*cursor_name* は、"*my_cursor*" または *my_cursor* のように、二重引用符で囲まれた文字列か引用符のない文字列にしてください。cursor_name にはホスト変数は指定できません。

例

```
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
                01     TITLE-ID     PIC X(7).
                01     BOOK-NAME      PIC X(80).
                01     TTYPE          PIC X(12).
                01     TITLE-INDIC      S9(9).
                01     TYPE-INDIC       S9(9).
           EXEC SQL END DECLARE SECTION END-EXEC.

                ...

           EXEC SQL DECLARE titlelist CURSOR FOR
                 SELECT type, title_id, title FROM titles
                     order by type END-EXEC.

           EXEC SQL OPEN titlelist END-EXEC.

           PERFORM FETCH-PARA UNTIL SQLCODE = 100.

           EXEC SQL CLOSE titlelist END-EXEC.
```

```
EXEC SQL DEALLOCATE CURSOR titlelist END-EXEC.
     ...


FETCH-PARA.
     EXEC SQL FETCH titlelist INTO
          :TTYPE      :TYPE-INDIC,
          :TITLE-ID,
          :BOOK-NAME     :TITLE-INDIC  END-EXEC.

     IF TYPE-INDIC <> -1
          DISPLAY "TYPE      :", TTYPE
     ELSE
          DISPLAY "TYPE      :UNDECIDED"
     END-IF.

     DISPLAY "TITLE ID :",TITLE-ID.

     IF TITLE-INDIC <> -1
          DISPLAY "TITLE     :", BOOK-NAME
     ELSE
          DISPLAY "TITLE     :Null value"
     END-IF.
END-FETCH-PARA.
```

使用法

- カーソルの割り付けを解除すると、そのカーソルに割り付けられたすべてのリソースが解放されます。特に deallocate cursor 文は、そのカーソルに対応する Client-Library コマンド・ハンドルと CS_COMMAND 構造体を削除します。
- 静的カーソルのオープン後はいつでもそのカーソルの割り付けを解除できます。また、動的カーソルの宣言後はいつでもそのカーソルの割り付けを解除できます。
- *cursor_name* がオープンしている場合、deallocate cursor 文はそのカーソルをクローズしてからその割り付けを解除します。
- 割り付けが解除されたカーソルは、参照したり再度オープンしたりすることはできません。このようなカーソルを参照または再度オープンしようとすると、エラーが発生します。
- 割り付けが解除されたカーソルと同じ名前を使用して新しいカーソルを宣言できます。割り付けが解除されたカーソルと同じ名前を使用してカーソルをオープンすることは、割り付けが解除されたカーソルを再度オープンすることと同じではありません。新しいカーソルは、割り付けが解除されたカーソルとは名前以外には何も共有しません。
- 割り付けが解除されたカーソルと同じ名前を使用して新しいカーソルを宣言すると、プリコンパイラが警告メッセージを表示する場合もあります。
- deallocate cursor 文は Sybase の拡張機能であり、SQL 標準では定義されていません。

注意 Embedded SQL プログラムで継続バインドを使用する場合は、注意して deallocate cursor 文を使用してください。カーソルの割り付けの解除を必要もないのに行うと、継続バインドの利点がなくなる可能性があります。

参照 close cursor、declare cursor、open (**静的カーソル**)

deallocate descriptor

説明 SQL 記述子の割り付けを解除します。

構文

```
exec sql deallocate descriptor descriptor_name
 end-exec
```

パラメータ

descriptor_name

準備文内の動的パラメータ・マーカまたは戻り値についての情報が入っている SQL 記述子の名前です。

例

```
        EXEC SQL BEGIN DECLARE SECTION END-EXEC.
              01     NUMCOLS        PIC S9(9) COMP.
              01     COLNUM         PIC S9(9) COMP.
              01     COLTYPE        PIC S9(9) COMP.
         EXEC SQL END DECLARE SECTION END-EXEC.

                 ...

         EXEC SQL ALLOCATE DESCRIPTOR big_desc WITH MAX 100 END-EXEC.
         EXEC SQL PREPARE dynstmt FROM "select * from huge_table" END-EXEC.

      * Assume that only one row of data is returned.
         EXEC SQL EXECUTE dynstmt INTO SQL DESCRIPTOR big_desc END-EXEC.
         EXEC SQL GET DESCRIPTOR big_desc :NUMCOLS = COUNT END-EXEC.

         MOVE 1 TO COLNUM.
         PERFORM GET-DESC-LOOP UNTIL COLNUM > NUMCOLS.

         EXEC SQL DEALLOCATE DESCRIPTOR big_desc END-EXEC.

                ...


         GET-DESC-LOOP.
         EXEC SQL GET DESCRIPTOR big_desc VALUE
                   :COLNUM :COLTYPE = TYPE END-EXEC.
         DISPLAY "COLUMN TYPE = ",COLTYPE.
         ADD 1 TO COLNUM.
```

使用法

- まだ割り付けられていない SQL 記述子の割り付けを解除しようとすると、エラーが発生します。

参照　allocate descriptor

deallocate prepare

説明　prepare 文を使用して準備された動的 SQL 文の割り付けを解除します。

構文

exec sql [at *connection_name*]
 deallocate prepare *statement_name* end-exec

パラメータ

statement_name
文が準備されたときに、動的 SQL 文に割り当てられる識別子です。

例

```
       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
             01     CMDBUF          PIC X(120).
             01     STATE           PIC X(3).
        EXEC SQL END DECLARE SECTION END-EXEC.

                ...

      * The 'select into table' statement returns no results
      * to the program, so it does not need a cursor.

         MOVE "select * into tmp from authors where state = ?"
                   TO CMDBUF.

         DISPLAY "STATE ? ".
         ACCEPT STATE.

         EXEC SQL PREPARE dynstmt FROM :CMDBUF END-EXEC.
         EXEC SQL EXECUTE dynstmt USING :STATE END-EXEC.

         EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
         EXEC SQL COMMIT WORK END-EXEC.
```

使用法

- prepare 文で文を指定してから、その文の割り付けを解除してください。prepare 文で指定されていない文の割り付けを解除しようとすると、エラーが発生します。
- *statement_name* は文バッファをユニークに識別する必要があり、変数の命名についての SQL 識別子の規則に準拠する必要があります。*statement_name* にはリテラルと文字配列ホスト変数のどちらかを指定できます。
- deallocate prepare 文は、*statement_name* に対して宣言されたすべての動的カーソルのクローズと割り付けの解除を行います。

警告！ Embedded SQL プログラムで継続バインドを使用する場合は、注意して deallocate prepare 文を使用してください。prepare 文で指定されていない文の割り付けの解除を必要もないのに行うと、継続バインドの利点がなくなる可能性があります。

参照 declare cursor (動的)、execute、execute immediate、prepare

declare cursor (動的)

説明 準備した動的 select 文によって返される複数のローを処理するためにカーソルを宣言します。

構文

```
exec sql [at connection_name]
 declare cursor_name
 cursor for prepped_statement_name end-exec
```

パラメータ

cursor_name

open 文、fetch 文、close 文の中のカーソルを参照するのに使用するカーソルの名前です。カーソルの名前は各接続上でユニークでなければなりません。また、その名前は 255 文字以下にしてください。

prepped_statement_name

実行する select 文を表す (前の prepare 文内で指定された) 名前です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC

       01      QUERY       PIC X(100).
       01      DATAVAL     PIC X(100).
       01      COUNTER     PIC S9(9) COMP.
       01      NUMCOLS     PIC S9(9) COMP.
       01      COLNAME     PIC X(32).
       01      COLTYPE     PIC S9(9) COMP.
       01      COLLEN      PIC S9(9) COMP.

  EXEC SQL END DECLARE SECTION END-EXEC.
                ...
  EXEC SQL WHENEVER SQLERROR PERFORM ERR-PARA END-EXEC.
  EXEC SQL WHENEVER SQLWARNING PERFORM WARN-PARA END-EXEC
  EXEC SQL WHENEVER NOT FOUND STOP END-EXEC.
                ...
EXEC SQL USE pubs2 END-EXEC.
MOVE "SELECT * FROM publishers " TO QUERY.
EXEC SQL ALLOCATE DESCRIPTOR dout WITH MAX 100 END-EXEC.
EXEC SQL PREPARE  dynstmt FROM :QUERY END-EXEC.
EXEC SQL DECLARE dyncur CURSOR FOR dynstmt END-EXEC.
EXEC SQL OPEN dyncur END-EXEC.
```

```
    PERFORM FETCH-LOOP UNTIL SQLCODE  = 100.

* Clean-up all open cursors, descriptors and dynamic statements.

    EXEC SQL CLOSE dyncur END-EXEC.
    EXEC SQL DEALLOCATE CURSOR dyncur END-EXEC.
    EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
    EXEC SQL DEALLOCATE DESCRIPTOR dout END-EXEC.
    EXEC SQL COMMIT WORK END-EXEC.

    STOP RUN.

     FETCH-LOOP.
           EXEC SQL FETCH dyncur INTO SQL DESCRIPTOR dout END-EXEC
           EXEC SQL GET DESCRIPTOR dout :NUMCOLS = COUNT END-EXEC
           DISPLAY "COLS = ", NUMCOLS
           MOVE 1 TO COUNTER
           PERFORM GET-DESC-PARA UNTIL  COUNTER > NUMCOLS.
     END-FETCH-LOOP.

      GET-DESC-PARA.
            EXEC SQL GET DESCRIPTOR dout VALUE :COUNTER
                            :COLNAME = NAME,
                            :COLTYPE = TYPE,
                            :COLLEN  = LENGTH
                         END-EXEC
            DISPLAY "NAME      :", COLNAME
            DISPLAY "TYPE      :", COLTYPE
            DISPLAY "LENGTH    :", COLLEN

            EXEC SQL GET DESCRIPTOR dout VALUE :COUNTER
                            :DATAVAL = DATA END-EXEC
            DISPLAY "DATA      :", DATAVAL
            DISPLAY " "
            ADD 1 TO COUNTER.
     END-GET-DESC-PARA.
```

使用法

- *prepped_statement_name* には、a compute 句を指定できません。
- *prepped_statement_name* が prepare 文で指定された接続で *cursor_name* を宣言してください。
- 静的 declare cursor 文が単に宣言であるのに対して、動的 declare cursor 文は実行文です。ホスト言語で実行文を使用できる場所に動的 declare 文を置いてください。また、そのホスト言語のプログラムは実行文のリターン・コード (SQLCODE、SQLCA、または SQLSTATE) をチェックする必要があります。
- 動的カーソルの for update 句と read only 句は declare cursor 文の一部ではありませんが、prepare 文で指定された文の select クエリ内に含まれている必要があります。

参照 close、connect、fetch、open、prepare

declare cursor (静的)

説明 select 文によって返される複数のローを処理するためにカーソルを宣言します。

構文

```
exec sql declare cursor_name
 cursor for select_statement
 [for update [of col_name_1 [, col_name_n]…]|
 for read only] end-exec
```

パラメータ

cursor_name

open 文、fetch 文、close 文の中のカーソルを参照するのに使用するカーソルの名前です。カーソルの名前は各接続上でユニークでなければなりません。また、その名前は 255 文字以下にしてください。

select_statement

カーソルのオープン時に実行される Transact-SQL の select 文です。『ASE リファレンス・マニュアル』の select 文の説明を参照してください。

for update

更新できるカーソルの結果リストを指定します (結果リストを更新するには update 文を使用します)。

of *col_name_n*

更新するカラムの名前です。

for read only

更新できないカーソルの結果リストを指定します。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     TITLE-ID      PIC X(6).
      01     BOOK-NAME         PIC X(25).
      01     TYPE           PIC X(15).
 EXEC SQL END DECLARE SECTION END-EXEC.

      01      ANSWER           PIC X(1).

      ....


 DISPLAY "TYPE OF BOOKS TO RETRIEVE ? ".
 ACCEPT BOOK-TYPE.
 EXEC SQL DECLARE titlelist CURSOR FOR
      SELECT title_id, substring(title,1,25) FROM
           titles WHERE type = :BOOK-TYPE END-EXEC.

 EXEC SQL OPEN titlelist END-EXEC.
 PERFORM FETCH-PARA UNTIL SQLCODE = 100.
```

```
EXEC SQL CLOSE titlelist END-EXEC.
EXEC SQL DEALLOCATE CURSOR titlelist END-EXEC.
EXEC SQL COMMIT WORK END-EXEC.


FETCH-PARA.
     EXEC SQL FETCH titlelist INTO
               :TITLE-ID, :BOOK-NAME END-EXEC.
     DISPLAY "TITLE ID : ",TITLE-ID
     DISPLAY "TITLE    : ",BOOK-NAME
     IF SQLCODE = 100
          DISPLAY "NO RECORDS TO FETCH. END OF PROGRAM RUN."
     ELSE
     DISPLAY "UPDATE/DELETE THIS RECORD (U/D)? "
     ACCEPT ANSWER.

     IF ANSWER = "U"
          DISPLAY "ENTER NEW TITLE :"
          ACCEPT BOOK-NAME
          EXEC SQL UPDATE titles SET title = :BOOK-NAME
                WHERE CURRENT OF titlelist END-EXEC
     ELSE
          IF ANSWER = "D"
               EXEC SQL DELETE titles
               WHERE CURRENT OF titlelist  END-EXEC
          END-IF
     END-IF
     END-IF.
END-FETCH-PARA.
```

使用法

- Embedded SQL プリコンパイラは、declare cursor 文に対してはコードを生成しません。
- open cursor 文を使用してプログラムがカーソルをオープンするまで、select_statement は実行されません。
- Embedded SQL では compute 句を使用できない点を除けば、select_statement の構文は『ASE リファレンス・マニュアル』に記載されている説明と同じです。
- select_statement にはホスト変数を指定できます。ホスト変数の値は、プログラムがカーソルをオープンするときに置き換えられます。
- for update 句と read only 句のどちらかを省略した場合、Adaptive Server Enterprise はそのカーソルが更新可能かどうかを判断します。

参照

close、connect、deallocate cursor、declare cursor (ストアド・プロシージャ)、declare cursor (動的)、fetch、open、update

declare cursor (ストアド・プロシージャ)

説明

ストアド・プロシージャに対してカーソルを宣言します。

構文

exec sql declare *cursor_name*
cursor for execute *procedure_name*
([[@*param_name* =]:*host_var*]
[,[@*param_name* =]:*host_var*]…) end-exec

パラメータ

cursor_name

open 文、fetch 文、close 文の中のカーソルを参照するのに使用するカーソルの名前です。カーソルの名前は各接続上でユニークでなければなりません。また、その名前は 255 文字以下にしてください。

procedure_name

実行するストアド・プロシージャの名前です。

param_name

ストアド・プロシージャ内のパラメータの名前です。

host_var

パラメータ値として渡されるホスト変数の名前です。

例

```
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01        TITLE-ID      PIC X(6).
       01        BOOK-NAME     PIC X(65).
       01        BOOK-TYPE     PIC X(15).
  EXEC SQL END DECLARE SECTION END-EXEC.

       01        ANSWER        PIC X(1).

       ....

* Create the stored procedure.

  EXEC SQL create procedure p_titles (@p_type varchar(30))
        as
            select title_id, substring(title,1,64)
                  from titles
                 where type = @p_type
  END-EXEC.
```

```
* To execute stored procedures, you must disable chained mode.
 EXEC SQL SET CHAINED OFF END-EXEC.

 DISPLAY "TYPE OF BOOKS TO RETRIEVE ? ".
 ACCEPT BOOK-TYPE.
 EXEC SQL DECLARE titlelist CURSOR FOR
            execute p_titles :BOOK-TYPE END-EXEC.
 EXEC SQL OPEN titlelist END-EXEC.
 PERFORM FETCH-LOOP UNTIL SQLCODE = 100.
 EXEC SQL CLOSE titlelist END-EXEC.
 EXEC SQL DEALLOCATE CURSOR titlelist END-EXEC.
 EXEC SQL COMMIT WORK END-EXEC.

 FETCH-LOOP.
      EXEC SQL FETCH titlelist INTO
           :TITLE-ID, :BOOK-NAME END-EXEC
      DISPLAY "TITLE ID : ", TITLE-ID
      DISPLAY "TITLE    : ", BOOK-NAME
      IF SQLCODE = 100
           DISPLAY "NO RECORDS TO FETCH. END OF PROGRAM RUN."
      ELSE
              DISPLAY "UPDATE/DELETE THIS RECORD ? "
              ACCEPT ANSWER

              IF ANSWER = "U"
                   DISPLAY "ENTER NEW TITLE :"
                   ACCEPT BOOK-NAME
                   EXEC SQL UPDATE titles SET title = :BOOK-NAME
                  WHERE CURRENT OF titlelist END-EXEC.
              ELSE
              IF ANSWER = "D"
                   EXEC SQL DELETE titles WHERE CURRENT OF
                       titlelist  END-EXEC
              END-IF
      END-IF.
      END-IF.
```

使用法

- *procedure_name* は、1 つの select 文だけで構成してください。
- カーソルを使用して実行したストアド・プロシージャの出力パラメータを取得することはできません。
- カーソルを使用して実行したストアド・プロシージャのリターン・ステータスを取得することはできません。

参照

close、deallocate cursor、declare cursor (静的)、declare cursor (動的)、fetch、open、update

declare scrollable cursor

説明

スクロール可能カーソルを宣言します。

構文

```
EXEC SQL DECLARE <curs_name>
                              [ <cursor sensitivity> ]
                              [ <cursor scrollability> ] CURSOR
                              FOR <cursor specification>
<cursor sensitivity> : : =
                          SEMI_SENSITIVE
                          | INSENSITIVE
<cursor scrollability> : : =
                          SCROLL
                          NO SCROLL
<cursor specification> : : =
          <select statement> [ <updatability clause> ]
<updatability clause> : : =
          FOR {READ ONLY | UPDATE [ OF <column name list> ]}
END-EXEC
```

パラメータ

cursor sensitivity

半反映型または非反映型のカーソルを宣言します。

cursor scrollability

スクロール可能または非スクロール可能なカーソルを宣言します。

注意 スクロール可能カーソルはフェッチ・ループを使用するのではなく、単一のフェッチ呼び出しを使用します。フェッチ・ループを使用するのは、非スクロール可能カーソルと前方スクロールのみのカーソルだけです。

例

```
EXEC SQL DECLARE c1 INSENSITIVE SCROLL CURSOR FOR
    select title_id, royalty
    from authors
    where royalty < 25 END-EXEC.
EXEC SQL OPEN c1 END-EXEC.
```

使用法

- *cursor sensitivity* を INSENSITIVE として指定した場合、SCROLL は暗黙的に指定されません。
- *cursor sensitivity* を INSENSITIVE または SEMI_SENSITIVE として指定せず、カーソル宣言で SCROLL も指定しない場合、カーソルは読み込み専用のスクロール可能カーソルとなり、変更反映の可否は指定に従います。
- *cursor sensitivity* を指定しない場合、カーソルは読み込み専用で非反映型の非スクロール可能カーソルとして宣言されます。
- *cursor scrollability* を SCROLL として指定すると、カーソルは INSENSITIVE になります。
- *cursor scrollability* を指定しない場合、デフォルトは NO SCROLL であり、カーソルは読み込み専用の非スクロール可能カーソルとして宣言されます。

参照

scroll fetch、open

delete (カーソル位置)

説明

オープンしているカーソルの現在のカーソル位置によって示されたローをテーブルから削除します。

構文

exec sql [at *connection_name*] delete
 [from] *table_name*
 where current of *cursor_name* end-exec

パラメータ

table_name

ローが削除されるテーブルの名前です。

where current of *cursor_name*

この句を指定すると、カーソル *cursor_name* の現在のカーソル位置によって示されたテーブルのローを Adaptive Server Enterprise が削除します。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01      PUB-NAME    PIC X(40).
            01      PUB-ID      PIC X(4).
            01      PUB-CTY     PIC X(15).
            01      PUB-ST      PIC X(2).
            01      ANSWER      PIC X(1).
 EXEC SQL END DECLARE SECTION END-EXEC.
      ....
      ....
 EXEC SQL DECLARE delcursor CURSOR FOR
      SELECT * FROM publishers END-EXEC.

 EXEC SQL OPEN delcursor END-EXEC.
 PERFORM FETCH-LOOP UNTIL SQLCODE = 100.
 EXEC SQL CLOSE delcursor END-EXEC.
 EXEC SQL DEALLOCATE CURSOR delcursor END-EXEC.
 EXEC SQL COMMIT WORK END-EXEC.
      ...
 FETCH-LOOP.
      EXEC SQL FETCH delcursor INTO
           :PUB-ID, :PUB-NAME,
           :PUB-CTY, PUB-ST END-EXEC.
      DISPLAY "PUB ID    :", PUB-ID
      DISPLAY "PUB NAME  :", PUB-NAME
      DISPLAY "PUB CITY  :", PUB-CTY
      DISPLAY "PUB STATE :", PUB-ST

      IF SQLCODE = 100
           DISPLAY "NO MORE RECORDS TO FETCH. END OF PROGRAM RUN."
      ELSE
           DISPLAY "DELETE THIS RECORD ?(Y/N) "
           ACCEPT ANSWER
           IF ANSWER = "Y"
           EXEC SQL DELETE publishers WHERE CURRENT OF
                delcursor  END-EXEC
      END-IF.
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の delete 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- このフォーマットの delete 文は、そのカーソルの *cursor_name* をオープンした接続で実行してください。delete 文内で at*connection_name* 句を指定する場合、その句は *cursor_name* をオープンした open cursor 文の at*connection_name* 句と一致していなければなりません。
- カーソルを for read only で宣言したり、select 文内に order by 句を指定した場合、delete 文は失敗します。

参照

close、declare cursor、fetch、open、update

delete (検索条件)

説明

検索条件によって指定されたローを削除します。

構文

exec sql [at *connection_name*] delete *table_name_1*
[from *table_name_n*
[, *table_name_n*]...]
[where *search_conditions*] end-exec

パラメータ

table_name_1

この delete 文によってローが削除されるテーブルの名前です。

from *table_name_n*

table_name_1 のどのローが削除されるかを決定するために、*table_name_1* とジョインされるテーブルの名前です。delete 文は、*table_name_n* からはローを削除しません。

where *search_conditions*

どのローが削除されるかを指定します。where 句を省略した場合、delete 文は *table_name_1* のすべてのローを削除します。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     AU-FNAME      PIC X(30).
      01     AU-LNAME      PIC X(30).
      01     AU-ID         PIC X(11).
      01     TITLE-ID      PIC X(6).
 EXEC SQL END DECLARE SECTION END-EXEC.
           ...
 EXEC SQL WHENEVER SQLERROR PERFORM ROLLBACK-PARA.

 EXEC SQL USE pubs2 END-EXEC.

 DISPLAY "AUTHOR FIRST NAME ? "
 ACCEPT AU-FNAME.
 DISPLAY "AUTHOR LAST NAME ? "
```

```
 ACCEPT AU-LNAME.

 EXEC SQL SELECT au_id FROM authors INTO :AU-ID
         WHERE au_fname = :AU-FNAME
              AND au_lname = :AU-LNAME END-EXEC.

 EXEC SQL BEGIN TRANSACTION END-EXEC.

* Delete matching records from the 'au_pix' table.
 EXEC SQL DELETE au_pix WHERE au_id = :AU-ID END-EXEC.

* Delete matching records from the 'blurbs' table.
 EXEC SQL DELETE blurbs WHERE au_id = :AU-ID END-EXEC.

* Delete matching records from the titleauthor table. Since
* we can't have titles associated with this author in other
* related tables, we delete those records too.
 EXEC SQL DECLARE selcursor CURSOR FOR
        SELECT title_id FROM titleauthor
            WHERE au_id = :AU-ID END-EXEC.
 EXEC SQL OPEN selcursor END-EXEC.
 PERFORM FETCH-DEL-LOOP UNTIL SQLCODE = 100.

 EXEC SQL CLOSE selcursor END-EXEC.
 EXEC SQL DEALLOCATE CURSOR selcursor END-EXEC.

* Delete matching records from the 'authors' table.
 EXEC SQL DELETE authors WHERE au_id = :AU-ID END-EXEC.

* Commit all the transactions to the database.
 EXEC SQL COMMIT TRANSACTION END-EXEC.

               ...

 FETCH-DEL-LOOP.
    EXEC SQL FETCH selcursor INTO :TITLE-ID END-EXEC
    IF SQLCODE <> 100
       EXEC SQL DELETE salesdetail WHERE title_id = :TITLE-ID END-EXEC
       EXEC SQL DELETE roysched    WHERE title_id = :TITLE-ID END-EXEC
       EXEC SQL DELETE titles      WHERE title_id = :TITLE-ID END-EXEC
       EXEC SQL DELETE titleauthor WHERE CURRENT OF selcursor END-EXEC
    END-IF.
 END-FETCH-LOOP.

 * Rollback the transacion in case of errors.
  ROLLBACK-PARA.
  DISPLAY "ERROR!ROLLING BACK TRANSACTION!"
     DISPLAY "Error code is " SQLCODE.
     DISPLAY "Error message is " SQLERRMC.

  EXEC SQL ROLLBACK TRANSACTION END-EXEC.
               ...
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の delete 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- カーソル・ポインタの現在の位置によって指定されるローを削除する必要がある場合は、delete (**カーソル位置**) 文を使用してください。

参照

close、declare cursor、fetch、open、update

describe input (SQL 記述子)

説明

準備された動的 SQL 文の動的パラメータ・マーカについての情報を取得し、その情報を SQL 記述子に保管します。

使用可能な SQL 記述子のデータ型のコードのリストについては、表 9-5 (161 ページ) を参照してください。

構文

exec sql describe input *statement_name*
using sql descriptor *descriptor_name* end-exec

パラメータ

statement_name

情報を得るための準備文で指定された文の名前です。*statement_name* は、準備文を示す必要があります。

sql descriptor

descriptor_name を SQL 記述子として指定します。

descriptor_name

準備文内の動的パラメータ・マーカについての情報が格納される SQL 記述子の名前です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     QUERY     PIC X(100).
      01     NIN      PIC S9(9) COMP.
      01     COUNTER PIC S9(9) COMP.
      01     COLTYPE PIC S9(9) COMP.
      01     COLLEN      PIC S9(9) COMP.
 EXEC SQL END DECLARE SECTION END-EXEC.
      ...
 EXEC SQL ALLOCATE DESCRIPTOR din WITH MAX 256 END-EXEC.

 DISPLAY "ENTER QUERY :"
 ACCEPT QUERY.

 EXEC SQL PREPARE dynstmt FROM :QUERY END-EXEC.
 EXEC SQL DESCRIBE INPUT dynstmt USING
           SQL DESCRIPTOR din END-EXEC.
```

```
EXEC SQL GET DESCRIPTOR din :NIN = COUNT END-EXEC.
MOVE 1 TO COUNTER.
PERFORM GET-DESC-LOOP UNTIL COUNTER > NIN.
EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
EXEC SQL DEALLOCATE DESCRIPTOR din END-EXEC.
     ...
GET-DESC-LOOP.
     EXEC SQL GET DESCRIPTOR din VALUE
               :COUNTER :COLTYPE = TYPE END-EXEC
     EXEC SQL GET DESCRIPTOR din VALUE
               :COUNTER :COLLEN = LENGTH END-EXEC
     DISPLAY "TYPE OF INPUT = ", COLTYPE
     DISPLAY "INPUT  LENGTH = ", COLLEN
     ADD 1 TO COUNTER .
END-GET-DESC-LOOP.
```

使用法

- 文についての情報は、using 句で指定された記述子に書き込まれます。describe input 文の実行後に get descriptor 文を使用して、記述子からホスト変数に情報を抽出してください。
- 記述子を割り付けてからでないと、describe input 文を実行できません。

参照

allocate descriptor、deallocate descriptor、describe output、get descriptor、prepare、set descriptor

describe input (SQLDA)

説明

準備された動的 SQL 文の動的パラメータ・マーカについての情報を取得し、その情報を SQLDA 構造体に保管します。

構文

exec sql describe input *statement_name*
 using descriptor *descriptor_name* end-exec

パラメータ

statement_name

情報を得るための準備文で指定された文の名前です。*statement_name* は、準備文を示す必要があります。

descriptor

descriptor_name を SQLDA 構造体として指定します。

descriptor_name

準備文内の動的パラメータ・マーカについての情報が格納される SQLDA 構造体の名前です。

例

```
...
...
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01      QUERY      PIC X(100).
EXEC SQL END DECLARE SECTION END-EXEC.

    01      din.
        05 SD-SQLN PIC S9(4) COMP.
        05 SD-SQLD PIC S9(4) COMP.
        05 SD-COLUMN OCCURS 3 TIMES.
            10 SD-DATAFMT.
                15 SQL--NM PIC X(132).
                15 SQL--NMLEN PIC S9(9) COMP.
                15 SQL--DATATYPE PIC s9(9) COMP.
                15 SQL--FORMAT PIC S9(9) COMP.
                15 SQL--MAXLENGTH PIC S9(9) COMP.
                15 SQL--SCALE PIC S9(9) COMP.
                15 SQL--PRECISION PIC S9(9) COMP.
                15 SQL--STTUS PIC S9(9) COMP.
                15 SQL--COUNT PIC S9(9) COMP.
                15 SQL--USERTYPE PIC S9(9) COMP.
                15 SQL--LOCALE PIC S9(9) COMP.
            10 SD-SQLDATA PIC S9(9) COMP.
            10 SD-SQLIND  PIC S9(9) COMP.
            10 SD-SQLLEN  PIC S9(9) COMP.
            10 SD-SQLMORE PIC S9(9) COMP.
    01 TMP PIC Z(8)9.

            ...

    DISPLAY "ENTER QUERY :"
    ACCEPT QUERY.

    EXEC SQL ALLOCATE DESCRIPTOR din WITH MAX 256 END-EXEC.
    EXEC SQL PREPARE dynstmt FROM :QUERY END-EXEC.
    EXEC SQL DECLAR selcursor CURSOR FOR dynstmt END-EXEC.
    EXEC SQL DESCRIBE INPUT dynstmt USING DESCRIPTOR din END-EXEC.

    * SD-SQLD contains the number of columns in the query being described
        MOVE SD-SQLD TO TMP.
        DISPLAY "Number of input parameters = ", SD-SQLD.

            ...
```

使用法

文についての情報は、using 句で指定された記述子に書き込まれます。get descriptor 文が実行されたあとに、SQLDA 構造体からの情報の読み込みができるようになります。

参照

allocate descriptor、deallocate descriptor、describe output、get descriptor、prepare、set descriptor

describe output (SQL 記述子)

説明

準備された動的 SQL 文の結果セットについてのロー・フォーマット情報を取得します。

使用可能な SQL 記述子のデータ型のコードのリストについては、表 9-5 (161 ページ) を参照してください。

構文

exec sql describe [output] *statement_name*
 using sql descriptor *descriptor_name* end-exec

パラメータ

output
　describe output 文では有効でないオプションのキーワードですが、SQL 標準に準拠しています。

statement_name
　実行する select 文を表す (前の prepare 文内で指定された) 名前です。

sql descriptor
　descriptor_name を SQL 記述子として指定します。

descriptor_name
　describe output 文によって返された情報を保管する SQL 記述子の名前です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
           01      QUERY         PIC X(100).
           01      NOUT          PIC S9(9) COMP.
           01      DATAVAL       PIC X(100).
           01      COUNTER       PIC S9(9) COMP.
           01      NUMCOLS       PIC S9(9) COMP.
           01      COLNAME       PIC X(32).
           01      COLTYPE       PIC S9(9) COMP.
           01      COLLEN        PIC S9(9) COMP.
 EXEC SQL END DECLARE SECTION END-EXEC.
            ...
 DISPLAY "ENTER QUERY :"
 ACCEPT QUERY.

 EXEC SQL ALLOCATE DESCRIPTOR desc_out WITH MAX 256 END-EXEC.
 EXEC SQL PREPARE dynstmt FROM :QUERY END-EXEC.
 EXEC SQL DECLARE selcursor CURSOR FOR dynstmt END-EXEC.
 EXEC SQL OPEN selcursor USING SQL DESCRIPTOR desc_out END-EXEC.
 EXEC SQL DESCRIBE OUTPUT dynstmt USING SQL DESCRIPTOR desc_out END-EXEC.

 PERFORM  FETCH-LOOP UNTIL SQLCODE = 100.
 EXEC SQL CLOSE selcursor END-EXEC.
 EXEC SQL DEALLOCATE CURSOR selcursor END-EXEC.
 EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
 EXEC SQL DEALLOCATE DESCRIPTOR desc_out END-EXEC.
            ...
 FETCH-LOOP.
      EXEC SQL FETCH selcursor INTO SQL DESCRIPTOR desc_out END-EXEC
```

```
    EXEC SQL GET DESCRIPTOR desc_out :NOUT = COUNT END-EXEC
    DISPLAY "COLS RETRIEVED = ", NOUT
    MOVE 1 TO COUNTER
    PERFORM GET-DESC-PARA UNTIL  COUNTER > NOUT.
END-FETCH-LOOP.

GET-DESC-PARA.
     EXEC SQL GET DESCRIPTOR desc_out VALUE :COUNTER
                      :COLNAME = NAME,
                      :COLTYPE = TYPE,
                      :COLLEN  = LENGTH
                  END-EXEC
       DISPLAY "NAME      :", COLNAME
       DISPLAY "TYPE      :", COLTYPE
       DISPLAY "LENGTH    :", COLLEN

       EXEC SQL GET DESCRIPTOR desc_out VALUE :COUNTER
                       :DATAVAL = DATA END-EXEC
       DISPLAY "DATA      :", DATAVAL
       DISPLAY " "
       ADD 1 TO COUNTER.
 END-GET-DESC-PARA.
```

使用法

- 得られる情報はタイプ、名前、長さ (数値であれば、精度と位取り)、null が許されるかどうかのステータス、結果セットの項目数です。
- この情報は、select カラム・リストからの結果カラムです。
- describe output 文を実行してから、準備文を実行してください。準備文の実行後に、describe output を実行してから、get descriptor を実行すると、結果は廃棄されます。

参照 allocate descriptor、describe input、execute、get descriptor、prepare

describe output (SQLDA)

説明 準備された動的 SQL 文の結果セットについてのロー・フォーマット情報を取得し、その情報を SQLDA 構造体に保管します。

構文

exec sql describe [output] *statement_name*
 using descriptor *sqlda_name* end-exec

パラメータ

output

describe output 文では有効でないオプションのキーワードですが、SQL 標準に準拠しています。

statement_name

実行する select 文を表す (前の prepare 文内で指定された) 名前です。

descriptor

descriptor_name を SQLDA 構造体として指定します。

sqlda_name

describe output 文によって返された情報を保管する SQLDA 構造体の名前です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01      QUERY          PIC X(100).
       01      CHARVAR        PIC X(100).
 EXEC SQL END DECLARE SECTION END-EXEC.

 01      dout.
                   05 SD-SQLN      PIC S9(4) COMP.
                   05 SD-SQLD      PIC S9(4) COMP.
                   05 SD-COLUMN OCCURS 3 TIMES.
                   10 SD-DATAFMT.
                      15 SQL--NM         PIC X(132).
                      15 SQL--NMLEN      PIC S9(9) COMP.
                      15 SQL--DATATYPE   PIC s9(9) COMP.
                      15 SQL--FORMAT      PIC S9(9) COMP.
                      15 SQL--MAXLENGTH   PIC S9(9) COMP.
                      15 SQL--SCALE       PIC S9(9) COMP.
                      15 SQL--PRECISION   PIC S9(9) COMP.
                      15 SQL--STTUS       PIC S9(9) COMP.
                      15 SQL--COUNT       PIC S9(9) COMP.
                      15 SQL--USERTYPE    PIC S9(9) COMP.
                      15 SQL--LOCALE      PIC S9(9) COMP.

                   10 SD-SQLDATA    PIC S9(9) COMP.
                   10 SD-SQLIND     PIC S9(9) COMP.
                   10 SD-SQLLEN     PIC S9(9) COMP.
                   10 SD-SQLMORE    PIC S9(9) COMP.
   01      TMP                 PIC Z(8)9.
   01      COLNUM              PIC S9(9) COMP.
   01      TMP1                PIC S9(9) COMP.
   01      TMP2                PIC S9(9) COMP.
   01      RETCODE             PIC S9(9) COMP.
       ...

 DISPLAY "ENTER QUERY :"
 ACCEPT QUERY.

 EXEC SQL ALLOCATE DESCRIPTOR dout WITH MAX 256 END-EXEC.
 EXEC SQL PREPARE dynstmt FROM :QUERY END-EXEC.
 EXEC SQL DECLARE selcursor CURSOR FOR dynstmt END-EXEC.
 EXEC SQL OPEN selcursor END-EXEC.
 EXEC SQL DESCRIBE OUTPUT dynstmt
              USING DESCRIPTOR dout END-EXEC.
```

```
MOVE 1  TO COLNUM.
MOVE 25 TO TMP1.
MOVE 0  TO TMP2.

CALL "SYBSETSQLDA" USING RETCODE dout COLNUM
      CHARVAR SYB-X-PIC TMP1 TMP2 SYB-NO-USAGE
            SYB-NO-SIGN.

EXEC SQL FETCH selcursor INTO DESCRIPTOR dout END-EXEC.
DISPLAY "CHARVAR = ", CHARVAR.

EXEC SQL CLOSE selcursor END-EXEC.
EXEC SQL DEALLOCATE CURSOR selcursor END-EXEC.
EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
EXEC SQL DEALLOCATE DESCRIPTOR dout END-EXEC.
```

使用法

- 得られる情報はタイプ、名前、長さ (数値であれば、精度と位取り)、null が許されるかどうかのステータス、結果セットの項目数など、SQLDA フィールド内に入るデータです。
- この情報は、select カラム・リストからの結果カラムです。

参照 describe input、execute、prepare

disconnect

説明 1 つ以上の Adaptive Server Enterprise への接続をクローズします。

構文

exec sql disconnect
{*connection_name* | current | DEFAULT| all} end-exec

パラメータ

connection_name
クローズする接続の名前です。

current
現在の接続をクローズすることを指定します。

DEFAULT
デフォルトの接続をクローズすることを指定します。次の例のような文字列変数を使用してデフォルトの *connection_name* を指定する場合は、このキーワードを大文字にしてください。

```
exec sql disconnect :hv;
```

all
アクティブなすべての接続をクローズすることを指定します。

例

```
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01     SERV-NAME    PIC X(25).
       01     USER-NAME    PIC X(25).
       01     PASSWORD     PIC X(25).
       01     CONN-NAME    PIC X(25).
  EXEC SQL END DECLARE SECTION END-EXEC.
            ...
  MOVE "sa" TO USER-NAME.
  MOVE ""   TO PASSWORD.

* Make a default connection.
  EXEC SQL CONNECT :USER-NAME IDENTIFIED BY :PASSWORD END-EXEC.
  EXEC SQL SELECT @@servername into :srvname END-EXEC.
  DISPLAY "NOW CONNECTED TO SERVER ", srvname.

* Accept a server name from the user and make a new connection.
  DISPLAY "SERVER NAME? ".
  ACCEPT SERV-NAME.
  EXEC SQL CONNECT :USER-NAME IDENTIFIED BY :PASSWORD
                   At conn2 USING :SERV-NAME END-EXEC.

  EXEC SQL SELECT @@servername into :srvname END-EXEC
  DISPLAY "NOW CONNECTED TO SERVER ", srvname.

* Make a third connection.
  EXEC SQL CONNECT :USER-NAME IDENTIFIED BY :PASSWORD
                   At conn3 USING :SERV-NAME END-EXEC.

  EXEC SQL SELECT @@servername into :srvname END-EXEC.
  DISPLAY "NOW CONNECTED TO SERVER ", srvname.

* Now set the current connection to DEFAULT.
  EXEC SQL SET CONNECTION DEFAULT END-EXEC.

* Now disconnect the first connection which is the default.
  DISPLAY "DISCONNECTING DEFAULT!".
  EXEC SQL DISCONNECT DEFAULT END-EXEC.

* Now sdet the current connection to connection2.
  EXEC SQL SET CONNECTION conn2 END-EXEC.

* Now disconnect the third connection.
  DISPLAY "DISCONNECTING THIRD!".
  EXEC SQL DISCONNECT conn3 END-EXEC.

* Disconnect remaining connections - case 'conn2' will be closed.
  DISPLAY "DISCONNECTING ALL!".
  EXEC SQL DISCONNECT ALL END-EXEC.
```

使用法

- disconnect キーワードは、単独では有効な文ではありません。したがって、このキーワードのあとに *connection_name*、current、DEFAULT、または all を指定してください。
- 接続をクローズすると、その接続に対応するすべてのメモリとリソースが解放されます。
- disconnect キーワードは現在のトランザクションを commit しないで、そのトランザクションをロールバックします。その接続で非連鎖トランザクションがアクティブである場合、disconnect キーワードはどのようなセーブポイントも無視して、そのトランザクションをロールバックします。
- 接続をクローズすると、オープンしているカーソルがクローズされて、テンポラリの Adaptive Server Enterprise オブジェクトは削除されます。また、Adaptive Server Enterprise でその接続が持っているすべてのロックが解放されて、Adaptive Server Enterprise へのネットワーク接続はクローズされます。

参照

commit work、commit transaction、connect、rollback transaction、rollback work

exec

説明

システム・プロシージャまたはユーザ定義ストアド・プロシージャを実行します。

構文

```
exec sql [at connection_name]
 exec [:status_var = status_value] procedure_name
[([[@parameter_name =]param_value [out[put]]],...)]
 [into :hostvar_1 [:indicator_1]
 [, hostvar_n [indicator_n,…]]]
 [with recompile] end-exec
```

注意 exec 文を Embedded SQL の execute 文と混同しないでください。これらの文は関係がありません。ただし、Embedded SQL の exec 文は Transact-SQL の execute 文と同じです。

パラメータ

status_var
ストアド・プロシージャのリターン・ステータスを受信するホスト変数です。

status_value
ストアド・プロシージャのリターン・ステータス変数 *status_var* の値です。

procedure_name
実行するストアド・プロシージャの名前です。

parameter_name
ストアド・プロシージャのパラメータの名前です。

param_value
ホスト変数またはリテラル値です。

output
ストアド・プロシージャがパラメータ値を返すことを示します。ストアド・プロシージャ内の対応するパラメータも、output キーワードを使用して作成されている必要があります。

into *:hostvar_1*
このキーワードを指定すると、ストアド・プロシージャから返されるローはここで指定したホスト変数 (*hostvar_1* ～ *hostvar_n*) に保管されます。それぞれのホスト変数にはインジケータ変数を指定できます。

with recompile
このキーワードを指定すると、プロシージャが実行されるたびに Adaptive Server Enterprise はこのストアド・プロシージャに対する新しいクエリ・プランを作成します。

例　　***例 1***

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     TITLE-ID   PIC X(6).
      01     TOTAL-DISC PIC S9(9).
      01     RET-STATUS PIC S9(9).
 EXEC SQL END DECLARE SECTION END-EXEC.
      ...
 EXEC SQL CREATE PROC get_sum_discounts(@title_id tid,
           @discount int output) as
           begin
                select @discount = sum (qty*discount)
                     from salesdetail
                     where title_id = @title_id
           end
 END-EXEC.

 EXEC SQL SET CHAINED ON END-EXEC.
 DISPLAY "TITLE ID ? ".
 ACCEPT TITLE-ID.

 EXEC SQL EXEC :RET-STATUS = get_sum_discounts
            :TITLE-ID, :TOTAL-DISC OUT END-EXEC.

 DISPLAY "TOTAL DISCOUNTS FOR TITLE ID ", TITLE-ID," = ",TOTAL-DISC.

      ...
```

例 2

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     PUB-ID         PIC X(4).
      01     NAME       PIC X(25).
      01     CITY       PIC X(25).
      01     STATE      PIC X(2).
      01     RET-STATUS PIC S9(9).
 EXEC SQL END DECLARE SECTION END-EXEC.
      ...
 EXEC SQL CREATE PROC get_publishers(@pubid char(4))
       as
           select pub_name, city, state from
             publishers where pub_id = @pubid
      END-EXEC.

 DISPLAY " DETAIL RECORD FOR PUBLISHER ? ".
 ACCEPT PUB-ID.

 EXEC SQL EXEC :RET-STATUS = get_publishers :PUB-ID
                INTO :NAME, :CITY, :STATE END-EXEC.

 IF RET-STATUS = 0
      DISPLAY " PUBLISHER NAME : ", NAME
      DISPLAY " CITY           : ", CITY
      DISPLAY " STATE           : ", STATE
```

使用法

- クライアント・アプリケーションにローを返すことができるのは、1 つの select 文だけです。
- ロー・データを返すことができる select 文がストアド・プロシージャに含まれている場合は、次の 2 つの方法のどちらかを使用してそのデータを保管してください。つまり、exec 文の into 句を使用する方法とそのプロシージャのカーソルを宣言する方法のどちらかです。into 句を使用した場合、ユーザが指定したホスト変数が配列でないかぎり、ストアド・プロシージャは複数のデータ・ローを返しません。
- 値 *param_value* にはホスト変数またはリテラル値を指定できます。ただし output キーワードを使用する場合は、*param_value* をホスト変数にしてください。
- *parameter_name* に output キーワードを指定できるのは、*procedure_name* を作成した create procedure 文の対応するパラメータにもこのキーワードを使用した場合にかぎります。
- Embedded SQL の exec 文の機能は、Transact-SQL の execute 文とほとんど同じです。

参照

declare cursor (ストアド・プロシージャ)、select

exec sql

説明

ホスト言語プログラムに埋め込まれた SQL 文の始めにマークを付けます。

構文

exec sql [at *connection_name*] *sql_statement* end-exec

パラメータ

at

この句を指定すると、SQL 文 *sql_statement* はその Adaptive Server 接続の *connection_name* で実行されます。

connection_name

sql_statement が実行される Adaptive Server 接続を識別する接続名です。*connection_name* は前の connect 文内で定義してください。

sql_statement

Transact-SQL 文または Embedded SQL 文です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     SITE1     PIC X(25).
          01     SALES1     PIC S9(9) COMP.
     EXEC SQL END DECLARE SECTION END-EXEC.

     EXEC SQL CONNECT "user" identified by "password"
          AT server1 USING "server1" END-EXEC.
     EXEC SQL CONNECT "user" identified by "password"
          AT server2 USING "server2" END-EXEC.


     EXEC SQL AT server1 USE pubs2 END-EXEC.
     EXEC SQL AT server2 USE pubmast END-EXEC.

     EXEC SQL AT server1 SELECT count(*) FROM sales
               INTO :sales1 END-EXEC.

     MOVE "server1" TO SITE1.

     EXEC SQL SET CONNECTION server2 END-EXEC.
     EXEC SQL INSERT numsales VALUES (:SITE1, :SALES1) END-EXEC.
     EXEC SQL COMMIT WORK END-EXEC.
          ...
```

使用法

- ホスト言語に埋め込まれた SQL 文は “exec sql” で始めてください。キーワード exec sql は、ホスト言語の文を開始できる場所であればどこにでも指定できます。
- 文 *sql_statement* を 1 つまたは複数のプログラム行に渡って指定できます。ただし、改行と継続行についてのホスト言語の規則に準拠する必要があります。

- at 句の影響が及ぶのは文 *sql_statement* だけです。この句は後続の SQL 文に影響を与えることも、現在の接続をリセットすることもありません。
- *sql_statement* が次に示す SQL 文のうちのどれかである場合は、at 句は有効ではありません。

表 *9-2: exec sql* の *at* 句を使用できない文

allocate descriptor	begin declare section	connect
deallocate descriptor	declare cursor (動的)	end declare section
exit	get diagnostics	include file
include sqlca	set connection	set diagnostics
whenever		

- *connection_name* は前の connect 文内で定義してください。
- ターミネータを使用してそれぞれの Embedded SQL 文を終了させてください。COBOL 言語の場合、ターミネータはキーワード end-exec です。

参照

begin declare section、connect、disconnect、set connection

execute

説明

準備文から動的 SQL 文を実行します。

execute immediate (134 ページ) を参照してください。

構文

```
exec sql [at connection_name] execute statement_name
 [into {host_var_list |
    descriptor descriptor_name |
    sql descriptor descriptor_name}]
 [using {host_var_list |
    descriptor descriptor_name |
    sql descriptor descriptor_name}] end-exec
```

注意 Embedded SQL の execute 文を Embedded SQL の exec 文または Transact-SQL の execute 文と混同しないでください。

パラメータ

statement_name

前の prepare 文内で定義された文に対するユニークな識別子です。

descriptor_name

文の動的パラメータ・マーカ、または select カラム・リストを記述するメモリ領域または SQLDA 構造体を指定します。

into

文が select 文 (単一ローの選択) を実行するときには、into 句が必要です。into 句の対象は SQL 記述子、SQLDA 構造体、あるいは 1 つまたは複数の Embedded SQL ホスト変数の一覧です。

host_var_list 中の各ホスト変数は、declare セクションの中で最初に定義しなければなりません。null データ値が取得された場合にそのことを示すために、インジケータ変数をホスト変数に対応付けることができます。

descriptor

descriptor_name を SQLDA 構造体として指定します。

sql descriptor

descriptor_name を SQL 記述子として指定します。

using

host_var_list 内の動的パラメータ・マーカと置き換えられるホスト変数です。declare セクション内で定義しなければならないホスト変数は、リストされた順に置き換えられます。この句は、*statement_name* に動的パラメータ・マーカが含まれているときだけ使います。また、動的パラメータ・マーカの値を動的記述子に入れることもできます。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
        01     DEMO-BUF   PIC X(100).
        01     TITLE-ID   PIC X(6).
        01     ORDER-NO   PIC X(20).
        01     QTY        PIC S9(9).
   EXEC SQL END DECLARE SECTION END-EXEC.
            ...
   MOVE "INSERT salesdetail(ord_num, title_id, qty) VALUES( :?, :?, :?)"
 -                  TO DEMO-BUF.
   EXEC SQL PREPARE ins_stmt FROM :DEMO-BUF END-EXEC.

   DISPLAY "RECORDING BOOK SALES".
   DISPLAY "ORDER # ? ".
   ACCEPT ORDER-NO.
   DISPLAY "TITLE ID ? ".
   ACCEPT TITLE-ID.
   DISPLAY "QTY SOLD? ".
   ACCEPT QTY.

   EXEC SQL EXECUTE ins_stmt USING :ORDER-NO, :TITLE-ID, :QTY END-EXEC.
            ...
```

使用法

- execute は動的 SQL のメソッド 2 の 2 番目の手順です。最初の手順は prepare 文です。
- prepare および execute は、複数ローの select 以外の SQL 文に有効です。複数ローの select の場合には、動的カーソルを使用してください。
- *statement_name* 内の文に動的パラメータ・マーカ (“?”) を記述できます。マーカは、文が実行する前にホスト変数値が置き換えられる位置を示します。
- execute キーワードは、この文と exec 文を区別します。exec の詳細については、exec (128 ページ) リファレンス・ページを参照してください。

参照

declare section、get descriptor、prepare、set descriptor

execute immediate

説明

文字列ホスト変数または引用符で囲まれた文字列に格納された動的 SQL 文を実行します。

構文

exec sql [at *connection_name*] execute immediate
{:*host_variable* | “string”} end-exec

パラメータ

host_variable

declare セクション内で定義された文字列ホスト変数です。execute immediate を呼び出す前に、ホスト変数は、文法的に正しい完全な Transact-SQL 文に設定されていなければなりません。

string

host_variable の代わりに使用される、引用符で囲まれたリテラル Transact-SQL 文の文字列です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     HOST-VAR    PIC X(100).
     EXEC SQL END DECLARE SECTION END-EXEC.
          ...
     DISPLAY "ENTER A NON-SELECT SQL STATEMENT: ".
     ACCEPT HOST-VAR.

     EXEC SQL EXECUTE IMMEDIATE :HOST-VAR END-EXEC.
          ...
```

使用法

- execute immediate 文を使用することは、動的 SQL のメソッド 1 を使用することを意味します。動的 SQL のメソッドの詳細については、「第 7 章 動的 SQL の使い方」を参照してください。
- *host_variable* 内の文は、メッセージを除いてプログラムに結果を返すことができません。したがって、その文には select 文などは指定できません。

- Embedded SQL プリコンパイラは、*host_variable* 内に保管された文の構文をチェックしないで Adaptive Server Enterprise に送信します。その文の構文が正しくない場合は、Adaptive Server Enterprise がエラー・コードとエラー・メッセージをプログラムに返します。
- ホスト変数から動的 SQL 文に値を置き換えるには、prepare 文と execute 文を使用してください (動的 SQL のメソッド 2)。
- 結果を返す動的 SQL 文とともに select 文を実行するには、prepare 文、open 文、fetch 文を使用してください (動的 SQL のメソッド 3)。

参照　execute、prepare

exit

説明　Client-Library をクローズして、プログラムに割り付けられたすべての Embedded SQL リソースの割り付けを解除します。

構文　exec sql exit end-exec

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     HOST-VAR    PIC X(100).
     EXEC SQL END DECLARE SECTION END-EXEC.
          ...
     EXEC SQL SELECT getdate() INTO :HOST-VAR END-EXEC.

     DISPLAY "THE CURRENT DATE AND TIME IS: ", HOST-VAR.

* Note that the exit statement must be the last embedded SQL statement
* in the program.

     EXEC SQL EXIT END-EXEC.
```

使用法

- exit 文は、プログラムがオープンしたすべての接続をクローズします。また exit 文は、プログラムに割り付けられたすべての Embedded SQL リソースと Client-Library リソースの割り付けも解除します。
- exit 文はすべてのプラットフォームで有効ですが、必須なのは一部のプラットフォームのみです。『Open Client/Server プログラマーズ・ガイド補足』を参照してください。
- exit 文の使用後は、Client-Library をもう一度初期化しないと Client-Library 機能を使用できません。Client-Library の初期化の詳細については、『Open Client Client-Library/C プログラマーズ・ガイド』を参照してください。
- exit 文は Sybase の拡張機能であり、SQL 標準では定義されていません。

参照　disconnect

fetch

説明

現在のカーソル・ローからホスト変数または動的記述子にデータ値をコピーします。

構文

exec sql [at *connection_name*] fetch [rebind | norebind] *cursor_name*
into {:*host_variable* [[indicator]:*indicator_variable*]
 [,:*host_variable*
 [[indicator]:*indicator_variable*]]… |
 descriptor *descriptor_name* |
 sql descriptor *descriptor_name*} end-exec

パラメータ

rebind | norebind
: ホスト変数がこの fetch 文に対して再バインドを要求するかどうかを指定します。rebind 句は、再バインドを制御するプリコンパイラ・オプションを無効にします。

cursor_name
: カーソルの名前です。この名前は前の declare cursor 文内で定義します。

host_variable
: declare セクション内で定義されたホスト言語変数です。

indicator_variable
: 前の declare セクションで宣言された 2 バイトのホスト変数です。対応する変数の値が null の場合、fetch 文はインジケータ変数を -1 に設定します。トランケーションが行われる場合、fetch 文はインジケータ変数を結果カラムの実際の長さに設定します。トランケーションが行われない場合は、fetch 文はインジケータ変数を *0* に設定します。

descriptor
: *descriptor_name* を SQLDA 構造体として指定します。

sql descriptor
: *descriptor_name* を SQL 記述子として指定します。

descriptor_name
: 結果セットを保持する動的記述子の名前です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     TITLE-ID   PIC X(6).
      01     BOOK-NAME  PIC X(80).
      01     BOOK-TYPE  PIC X(12).
      01     I-TITLE    PIC S9(9).
      01     I-TYPE     PIC S9(9).
 EXEC SQL END DECLARE SECTION END-EXEC.
      ...
 EXEC SQL DECLARE title_list CURSOR FOR
      SELECT type, title_id, title FROM titles
      ORDER BY type END-EXEC.
```

```
EXEC SQL OPEN title_list END-EXEC.
PERFORM FETCH-LOOP UNTIL SQLCODE = 100.
EXEC SQL CLOSE title_list END-EXEC.
     ...
FETCH-LOOP.
     EXEC SQL FETCH title_list INTO
               :BOOK-TYPE :I-TYPE,
               :TITLE-ID,
               :BOOK-NAME :I-TITLE END-EXEC
* Check the indicator value - if not null display the value, else
* display UNDECIDED.
     IF I-TYPE <> -1
          DISPLAY "TYPE : ", BOOK-TYPE
     ELSE
          DISPLAY "TYPE : UNDECIDED"
     END-IF

     DISPLAY "TITLE ID : ", TITLE-ID

     IF I-TITLE <> -1
          DISPLAY "TITLE : ", BOOK-NAME
     ELSE
          DISPLAY "TITLE : UNDECIDED"
     END-IF.
END-FETCH-LOOP.
```

使用法

- fetch 文は、動的 SQL 内のカーソルまたは静的カーソルのいずれにも使用できます。
- open 文を実行してから、fetch 文を実行してください。
- オープン・カーソルでの最初の fetch は、カーソルの結果テーブルから最初のローまたはローのグループを返します。後続の fetch は、それぞれ次のローまたはローのグループを返します。
- 複数のローを配列にフェッチできます。
- 「現在のロー」は最後にフェッチされたローです。ローを更新または削除するには、update 文または delete 文に where current of *cursor_name* 句を使用します。これらの文は、ローがフェッチされるまで無効です。
- カーソルからすべてのローがフェッチされたあと、fetch の呼び出しは SQLCODE を *100* に設定します。select 文の実行によって結果が生成されない場合、最初のフェッチ時に SQLCODE が 100 に設定されます。
- 結果セットの各カラムには *host_variable* が 1 つだけ必要です。
- rebind も norebind も指定しない場合、バインド動作はプリコンパイラ・オプション -b によって決定されます。プリコンパイラ・オプションの詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

- *indicator_variable* が null 値を受信できるように *host_variable* を指定する必要があります。インジケータ変数のないホスト変数が null 値をフェッチする場合、実行時エラーが発生します。
- Client-Library は、可能な場合に、結果カラムのデータ型を対応するホスト変数のデータ型に変換します。Client-Library がデータ型を変換できない場合には、エラー・メッセージが表示されます。変換ができない場合にはエラーが発生します。

参照

allocate descriptor、close、declare、delete (**カーソル位置**)、open、prepare、update

scroll fetch

説明

CURSOR OPEN の時点での ROW_COUNT 指定に応じて、カーソル結果セットから単一または複数のローをフェッチします。

カーソルをスクロール可能カーソルとして指定している場合、FETCH 文の *fetch orientation* でフェッチ方向を指定します。

カーソルをスクロール可能として指定していない場合、FETCH は、結果セットの次のローを取得します。

構文

```
EXEC SQL FETCH [ <fetch orientation> ]
                        [ FROM ] <cursor name>
                        { [ INTO <fetch target list> ] |
                        [SQL DESCRIPTOR <>]
<fetch orientation> : : =
                      | NEXT
                      | PRIOR
                      | FIRST
                      | LAST
                      | ABSOLUTE <fetch_offset>
                      | RELATIVE <fetch_offset>

<fetch offset> : : =
                      <signed_numeric_literal>
<fetch target list> : : =
                      <target specification>
                      [ { <comma> <target specification> } ]
END-EXEC
```

パラメータ

fetch orientation

NEXT、PRIOR、FIRST、LAST、ABSOLUTE、RELATIVE のいずれかとして指定します。

fetch offset

位取り 0 の符号付き真数値として指定します。

例

カーソルを宣言してオープンした後、ローをフェッチするには、次のようにします。

```
EXEC SQL FETCH LAST FROM c1 INTO :title,:roy END-EXEC.
```

前のローをフェッチするには、次のようにします。

```
EXEC SQL FETCH PRIOR FROM c1 INTO :title,:roy END-EXEC.
```

ロー 20 をフェッチするには、次のようにします。

```
EXEC SQL FETCH ABSOLUTE 20 FROM c1 INTO :title, :roy
          END-EXEC.
```

使用法

fetch orientation を指定しない場合、デフォルトは NEXT です。

注意　非スクロール可能カーソルに対して NEXT 以外の種類の *fetch orientation* を指定すると、次のメッセージが返されます。

フェッチ・タイプは、スクロール可能カーソルでのみ使用できます。

fetch orientation によってカーソルが最終ローより後または先頭ローより前に置かれた場合、*sqlca.sqlcode* は 100 に設定され、ローが見つからなかったことを示します。エラー・ハンドラがインストールされている場合は、エラー・ハンドラによって追加情報が提供される場合があります。

参照

declare、open

get descriptor

説明

動的パラメータ・マーカおよび select カラム・リスト属性の属性情報、および SQL 記述子からのデータを取得します。

SQL 記述子のデータ型コードのリストについては、表 9-5 (161 ページ) を参照してください。

構文

exec sql get descriptor *descriptor_name*
 {:*host_variable* = *count* |
 value *item_number* :*host_variable* = *item_name*
[, :*host_variable* = *item_name*]...} end-exec

パラメータ

descriptor_name
　準備文内の動的パラメータ・マーカまたは戻りカラムについての情報が入っている SQL 記述子の名前です。

host_variable
　declare セクション内で定義された変数です。

count
　取得される動的パラメータの数です。

item_number

get descriptor 文が情報を取得する *n* 番目の動的パラメータ・マーカまたは select カラムを指定する数字です。

item_name

取得される属性の名前です。詳細については、表 9-3 を参照してください。

表 *9-3:* 有効な *item_name* 値

値	説明
data	指定した SQL 記述子と対応する動的パラメータ・マーカまたはターゲットの値。インジケータが負の値である場合、このフィールドは未定義である。
indicator	動的パラメータ・マーカまたはターゲットと対応するインジケータ・パラメータの値。
length	指定した SQL 記述子のターゲットの動的パラメータ・マーカの文字単位での長さ。
name	動的パラメータ・マーカについての情報が入っている、指定した SQL 記述子の名前。
nullable	動的パラメータ・マーカで null 値が許可される場合は *0* であり、null 値が許可されない場合は 1 である。
precision	CS_NUMERIC 変数の精度の有効桁数を指定する整数。
returned_length	select カラム・リストからの character データ型の値の長さ。
scale	CS_NUMERIC 変数の小数点桁数を指定する整数。
type	ロー内のこのカラム (項目番号) のデータ型。値については、表 9-5 (161 ページ) を参照。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     QUERY          PIC X(100).
      01     CHARBUF        PIC X(100).
      01     NUMCOLS        PIC S9(9) COMP.
      01     COLNUM         PIC S9(9) COMP.
      01     COLTYPE        PIC S9(9) COMP.
      01     INTBUF         PIC S9(9).
 EXEC SQL END DECLARE SECTION END-EXEC.
      ...
 DISPLAY "ENTER A SELECT STATEMENT :"
 ACCEPT QUERY.
 EXEC SQL ALLOCATE DESCRIPTOR big_desc WITH MAX 256 END-EXEC.
 EXEC SQL PREPARE  dynstmt FROM :QUERY END-EXEC.
 EXEC SQL EXECUTE dynstmt INTO SQL DESCRIPTOR big_desc END-EXEC.
 EXEC SQL GET DESCRIPTOR big_desc :NUMCOLS = COUNT END-EXEC.

 MOVE 1 TO COLNUM.
 PERFORM GET-DESC-LOOP UNTIL COLNUM > NUMCOLS.
 EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
 EXEC SQL DEALLOCATE DESCRIPTOR big_desc END-EXEC.
      ...
```

```
    GET-DESC-LOOP.
         EXEC SQL GET DESCRIPTOR big_desc
                  VALUE :COLNUM
                  :COLTYPE = TYPE END-EXEC
* Check the type data returned and store in appropriate host variables.
         IF COLTYPE = 4
              DISPLAY "INTEGER DATA! "
              EXEC SQL GET DESCRIPTOR big_desc
                   VALUE :COLNUM :INTBUF = DATA END-EXEC
         ELSE
              IF COLTYPE = 1
                     DISPLAY "CHARACTER DATA! "
                   EXEC SQL GET DESCRIPTOR big_desc
                   VALUE :COLNUM :CHARBUF = DATA END-EXEC

 * Handle other data types accordingly or store them all as characters.
                ...
         ADD 1 TO COLUMN.
    END-GET-DESC-LOOP.
```

使用法

- get descriptor 文は、指定した動的パラメータの数または属性についての情報、または準備文で指定された文の select リスト・カラムについての情報を返します。
- get descriptor 文は、describe input 文、describe output 文、execute 文、または fetch (動的) 文を実行したあとに実行してください。
- execute 文または fetch 文を実行してその記述子と対応するデータをサーバから取得するまでは、*data*、*indicator* または *returned_length* を取得できません。

参照　describe input、describe output、fetch、set descriptor

get diagnostics

説明　Client-Library からエラー、警告、情報メッセージを取得します。

構文

```
get diagnostics
 {:hv = statement_info [, :hv = statement_info]...|
 exception :condition_number
 :hv = condition_info [, :hv = condition_info]...}
 end-exec
```

パラメータ

statement_info

キーワード number は、現在サポートされている唯一の *statement_info* タイプです。このキーワードは例外の数の合計を診断キューに返します。

condition_info

sqlca_info、sqlcode_number、returned_sqlstate のいずれかのキーワードを使用します。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
           01     NUM-MSGS      PIC S9(9) COMP.
           01     CONDCNT       PIC S9(9) COMP.
      EXEC SQL END DECLARE SECTION END-EXEC.
           ...
      EXEC SQL GET DIAGNOSTICS :NUM-MSGS = NUMBER   END-EXEC.
      MOVE 1 TO CONDCNT.
      PERFORM GET-DIAG-PARA UNTIL CONDCNT > NUM-MSGS.
           ...
      GET-DIAG-PARA.
           EXEC SQL GET DIAGNOSTICS EXCEPTION
                 :CONDCNT :SQLCA = SQLCA_INFO END-EXEC
           DISPLAY "DIAG. SQLCODE     = ",SQLCODE
           DISPLAY "DIAG. MESSAGE     = ",SQLERRMC

           ADD 1 TO CONDCNT.
      END-GET-DIAG-PARA.
```

使用法

- 多くの Embedded SQL 文は複数の警告やエラーを発生させる傾向があります。通常、最初のエラーだけが、SQLCODE、SQLCA、または SQLSTATE によって報告されます。すべてのエラーを処理するには get diagnostics を使用してください。
- get diagnostics は、whenever 文の call 句、perform 句、または go to 句のターゲットであり、コード内で使用できます。
- 情報メッセージを取得する文のあとで get diagnostics を使用できます。

参照

whenever

include “filename”

説明

Embedded SQL のソース・ファイルに外部ファイルをインクルードします。

構文

exec sql include "filename" end-exec

パラメータ

“filename”

この文が指定されている Embedded SQL のソース・ファイルにインクルードされるファイルの名前です。

注意 COPY 文でサポートされる最大長は、ファイルとパス名を含めて 70 文字です。

例

例 *1*：*COPY* の使用

```
COPY "generic".
     ...
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01     SRV-NAME   PIC X(80).
     01     UID        PIC X(32).
     01     PASS       PIC X(32).
EXEC SQL END DECLARE SECTION END-EXEC.
     ...
MOVE USER-NAME TO UID.
MOVE PASSWORD  TO PASS.

EXEC SQL CONNECT :UID IDENTIFIED BY :PASS END-EXEC.

EXEC SQL SELECT @@servername INTO :SRV-NAME END-EXEC.

DISPLAY "CONNECTED TO SERVER ",SRV-NAME.
```

コピー・ファイル・コード

```
01      USER-NAME     PIC X(33)     VALUE IS "sa".
         01      PASSWORD      PIC X(33)     VALUE IS "syb123".
```

例 *2*：*INCLUDE* の使用

```
EXEC SQL INCLUDE "./generic" END-EXEC.

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01     SRV-NAME   PIC X(80).
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL CONNECT :USER-NAME IDENTIFIED BY :PASSWORD END-EXEC.

EXEC SQL SELECT @@servername INTO :SRV-NAME END-EXEC.

DISPLAY "CONNECTED TO SERVER ",SRV-NAME.
```

コピー・ファイル・コード

```
01      USER-NAME     PIC X(33)     VALUE IS "sa".
        01      PASSWORD      PIC X(33)     VALUE IS "syb123".
```

使用法

- Embedded SQL プリコンパイラは、すべての declare セクションと SQL 文を認識して、インクルード・ファイルを Embedded SQL のソース・ファイルの一部のように処理します。Embedded SQL プリコンパイラは、この処理の結果生じるホスト言語のソース・コードを生成されたファイルに書き込みます。
- インクルード・パスのプリコンパイラ・コマンド・ライン・オプションを使用して、インクルード・ファイルを検索するディレクトリを指定してください。『Open Client/Server プログラマーズ・ガイド補足』を参照してください。
- インクルード・ファイルは最大 32 個の深さまでネストできます。
- include "*filename*" 文はどこにでも使用できます。

参照 declare section

include sqlca

説明 Embedded SQL プログラムで (SQLCA (SQL 通信領域)) を定義します。

構文 exec sql include sqlca end-exec

例

```
EXEC SQL INCLUDE SQLCA END-EXEC.
          ...
     EXEC SQL UPDATE test SET col1 = col1 + 100 END-EXEC.
     IF SQLCODE = 0
         DISPLAY "UPDATED ",SQLERRD(3), " ROWS."
     ELSE
         IF SQLCODE = 100
             DISPLAY "NO ROWS WERE AFFECTED."
         ELSE
             DISPLAY "AN ERROR OCCURED - ",SQLERRMC.
         END-IF
     END-IF.
     EXEC SQL COMMIT WORK END-EXEC.
```

使用法 ホスト言語の宣言が可能な場所であればどこにでも include sqlca 文を使用できます。

参照 begin declare section

include sqlda

説明　Embedded SQL プログラムで SQLDA 構造体を定義します。

構文　exec sql include sqlda;

使用法　ホスト言語の宣言が可能な場所であればどこにでも include sqlda 文を使用できます。

initialize_application

説明　グローバルな CS_CONTEXT ハンドルに、アプリケーション名を設定するための呼び出しを生成します。プリコンパイル時に -x オプションを指定した場合は、cs_config(CS_SET、CS_EXTERNAL_CONFIG、CS_TRUE) プロパティも設定されます。

構文　exec sql initialize_application
[application_name "=" *application_name*] end-exec

例

```
EXEC SQL INCLUDE SQLCA END-EXEC.

   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01 SPID          PIC S9(9) COMP.
          01 PROG-NAME     PIC X(33).
          01 UID          PIC X(33).
          01 PASS          PIC X(33).
   EXEC SQL END DECLARE SECTION END-EXEC.

   PROCEDURE DIVISION.
   PO.

* The INITIALIZE_APPLICATION MUST be the FIRST embedded SQL statement
* in the program.

 EXEC SQL INITIALIZE_APPLICATION APPLICATION_NAME
             = "TEST" END-EXEC.
* The body of the main procedure division goes here including all ESQL
* statements.
   ...EXEC SQL CONNECT :UID IDENTIFIED BY :PASS END-EXEC.
      EXEC SQL SELECT @@spid INTO :SPID END-EXEC.
      EXEC SQL SELECT program_name INTO :PROG-NAME
         FROM master..sysprocesses
         WHERE spid = :SPID END-EXEC.
      DISPLAY "THIS APPLICATION'S NAME IN SYSPROCESSES IS ", PROG-NAME.

    ...EXEC SQL EXIT END-EXEC.
```

使用法

- *application_name* には文字列のリテラルと、アプリケーションの名前が入っている文字変数のどちらかを指定できます。
- initialize_application 文がアプリケーションで実行される最初の Embedded SQL 文である場合に -x オプションを指定すると、ct_init は外部設定オプションを使用して CS_CONTEXT 構造体の Client-Library 部分を初期化します。
- initialize_application が最初の Embedded SQL 文でない場合は、ct_init が外部設定オプションを選択することはありません。
- initialize_application 文がアプリケーションで実行される最初の Embedded SQL 文であるかどうかに関係なく、-x オプションを指定すると、exec sql connect 文は外部設定データを使用します。-e オプションも指定した場合、Sybase では設定データへのキーとしてサーバ名が使用されます。-e オプションを指定しない場合、アプリケーション名 (または DEFAULT) が設定データへのキーとして使用されます。
- -x オプションとアプリケーション名を指定した場合は、次のようになります。
  - ct_init はアプリケーション名を使用して、外部設定ファイルのどのセクションを初期化に使用するかを決定します。
  - アプリケーション名は connect 文の一部として Adaptive Server Enterprise に渡されます。アプリケーション名は、sysprocesses.program_name テーブルに入力されます。
- -x オプションを指定しないで -e オプションを指定した場合、ct_init は初期化時に外部設定データを使用しますが、すべての接続で外部設定データへのキーとしてサーバ名が使用されます。コマンド・ライン・オプションの詳細については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

参照

exit

open (動的カーソル)

説明

前に宣言されている動的カーソルをオープンします。

構文

```
exec sql [at connection_name] open cursor_name
[row_count = size] [using {host_var_list |
 descriptor descriptor_name |
 sql descriptor descriptor_name}] end-exec
```

パラメータ

cursor_name

declare cursor 文を使用して宣言されているカーソルに名前を付けます。

size
ネットワーク経由で 1 度に返されるローの数です。ホスト変数にフェッチされるローの数ではありません。*size* 引数にはリテラルまたは宣言されたホスト変数のどちらかを指定できます。

host_var_list
動的パラメータ・マーカの値が入っているホスト変数に名前を付けます。

descriptor
descriptor_name を SQLDA 構造体として指定します。

sql descriptor
descriptor_name を SQL 記述子として指定します。

descriptor_name
準備文で指定された文内の動的パラメータ・マーカについての情報が入っている動的記述子に名前を付けます。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
            01     DYNABUF     PIC X(200).
            01     TITLE-ID    PIC X(6).
            01     LNAME       PIC X(15).
            01     FNAME       PIC X(15).
            01     PHONE       PIC X(15).
       EXEC SQL END DECLARE SECTION END-EXEC.
            ...
       MOVE "SELECT a.au_lname, a.au_fname, a.phone
                  FROM authors a, titleauthor t
                  WHERE a.au_id = t.au_id
                  AND   t.title_id = ?" TO DYNABUF.

       EXEC SQL PREPARE dynastmt FROM :DYNABUF END-EXEC.
       EXEC SQL DECLARE who_wrote CURSOR FOR dynastmt END-EXEC.

       DISPLAY "LIST AUTHORS FOR WHAT TITLE ? "
       ACCEPT TITLE-ID.

       EXEC SQL OPEN who_wrote USING :TITLE-ID END-EXEC.
       PERFORM FETCH-LOOP UNTIL SQLCODE = 100.
       EXEC SQL CLOSE who_wrote END-EXEC.
       EXEC SQL DEALLOCATE CURSOR who_wrote END-EXEC.
       EXEC SQL DEALLOCATE dynastmt END-EXEC.
            ...
       FETCH-LOOP.
     EXEC SQL FETCH who_wrote INTO
                  :LNAME, :FNAME, :PHONE END-EXEC
            DISPLAY "LAST  NAME : ", LNAME
            DISPLAY "FIRST NAME : ", FNAME
            DISPLAY "PHONE         : ", PHONE.
       END-FETCH-LOOP.
```

使用法

- open 文は、対応する declare cursor 文で指定された文を実行します。その後ユーザは fetch 文を使用して、準備文の結果を取得できます。
- オープンできるカーソルの数に制限はありません。
- using 句は、select 文内の動的パラメータ・マーカ (“?”) をホスト変数または動的記述子の内容に置き換えます。

参照

close、declare、fetch、prepare

open (静的カーソル)

説明

事前に宣言されている静的カーソルをオープンします。この文はストアド・プロシージャを含む任意の静的カーソルをオープンするのに使用できます。

構文

exec sql [at *connection_name*] open *cursor_name*
[row_count = *size*] end-exec

パラメータ

cursor_name

オープンするカーソルの名前です。

row_count

ネットワーク経由で 1 度に返されるローの数です。ホスト変数にフェッチされるローの数ではありません。

size

Adaptive Server Enterprise からクライアントに同時に移動されるローの数です。ローがアプリケーションによってフェッチされるまで、クライアントはこれらのローをバッファします。このパラメータを使用して、ネットワークの効率をチューニングできます。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      01     TITLE-ID     PIC X(6).
      01     BOOK-NAME    PIC X(25).
      01     BOOK-TYPE    PIC X(15).
 EXEC SQL END DECLARE SECTION END-EXEC.
      01     ANSWER       PIC X(1).
      ...
 DISPLAY "TYPE OF BOOKS TO RETRIEVE ? ".
 ACCEPT BOOK-TYPE.
 EXEC SQL DECLARE titlelist CURSOR FOR
      SELECT title_id, substring(title,1,25) FROM
           titles WHERE type = :BOOK-TYPE END-EXEC.
```

```
EXEC SQL OPEN titlelist END-EXEC.
PERFORM FETCH-LOOP UNTIL SQLCODE = 100.
EXEC SQL CLOSE titlelist END-EXEC.
EXEC SQL DEALLOCATE CURSOR titlelist END-EXEC.
EXEC SQL COMMIT WORK END-EXEC.

FETCH-LOOP.
     EXEC SQL FETCH titlelist INTO :TITLE-ID, :BOOK-NAME END-EXEC.
     DISPLAY "TITLE ID : ", TITLE-ID
     DISPLAY "TITLE    : ", BOOK-NAME
     DISPLAY "UPDATE/DELETE THIS RECORD ? "
     ACCEPT ANSWER

     IF ANSWER = "U"
          DISPLAY "ENTER NEW TITLE :"
          ACCEPT BOOK-NAME
          EXEC SQL UPDATE titles SET title = :TITLE
                WHERE CURRENT OF titlelist END-EXEC
     ELSE
          IF ANSWER = "D"
               EXEC SQL DELETE titles WHERE CURRENT OF
                     titlelist  END-EXEC
          END-IF
     END-IF.
END-FETCH-LOOP.
```

使用法

- open は declare cursor 文によって指定された select 文を実行し、fetch 文に対する結果を準備します。
- オープンするカーソルの数に制限はありません。
- 静的カーソルのオープンは、そのカーソルが宣言されるファイルでのみ実行してください。静的カーソルのクローズはどのファイルでも実行できます。
- declare cursor 文に埋め込まれたホスト変数の値は、オープン時に受け取られます。
- *cursor_name* を指定する場合、割り付けを解除された静的カーソルの名前を使用できます。その場合、プリコンパイラは割り付けが解除されたカーソルと同じ名前の新しいカーソルを宣言してオープンします。つまり、プリコンパイラは割り付けが解除されたカーソルを再度オープンするのではなく、新しいカーソルを作成します。これら 2 つのカーソルの結果セットは異なる可能性があります。

open scrollable cursor

説明 事前に宣言されている静的カーソルをオープンします。

構文 EXEC SQL OPEN <cursor_name> [*ROW_COUNT* = *size*] END-EXEC

パラメータ

size
プリフェッチ・カウントとして指定します。この値はホスト配列サイズと同じです。

ROW_COUNT
ホスト配列をホスト変数として使用する場合にのみ指定します。

使用法 *size* の値は、ホスト配列サイズと同じです。

参照 scroll fetch、declare

prepare

説明 動的 SQL 文バッファの名前を宣言します。

構文

exec sql [at *connection_name*] prepare *statement_name* from
{:*host_variable* | "*string*"} end-exec

パラメータ

statement_name
文を参照するときに使用する識別子です。*statement_name* は文バッファをユニークに識別する必要があり、変数の命名についての SQL 識別子の規則に準拠する必要があります。また、有効な SQL 識別子を含む *host_variable* 文字列を指定することもできます。*statement_name* は 255 文字以下にしてください。

host_variable
実行可能な SQL 文を含んでいる文字列ホスト変数です。ホスト変数の値が置き換えられる select 文の任意の場所に動的パラメータ・マーカ (“?”) を記述してください。

string
host_variable の代わりに使用できるリテラル文字列です。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     DEMO-BUFFER     PIC X(120).
          01     STATE          PIC X(3).
     EXEC SQL END DECLARE SECTION END-EXEC.
          ...
* The 'select into table' statement returns no results
* to the program, so it does not need a cursor.
```

```
        MOVE "select * into #work from authors where state = ?" TO
-               DEMO-BUFFER.

        DISPLAY "STATE ? ".
        ACCEPT STATE.

        EXEC SQL PREPARE dynstmt FROM :DEMO-BUFFER END-EXEC.
        EXEC SQL EXECUTE dynstmt USING :STATE END-EXEC.

        EXEC SQL DEALLOCATE PREPARE dynstmt END-EXEC.
```

使用法

- 現在の実装では、Sybase はリテラル文字列またはホスト変数に保管される動的 SQL 文のテンポラリ・ストアド・プロシージャを作成します。
- prepare は、*host_variable* の内容を Adaptive Server Enterprise へ送信し、テンポラリ・ストアド・プロシージャに変換します。このテンポラリ・ストアド・プロシージャは、文を割り付け解除するか接続を切断するまで Adaptive Server Enterprise の tempdb に存在します。
- *statement_name* のスコープはプログラムに対してはグローバルですが、接続 *connection_name* に対してはローカルです。この文は、プログラムによってその文の割り付けが解除されるか、接続がクローズされるまで継続します。
- prepare は、動的 SQL のメソッド 2、3、および 4 の場合に有効です。
- メソッド 2 (prepare および execute) では、ホスト変数があれば、その値を execute 文が準備した文に置き換え、完成した文を Adaptive Server Enterprise に送信します。置き換えるホスト変数がなく、結果もない場合には、代わりに execute immediate を使用できます。
- メソッド 3 (prepare および fetch) では、declare cursor 文が、保存された select 文とカーソルを対応付けます。ホスト変数があれば、その値を open 文が select 文に置き換え、実行のために結果を Adaptive Server Enterprise に送ります。
- メソッド 2、3、4 (パラメータ記述子を持つ prepare および fetch) では、疑問符 (“?”) で表された動的パラメータ記述子で、ホスト変数で置き換えられる箇所を示します。
- 準備文は、その文が準備された接続上で実行してください。準備文を使用してカーソルを宣言する場合、そのカーソル上でのすべてのオペレーションは、準備文と同じ接続を使用します。
- *host_variable* 内の文は、ホスト変数の値を文に置き換える箇所を示す動的パラメータ・マーカを含むことができます。

参照　declare cursor、execute、execute immediate、deallocate prepare

rollback

説明

データベースの変更を、トランザクション・セーブポイント、またはトランザクションの開始点にロールバックします。

構文

exec sql [at *connection_name*]
 rollback [transaction | tran | work]
 [*transaction_name* | *savepoint_name*] end-exec

パラメータ

transaction | trans | work

rollback 文では、キーワード transaction、trans、work は交換可能ですが work だけは ANSI 準拠です。

transaction_name

ロールバックするトランザクションの名前です。

savepoint_name

save transaction 文でセーブポイントに割り当てた名前です。*savepoint_name* を省略した場合、Adaptive Server はトランザクション全体をロールバックします。

例

```
          ...
     EXEC SQL CONNECT "user" IDENTIFIED BY "password"
          AT connect1 USING "srvname" END-EXEC.
          ...
     EXEC SQL AT connect1 UPDATE test SET col1 = 'x' END-EXEC.
     IF SQLCODE = 0
          DISPLAY "ROWS UPDATED = ",SQLERRD(3)
     ELSE
DISPLAY "AN ERROR OCCURED -",SQLERRMC
          ESQL SQL AT connect1 ROLLBACK TRANSACTION END-EXEC
     END-IF.
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の rollback 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- トランザクション名とセーブポイント名は、Transact-SQL の識別子規則に準拠する必要があります。
- トランザクション名とセーブポイントは Transact-SQL の拡張機能であり、ANSI 標準には準拠していません。トランザクション名またはセーブポイント名を ANSI 標準に準拠するキーワード work とともに使用しないでください。

参照

begin transaction、commit

select

説明　データベース・オブジェクトからローを取得します。

構文

exec sql [at *connect_name*]
 select *select_list*
 into *destination*
from *table_name*... end-exec

パラメータ

select_list

Embedded SQL での *select_list* は変数の割り当てを実行できませんが、それ以外は Transact-SQL の select 文内の *select_list* と同じです。

destination

テーブルまたは 1 つ以上の Embedded SQL ホスト変数です。前の declare セクションで最初にそれぞれのホスト変数を定義してください。インジケータ変数はホスト変数と対応付けることができます。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     LNAME      PIC X(25).
          01     FNAME      PIC X(25).
          01     PHONE      PIC X(15).
          01     AU-ID      PIC X(12).
     EXEC SQL END DECLARE SECTION END-EXEC.

          ...


     DISPLAY "AUTHOR ID ? ".
     ACCEPT AU-ID.

     EXEC SQL SELECT au_lname, au_fname, phone
               INTO :LNAME, :FNAME, :PHONE
           FROM authors
           WHERE au_id = :AU-ID END-EXEC.

     IF SQLCODE = 100
DISPLAY "COULD NOT LOCATE AUTHOR ",AU-ID
     ELSE
          DISPLAY "DETAIL RECORD FOR AUTHOR:", AU-ID
               DISPLAY "NAME  :",LNAME, " ", FNAME
               DISPLAY "PHONE :",PHONE
     END-IF.
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の select 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- Embedded SQL プログラムでは、Transact-SQL の select 文の compute 句を使用できません。
- select 文内のホスト変数は、into 句以外では入力変数のみです。into 句内のホスト変数は出力変数です。
- 事前に宣言された入力ホスト変数は、リテラル値または Transact-SQL 変数が許される select 文内の任意の箇所で使用できます。入力ホスト変数にインジケータ変数を対応付けて、null 値を指定することができます。
- select 文が 2 つ以上のローを返す場合、その文の into 句内にあるそれぞれのホスト変数は、すべてのローを入れるのに十分な領域を持った配列にしてください。そうでない場合は、カーソルを使用してローを一度に 1 つずつ処理する必要があります。

参照 declare cursor

set connection

説明 指定した既存の接続を現在の接続に変更します。

構文 set connection {*connection_name* | DEFAULT} end-exec

パラメータ

connection_name
現在の接続に変更する既存の接続の名前です。

default
名前が指定されていないデフォルトの接続を現在の接続に変更することを指定します。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     MYID     PIC X(33).
     EXEC SQL END DECLARE SECTION END-EXEC.
          ...
     EXEC SQL CONNECT "user1" AT connect1 USING "SERVER1" END-EXEC.
     EXEC SQL CONNECT "user2" AT connect2 USING "SERVER2" END-EXEC.

* The next statement executes on connect2, because that was the
* last connection made.

     EXEC SQL SELECT user_name() INTO :MYID END-EXEC.

     DISPLAY "The user connected to SERVER2 is: ",MYID.
```

```
* Explicitly set the connection to now use to connect1.

     EXEC SQL SET CONNECTION connect1 END-EXEC.

* The following statement will execute on connect1.

     EXEC SQL SELECT user_name() INTO :MYID END-EXEC.

     DISPLAY "The user connected to SERVER1 is: ",MYID.
```

使用法

- set connection 文は、exec sql 文の at 句が前に入っている文を除き、後続のすべての SQL 文に対して現在の接続を指定します。
- set connection 文を再使用することによって現在の接続とは別の接続を選択するまで、set connection 文は有効になります。

参照

at connection_name、connect

set descriptor

説明

SQL 記述子へデータを挿入または更新します。

使用可能な SQL 記述子のデータ型のリストについては、表 9-5 (161 ページ) を参照してください。

構文

exec sql set descriptor *descriptor_name*
{count = *host_variable*} |
{value *item_number* {*item_name* =
:host_variable}[,...]end-exec

パラメータ

descriptor_name
準備文内の動的パラメータ・マーカについての情報が入っている SQL 記述子の名前です。

count
記述される動的パラメータの指定回数です。

host_variable
declare セクション内で定義されたホスト変数です。

item_number
動的パラメータ・マーカまたは選択カラムの *n* 番目のオカレンスを表します。

item_name
動的パラメータ・マーカまたは select リスト・カラムの属性情報を表します。表 9-4 に、*item_name* の値をリストします。

表 9-4: item_name の値

値	説明
data	指定した SQL 記述子と対応する動的パラメータ・マーカまたはターゲットの値。インジケータが負の値である場合、このフィールドは未定義である。
length	指定した SQL 記述子のターゲットの動的パラメータ・マーカの文字単位での長さ。
precision	CS_NUMERIC 変数の精度の有効桁数を指定する整数。
scale	CS_NUMERIC 変数の小数点桁数を指定する整数。
type	ロー内のこのカラム (項目番号) のデータ型。値については、表 9-5 (161 ページ) を参照。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
          01     TITLE-ID     PIC X(6).
          01     SALES1       PIC S9(9).
          01     SALES2       PIC S9(9).
          01     ROYALTY      PIC S9(9) COMP.
      EXEC SQL END DECLARE SECTION END-EXEC.
          ...
      EXEC SQL ALLOCATE DESCRIPTOR roy_desc WITH MAX 3 END-EXEC.
      EXEC SQL PREPARE  getroylty FROM "SELECT royalty FROM roysched
         WHERE title_id = ?and lorange <= ?AND hirange > ?"
      END-EXEC.

      MOVE "BU1032" TO TITLE-ID.
      MOVE 1000     TO SALES1.
      MOVE 10           TO SALES2.

      EXEC SQL SET DESCRIPTOR roy_desc VALUE 1 DATA = :TITLE-ID END-EXEC.
      EXEC SQL SET DESCRIPTOR roy_desc VALUE 2 DATA = :SALES1   END-EXEC.
      EXEC SQL SET DESCRIPTOR roy_desc VALUE 3 DATA = :SALES2   END-EXEC.

      EXEC SQL EXECUTE getroylty INTO :ROYALTY USING SQL
                DESCRIPTOR roy_desc END-EXEC.

      DISPLAY "ROYALTY = ", ROYALTY.
```

使用法

Embedded SQL プログラムは、属性および値の情報を Client-Library に渡します。Client-Library は、プログラムが文を実行する要求を発行するまで、指定された SQL 記述子にデータを保持します。

参照

allocate descriptor、describe input、describe output、execute、fetch、get descriptor、open (**動的カーソル**)

update

説明

テーブルのロー内のデータを変更します。

構文

exec sql [at *connection_name*] update *table_name*
 set [*table_name*]
  column_name1 = {*expression1*
   | NULL | (*select_statement*)}
  [, *column_name2* =
  {*expression2* | NULL
   | (*select_statement*)}]…
 [from *table_name*
  [, *table_name*]…
 [where {*search_conditions* | current of *cursor_name*}]
 end-exec

パラメータ

table_name

テーブルまたはビューの名前です。この名前を指定する場合、Transact-SQL の update 文に対して有効であればどのようなフォーマットでも使用できます。

例

```
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
               01     STORE-NAME        PIC X(40).
               01     DISC-TYPE         PIC X(40).
               01     LOWQTY            PIC S9(9) COMP.
               01     HIGHQTY           PIC S9(9) COMP.
               01     DISCOUNT          PIC S9(9) COMP.
          EXEC SQL END DECLARE SECTION END-EXEC.
               ...
          EXEC SQL DECLARE upd_cursor CURSOR FOR
               SELECT s.stor_name, d.discounttype, d.lowqty,
                      d.highqty  , d.discount
               FROM   stores    s, discounts d
               WHERE  s.stor_id  = d.stor_id END-EXEC.

          EXEC SQL OPEN upd_cursor END-EXEC.
          PERFORM FETCH-LOOP UNTIL SQLCODE = 100.
          EXEC SQL CLOSE upd_cursor END-EXEC.
          EXEC SQL DEALLOCATE CURSOR upd_cursor END-EXEC.
          EXEC SQL COMMIT WORK END-EXEC.
               ...
          FETCH-LOOP.
     EXEC SQL FETCH upd_cursor INTO :STORE-NAME, :DISC-TYPE,:LOWQTY      ,
    :HIGHQTY,:DISCOUNT  END-EXEC.
               IF SQLCODE = 100
                    DISPLAY "NO MORE RECORDS TO FETCH.END OF PROGRAM RUN."
               ELSE
          DISPLAY "NEW DISCOUNT : "
                    ACCEPT DISCOUNT
                    EXEC SQL UPDATE discounts
                        SET discount = :DISCOUNT
                         WHERE CURRENT OF upd_cursor END-EXEC
               END-IF.
          END-FETCH-LOOP.
```

使用法

- このリファレンス・ページでは、主に Transact-SQL の update 文を Embedded SQL で使用した場合の相違点について説明します。『ASE リファレンス・マニュアル』を参照してください。
- ホスト変数は、式または where 句内のどこにでも指定できます。
- where 句を使用して、テーブル内の選択したローを更新できます。テーブル内のすべてのローを更新するには、where 句を省略してください。オープンしているカーソルの現在のローを更新するには、where current of *cursor_name* を使用してください。
- where current of *cursor_name* を指定する場合、open cursor 文内で指定した接続でその文を実行してください。at *connection_name* 句を使用する場合、その句は open cursor 文と一致していなければなりません。

参照

close、delete cursor、fetch、open、prepare

whenever

説明

実行可能な SQL 文が、指定した条件になると実行する動作を指定します。

構文

```
exec sql whenever {sqlerror | not found | sqlwarning}
 {continue | go to label | goto label |
 stop | call routine_name [args]} end-exec
```

パラメータ

sqlerror

Adaptive Server から Embedded SQL プログラムに返される構文エラーなど、エラーを検出したときに行う動作を指定します。

not found

fetch 文または select into 文がデータを取得しない場合や、検索された update 文または delete 文がローに影響を与えない場合に行う動作を指定します。

sqlwarning

警告を受け取ったとき (たとえば、文字列がトランケートされたとき) に行う動作を指定します。

continue

指定した条件になっても何の動作も行いません。

go to | goto

指定した *label* のプログラム文に制御を渡します。

label

C ラベルなどのホスト言語文ラベルです。

stop

指定した条件が発生したときに Embedded SQL プログラムを終了します。

call
ユーザ定義の関数またはサブルーチンなどのプログラム内の呼び出し可能ルーチンに制御を渡します。

routine_name
呼び出しのできるホスト言語ルーチンです。このルーチンは、whenever 文を含んでいるソース・ファイルから呼び出し可能でなければなりません。Embedded SQL プログラムをコンパイルするには、そのルーチンを外部ルーチンとして宣言する必要があります。

args
ホスト言語のパラメータ渡し規約を使用して、呼び出し可能なルーチンに渡す 1 つ以上の引数です。引数は、ホスト言語が許可するホスト変数、リテラル、または式のリストです。個々の引数を区切るには、スペース文字を使用します。

例

```
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01        LNAME      PIC X(15).
       01        FNAME      PIC X(15).
       01        PHONE      PIC X(15).
  EXEC SQL END DECLARE SECTION END-EXEC.

  EXEC SQL WHENEVER SQLERROR PERFORM ERR-PARA END-EXEC.
  EXEC SQL WHENEVER SQLWARNING PERFORM WARN-PARA END-EXEC.
* If there are no more records to process from the fetch, stop the
* program.
  EXEC SQL WHENEVER NOT FOUND STOP END-EXEC.
        ...
  EXEC SQL DECLARE au_list CURSOR FOR
              SELECT au_lname, au_fname, phone
              FROM authors
              ORDER BY au_lname END-EXEC.

  EXEC SQL OPEN au_list END-EXEC.

  PERFORM FETCH-LOOP UNTIL SQLCODE = 100 END-EXEC.
  EXEC SQL CLOSE au_list END-EXEC.
        ...
  FETCH-LOOP.
        EXEC SQL FETCH au_list INTO
                    :LNAME, :FNAME, :PHONE END-EXEC
        DISPLAY "LAST NAME  : ",LNAME
        DISPLAY "FIRST NAME : ",FNAME
        DISPLAY "PHONE      : ",PHONE
  END-FETCH-LOOP.
```

```
 WARN-PARA.
      DISPLAY "Warning code is " SQLCODE.

      DISPLAY "Warning message is " SQLERRMC.
            ...
  WARN-PARA-END.
      EXIT.

  ERR-PARA.
*
* print the error code, the error message and the line number of
* the command that caused the error.
*

      DISPLAY "Error code is " SQLCODE.

      DISPLAY "Error message is " SQLERRMC.

      EXIT.
```

使用法

- whenever 文によって、Embedded SQL プログラムのプリコンパイラは実行可能な各 SQL 文のあとにコードを生成します。生成されたコードには、条件に対するテスト、およびホスト言語文または指定された動作を行う文が含まれます。
- Embedded SQL プリコンパイラは、ソース・ファイル内の whenever 文のあとの SQL 文に対して、コードを生成します。これには、同じソース・ファイル内で定義された、サブルーチン内の SQL 文を含みます。
- 前回の whenever 文をキャンセルするには、whenever…continue を使用します。continue の動作は、Embedded SQL プログラムのプリコンパイラに条件を無視するように命令します。無限ループを防ぐには、エラー・ハンドラ内で whenever…continue を使用してから、Embedded SQL 文を実行してください。
- whenever…go to *label* を使用する場合、*label* は実行の継続可能なロケーションを示さなければなりません。たとえば C では、whenever 文のスコープ内で実行可能な SQL 文を持っているルーチン内で、*label* を宣言しなければなりません。C の場合、goto 文は他の関数内で宣言されたラベルにジャンプすることはできません。
- プログラム内で whenever 文を使用し、SQLCA または SQLSTATE ステータス変数を宣言していない場合、Embedded SQL プログラムのプリコンパイラは SQLCODE 変数が使用されていると想定します。SQLCODE が宣言されていることを確認してください。そうでない場合、生成されたコードはコンパイルされません。

SQL 記述子のコード

表 9-5 は、動的 SQL 文に使用される SQL 記述子を示します。Sybase での動的 SQL 値の使用方法は、ANSI/ISO 185-92 SQL-92 標準に準拠しています。対応する ANSI/ISO のマニュアルを参照してください。

表 *9-5: SQL* 記述子のデータ型のコード

ANSI SQL データ型	コード
bit	14
character	1
character varying	12
date、time	9
decimal	3
double precision	8
float	6
integer	4
numeric	2
real	7
smallint	5

Sybase 定義のデータ型	Client-Library コード
smalldatetime	-9
money	-10
smallmoney	-11
text	-3
image	-4
tinyint	-8
binary	-5
varbinary	-6
long binary	-7
longchar	-2

表 *9-6: SQL* 記述子の識別子値

値	説明
type	ロー内のこのカラム (項目番号) のデータ型。値については、表 9-5 (161 ページ) を参照。
length	指定した SQL 記述子のターゲットの動的パラメータ・マーカの文字単位での長さ。
returned_length	select カラム・リストからの char データ型の値の長さ。
precision	CS_NUMERIC 変数の精度の有効桁数を指定する整数。
scale	CS_NUMERIC 変数の小数点桁数を指定する整数。
nullable	動的パラメータ・マーカで null 値が許可される場合は 0 であり、null 値が許可されない場合は 1 である。
indicator	動的パラメータ・マーカまたはターゲットと対応するインジケータ・パラメータの値。
data	指定した SQL 記述子と対応する動的パラメータ・マーカまたはターゲットの値。インジケータが負の値である場合、このフィールドは未定義である。
name	動的パラメータ・マーカについての情報が入っている、指定した SQL 記述子の名前。

第 10 章

Open Client/Server 設定ファイル

Open Client/Server 設定ファイルを使用すると、Open Client/Server アプリケーションを簡単に設定できます。デフォルトではこの設定ファイルは *ocs.cfg* という名前です。UNIX の場合は *$SYBASE/$SYBASE_OCS/config* ディレクトリにあります。Microsoft Windows の場合は、*%SYBASE%¥%SYBASE_OCS%¥ini* ディレクトリにあります。

Open Client/Server 設定ファイルの使用目的

Open Client/Server 設定ファイルを使用すると、すべての Open Client/Server アプリケーションの接続を一箇所で設定できます。この設定ファイルを使用すると、標準の設定を確立する作業と設定の変更を管理する作業が簡単になります。

設定機能へのアクセス

この機能は、initialize_application 文の 2 つの新しいコマンドライン・オプションを使用してアクセスできます。

- -x – これは外部設定のためのオプションです。アプリケーションはある名前を使用してアプリケーションを初期化する必要があります。Open Client/Server 設定ファイルには、このアプリケーション名を使用するセクションがあります。このセクションには、このアプリケーションに対して設定する必要があるすべてのプロパティを指定してください。-x オプションは、initialize_application 文とともに使用した場合にだけ有効になります。初期化を行わないで -x オプションを使用した場合は、設定ファイルのデフォルト・セクションがアクセスされます。

- -e – このオプションは、サーバ名を使用して設定できるようにします。initialize_application 文への呼び出しは必要ありません。サーバ名は、そのサーバ名によって定義されたセクションに設定されるプロパティを設定ファイル内で検索するためのキーとして使用されます。これによって、ユーザは特定の接続プロパティと接続名を対応させることができます。

注意 実行される最初の Embedded SQL 文が INITIALIZE_APPLICATION 文でない場合、外部設定プロパティは設定されません。実行される最初の Embedded SQL 文が INITIALIZE_APPLICATION 文である場合は、外部設定オプションが初期化に使用されます。

デフォルト設定

次は、デフォルト設定の Open Client/Server 設定ファイルです。必要に応じてこのファイルをカスタマイズできます。

```
[DEFAULT]


;This is the default section loaded by applications that use the
 ;external configuration feature, but which do not specify their
 ;own application name. Initially this section is empty.Defaults
 ;from all properties will be the same as earlier versions of
 ;Open Client libraries.


[ANSI_ESQL]


;This section defines configuration which an ANSI conforming
 ;Embedded SQL application should use to get ANSI-defined
 ;behavior from Adaptive Server Enterprises and Open Client libraries. This set
of
 ;configuration ;properties matches the set which earlier
 ;versions of Embedded SQL (version 10.0.x) automatically set for
 ;applications duringexecution of a CONNECT statement.



CS_CAP_RESPONSE=CS_RES_NOSTRIPBLANKS
 CS_EXTRA_INF=CS_TRUE
 CS_ANSI_BINDS=CS_TRUE
 CS_OPT_ANSINULL=CS_TRUE
 CS_OPT_ANSIPERM=CS_TRUE
```

```
CS_OPT_STR_RTRUNC=CS_TRUE
CS_OPT_ARITHABORT=CS_FALSE
CS_OPT_TRUNCIGNORE=CS_TRUE
CS_OPT_ISOLATION=CS_OPT_LEVEL3
CS_OPT_CHAINXACTS=CS_TRUE
CS_OPT_CURCLOSEONXACT=CS_TRUE
CS_OPT_QUOTED_IDENT=CS_TRUE
;End of default sections
```

Open Client/Server 設定ファイルの構文

Open Client/Server 設定ファイルの構文は、多少の違いはありますが、CS-Library でサポートされる Sybase ローカライゼーション・ファイルや設定ファイルの既存の構文とほぼ一致しています。

構文

- ; ― コメント行を示します。
- [section_name] ― セクション名は角カッコ ([]) で囲まれています。Open Client/Server 設定ファイルには DEFAULT と ANSI_ESQL という名前のセクションがあります。-x オプションを使用してコンパイルされたアプリケーションでは、アプリケーション名がセクション名として使用されます。-e オプションを使用してコンパイルされたアプリケーションの場合、そのサーバ名がセクション名に使用されます。複数のセクションで使用される設定が入っているセクションの名前は、どのような名前でも使用できます。次の例は、GENERIC という任意の名前のセクションと、そのセクションがその他のセクションにどのように指定されているかを示します。

```
[GENERIC]
 CS_OPT_ANSINULL=CS_TRUE


[APP_PAYROLL]
 include=GENERIC
 CS_CAP_RESPONSE=CS_RES_NOSTRIPBLANKS


[APP_HR]
 include=GENERIC
 CS_OPT_QUOTED_IDENT=CS_TRUE
```

- entry_name=entry_value
  - エントリ値には、整数や文字列など、どのような値でも指定できます。エントリ値の行が '¥'<newline> で終了する場合、そのエントリ値は次の行に続きます。
  - エントリ値の最初と最後にあるスペースはトリムされます。
  - エントリ値の最初または最後にスペースが必要な場合は、そのスペースを二重引用符で囲んでください。
  - 二重引用符で始まるエントリは、二重引用符で終了する必要があります。引用符で囲まれた文字列の中に連続する 2 つの二重引用符がある場合、それは値文字列の中の 1 つの二重引用符を表します。二重引用符内に改行文字が検出された場合は、文字どおりにその値の一部とみなされます。
  - エントリ名とセクション名は、アルファベット (大文字と小文字の両方)、0 ～ 9 の数字、区切り文字 (! " # $ % & ' ( ) * + , - . / : ; < > ? @ ¥ ^ _ ` { | } ~) から構成できます。

    角カッコ ([ ])、スペース、等号記号 (=) は使用できません。ただし、最初の文字は必ずアルファベットにしてください。
  - エントリ名とセクション名は大文字と小文字が区別されます。
  - Include=*earlier_section*

    セクションに include というエントリがある場合、以前に定義されたセクションの内容全体がこのセクション内にコピーされるとみなされます。つまり、以前のセクションで定義されたプロパティはこのセクションによって継承されます。

    インクルードされるセクションは、別のセクションにインクルードされる前に定義されている必要があるので注意してください。これによって、設定ファイルの解析を単一のパスで行うことができ、再帰的にインクルードされたディレクティブ (命令語) を検出する必要がなくなります。

    インクルードされたセクションが順に別のセクションをインクルードする場合、エントリ値の順序は、インクルードされたセクションの「入れ子の深い方から浅い方へ」の検索によって定義されます。

    セクションには、そのセクション自体への参照をインクルードすることはできません。つまり、以前に定義されたセクションをインクルードする必要があるので再帰は不可能です。また、定義されていないセクションをインクルードすることはできません。

あるセクションで定義されたすべての直接エントリ値は、別のセクションからインクルードされた可能性のある値を置き換えます。次の例では、CS_OPT_ANSINULL は APP.PAYROLL アプリケーションでは false に設定されます。include 文の位置は、この規則には影響を与えません。

```
[GENERIC]
 CS_OPT_ANSINULL=CS_TRUE


[APP_PAYROLL]
 CS_OPT_ANSINULL=CS_FALSE
 include=GENERIC
```

サンプル・プログラム

次のような例があると仮定します。Embedded SQL プログラムではカーソルを定義して、pubs2 データベースにある titles テーブルからローを取得します。WHERE 句は ANSI 標準ではない NULL チェック機能を使用します。IS NULL と IS NOT NULL は、Embedded SQL プログラムで使用されるデフォルトであり、これは ANSI 標準です。ただし、= NULL または != NULL を使用する Embedded SQL プログラムでは、ANSINULL 動作を OFF にして、代わりに Transact-SQL 構文を使用する必要がありますバージョン 11.1 より前の Embedded SQL の Transact-SQL 構文で NULL を比較する場合は、次の呼び出しを行う必要があります。

```
EXEC SQL set ansinull off END-EXEC.
```

次の例では Embedded SQL コードへの変更はありませんが、Open Client/Server 設定ファイルに適切なプロパティを設定することによって適切な動作を実行します。

以降の項に同じプログラムの 2 つのバージョンを示します。1 つは -e オプションを使用するバージョンであり、もう 1 つは -x オプションを使用するバージョンです。

Embedded SQL/COBOL サンプル プログラム

サンプル・プログラムを使用する前に、次の操作を実行します。

- IBM の場合、サンプル・プログラム構築用に提供されている Embedded SQL/COBOL makefile の SYBPLATFORM 環境変数を “rs6000” に設定します。
- Sun Solaris の場合、サンプル・プログラム構築用に提供されている Embedded SQL/COBOL makefile の SYBPLATFORM 環境変数を “sun_svr4” に設定します。
- HP の場合、サンプル・プログラム構築用に提供されている Embedded SQL/COBOL makefile の SYBPLATFORM 環境変数を “hpux” に設定します。
- HP Itanium の場合、サンプル・プログラム構築用に提供されている Embedded SQL/COBOL makefile の SYBPLATFORM 環境変数を “hpia” に設定します。
- Linux の場合、サンプル・プログラム構築用に提供されている Embedded SQL/COBOL makefile の SYBPLATFORM 環境変数を “linux” に設定します。

-x オプションを使用する Embedded SQL プログラム

```
* ocs_ex.pco

 * Description  :
 * This program declares a cursor which retrieves rows from
 * the 'titles' table based on condition checking for NULLS
 * in the NON-ANSI style ( CS_OPT_ANSINULL = CS_FALSE ).
 * The program will be compiled using the -x option which will
 * use an external configuration file (ocs.cfg) based on the
 * name of the application.The name of the application is
 * defined at the time of INITIALIZING the application.
 *
 *
 * Notes :Copy the file ocs.cfg in this directory to the $SYBASE direc-
 *         tory or add the entries from the section TEST1 in this file
 *         to your existing ocs.cfg file in the $SYBASE directory.
 *         Compile the program using the pre-processor flag -x.
 *         See the attached ocs.cfg file for details on the properties
 *         being set.

      EXEC SQL INCLUDE SQLCA END-EXEC.

      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
                     ...
    01    TITLE-ID          PIC X(6).
    01    PRICE             PIC X(30).
      EXEC SQL END DECLARE SECTION END-EXEC.
                     ...
```

```
      EXEC SQL INITIALIZE_APPLICATION APPLICATION_NAME
                 = "TEST1" END-EXEC.

      EXEC SQL CONNECT :UID IDENTIFIED BY :PASS END-EXEC.
      EXEC SQL USE pubs2 END-EXEC.

 * Declare and open the cursor for select
      EXEC SQL DECLARE title_list CURSOR FOR
                    SELECT title_id, price FROM titles
                     WHERE price != NULL END-EXEC.

      EXEC SQL OPEN title_list END-EXEC.

 * Fetch the data into host variables.
      PERFORM FETCH-LOOP UNTIL  SQLCODE = 100.

                    ...

      EXEC SQL CLOSE title_list END-EXEC.
      EXEC SQL DEALLOCATE CURSOR title_list END-EXEC.

      STOP RUN.

  FETCH-LOOP.


     EXEC SQL FETCH title list INTO
                    :TITLE-ID,
                    :PRICE END-EXEC.
           ...


     END-IF.
```

注意　makefile で設定するプリコンパイラ・オプション：cobpre -x

次は、前述のプログラムの設定ファイルの例です。

```
[DEFAULT]
 ;

 [TEST1]
 ;This is name of the application set by INITIALIZE_APPLICATION. ;Therefore this
is the section that will be referred to a runtime.

 CS_OPT_ANSINULL=CS_FALSE

 ;The above option will enable comparisons of nulls in the NON-ANSI
 ;style.
```

-e オプションを使用する同じ Embedded SQL プログラム

```
* Program name: ocs_test.cp
 *
 * Description  : This program declares a cursor that retrieves rows
 * from the 'titles' table based on condition checking for NULLS
 * in the NON-ANSI style.
 * The program will be compiled using the -e option, which will
 * use the server name that the application connects to, as the
 * corresponding section to look up in the configuration file.
 *

      EXEC SQL INCLUDE SQLCA END-EXEC.

      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
                    ...
    01    TITLE-ID          PIC X(6).
    01    PRICE             PIC X(30).
      EXEC SQL END DECLARE SECTION END-EXEC.
                    ...

      EXEC SQL CONNECT :UID IDENTIFIED BY :PASS END-EXEC.
      EXEC SQL USE pubs2 END-EXEC.

 * Declare and open the cursor for select
      EXEC SQL DECLARE title_list CURSOR FOR
                    SELECT title_id, price FROM titles
                     WHERE price != NULL END-EXEC.

      EXEC SQL OPEN title_list END-EXEC.

 * Fetch the data into host variables.
      PERFORM FETCH-LOOP UNTIL  SQLCODE = 100.

                    ...

      EXEC SQL CLOSE title_list END-EXEC.
      EXEC SQL DEALLOCATE CURSOR title_list END-EXEC.

      STOP RUN.

  FETCH-LOOP.


     EXEC SQL FETCH title list INTO
                    :TITLE-ID,
                    :PRICE END-EXEC.
           ...


     END-IF.
```

注意 makefile で設定するプリコンパイラ・オプション：cobpre -e

次は、前述のプログラムの設定ファイルの例です。

```
[DEFAULT]
 ;

 [SYBASE]
 ;This is name of the server that the application connect to. Therefore
 ;this is the section that will be referred to a runtime.
 ;
 CS_OPT_ANSINULL=CS_FALSE
 ;The above option will enable comparisons of nulls in the NON-ANSI
 ;style.
```

上の設定ファイルはかなり簡略化してあります。一般的な Open Client/Server 設定ファイルは次のような形式になります。

```
[DEFAULT]
 ;
 [ANSI_ESQL]
 CS_CAP_RESPONSE=CS_RES_NOSTRIPBLANKS
 CS_EXTRA_INF=CS_TRUE
 CS_ANSI_BINDS=CS_TRUE
 CS_OPT_ANSINULL=CS_TRUE
 CS_OPT_ANSIPERM=CS_TRUE
 CS_OPT_STR_RTRUNC=CS_TRUE
 CS_OPT_ARITHABORT=CS_FALSE
 CS_OPT_TRUNCIGNORE=CS_TRUE
 CS_OPT_ISOLATION=CS_OPT_LEVEL3
 CS_OPT_CHAINXACTS=CS_TRUE
 CS_OPT_CURCLOSEONXACT=CS_TRUE
 CS_OPT_QUOTED_IDENT=CS_TRUE
 ;
 ;The following is a sample section showing how to alter standard
 ;configuration:
 ;
 [RELEVANT_SECION_NAME]
 ;
 ;Use most of the ANSI properties defined above,
 ;
 include=ANSI_ESQL

 ;but override some default properties

 CS_OPT_ANSINULL=CS_TRUE    ; enable non-ansi style null comparisons
 CS_OPT_CHAINXACTS=CS_FALSE ; run in autocommit mode
```

付 録 A

プリコンパイラの警告とエラー・メッセージ

Embedded SQL プリコンパイラは、この付録の表に示す情報メッセージ、警告メッセージ、エラー・メッセージを生成します。

表のコードについて

表 A-1 から A-9 の「重大度」カラムの意味は、次のとおりです。

- 情報メッセージ ― エラーも警告も検出されず、プリコンパイラの作業は成功しました。このメッセージは、情報を提供するだけのものです。
- 警告 ― 致命的でないエラーが検出されましたが、プログラムはプリコンパイルされました。
- 重大なエラー ― エラーが発生し、コードの生成は行われませんでした。プリコンパイルは失敗しました。
- 致命的なエラー ― プリコンパイラがリカバリできない重大なエラーが発生しました。これ以上ファイルを処理することができません。プリコンパイラは終了します。

表 *A-1:* コマンド・ライン・オプション・メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_COMPAT_INFO	互換モードが指定されました。	情報メッセージ	対処不要。
M_DUPOPT	重複したコマンド・ライン・オプションが指定されました。	重大なエラー	コマンド・ラインでは、オプションを重複して指定してはならない。間違って指定した方のオプションを削除する。
M_EXCFG_OVERRIDE	外部設定スイッチ *value* が指定されていないため、スイッチ *value* による影響はありません。	警告	外部設定ファイルを使用している場合、コマンド・ラインで設定されているオプションを上書きしている可能性がある。設定オプションの 1 つを選択する。
M_INVALID_COMPAT	認識できない互換モードが指定されました。	情報メッセージ	対処不要。
M_INVALID_FILE_FMT	ファイル *value* の行 *value* に正しくない文字があります。	重大なエラー	入力ファイルの文字が正しいこと、および使用する文字セットの設定が正しいことを確認する。

メッセージ ID	メッセージ	重大度	対処法
M_INVALID_FIPLEVEL	指定した FIPS レベルが正しくありません。	重大なエラー	正しい値は、SQL-92E と SQL-89。
M_INVALID_SYNLEVEL	指定した構文チェック・レベルが正しくありません。	重大なエラー	正しい値は NONE、SYNTAX、SEMANTIC。
M_INVLD_HLANG	指定されたホスト言語が正しくありません。	重大なエラー	有効なオプションは、COB_MF1、COB_MF2、COB_RM1、COB_RM2、COB_LPI、COB_VAXVMS。
M_INVLD_OCLIB_VER	UI-The Open Client Client-Library version is invalid.	重大なエラー	正しいバージョン文字列は "CS_VERSION_110" 以降。
M_INVOPT	オプションが正しくありません。	重大なエラー	無効なオプション指定。正しい値に置き換える。
M_LABEL_SYNTAX	セキュリティ・ラベルが不適切に指定されています。正しいフォーマットは、'labelname=labelvalue' です。	重大なエラー	許可されている構文を使用する。
M_MSGINIT_FAIL	ローカライズされたエラー・メッセージの初期化のエラーです。	警告	Sybase のインストールが完了しているか、*locales.dat* ファイルに LANG 変数に対する有効なエントリがあるかどうかを確認する。
M_MULTI_IN_USE_DEF_OUT	複数の入力ファイルをプリコンパイルするときは出力 (リスティング、SQL、または言語) ファイル名を指定できません。	重大なエラー	コマンド・ラインから -G、-L、-O フラグをすべて削除するか、1 回にプリコンパイルするファイルを 1 つだけにする。
M_NO_INPUT_FILE	エラー : プリコンパイルする入力ファイルが指定されていません。	重大なエラー	プリコンパイルする入力ファイルを指定する。 **注意** このエラーは、オプションの引数を持つフラグ (ストアド・プロシージャを生成する -G など) を入力ファイル名の前に置くと発生することがある。これを解決するには、入力ファイル名の前に別のフラグを付ける。たとえば、cpre -G file.pc を cpre -G -C*compilername* に置き換える。
M_NO_PERSISTENT_COBOL	持続的に入力されるホスト変数を示すオプション -p は、使用できません。	情報	対処不要。
M_OPEN_INCLUDE	インクルード・ファイル *file* をオープンできません。	重大なエラー	指定したファイルがパスにないか、必要な読み込みパーミッションがない。-I フラグを使用してパスを指定するか、読み込みパーミッションを確認する。

メッセージ ID	メッセージ	重大度	対処法
M_OPEN_INPUT	入力ファイル *file* をオープンできません。	重大なエラー	指定したパスとファイル名が有効かどうかを確認する。ファイル名の拡張子を指定していない場合、プリコンパイラはデフォルトの拡張子を検索する。
M_OPEN_ISQL	ISQL ファイル *file* をオープンできません。	重大なエラー	isql ファイル名 (ストアド・プロシージャが記述されているファイル) が有効かどうかを確認する。ファイルを作成しているディレクトリに、書き込みパーミッションがあるかどうかを確認する。
M_OPEN_LIST	リスティング・ファイル *file* をオープンできません。	重大なエラー	リスティング・ファイル名が有効かどうかを確認する。ファイルを作成しているディレクトリに、書き込みパーミッションがあるかどうかを確認する。
M_OPEN_TARGT	ターゲット・ファイル *file* をオープンできません。	重大なエラー	出力ファイル名が有効かどうかを確認する。ファイルを作成しているディレクトリに、書き込みパーミッションがあるかどうかを確認する。
M_OPT_MUST_BE_PROVIDED	オプション *value* を指定しなければいけません。	重大なエラー	オプションの値を設定する。
M_OPT_REINIT	警告：*value* スイッチが複数回初期化されました。	警告	指定したスイッチが複数回初期化されている。2 回目以降の値は無視される。
M_PATH_OFL	エラー : インクルード・ファイルの最大許容パスは 64 です。(オーバーフロー)	重大なエラー	コマンド・ラインで指定できるパスの最大数を超えた。*INCLUDE* ファイルをフェッチするディレクトリの数を減らす。
M_STATIC_HV_CNAME	統計カーソル名にホスト変数 *line* は使用できません。	重大なエラー	ホスト変数を SQL 識別子に置き換える。
M_UNBALANCED_DQ	区切られた識別子に対になっていない引用符があります。	重大なエラー	左右の引用符の数を一致させる。
M_VMS_NO_PERSISTENT_ COBOL	persistent オプションは使用できません	情報	

表 *A-2:* 第 *1* パス・パーサ・メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_64BIT_INT	警告：64 ビットの整数ホスト変数はサポートされていません。行 *value*	警告	他のホスト変数データ型 (浮動小数点、数値、32 ビット整数) を使用する。必要なら、ホスト変数と 64 ビット・プログラム変数間で値をコピーする。
M_BLOCK_ERROR	*value* の行にノンマッチング・ブロック・ターミネータがあります。*value*	重大なエラー	プログラムの構文を修正する。
M_COB_INC_SQLDA	エラー : INCLUDE SQLDA 文は、ESQL/COBOL では無効です。	重大なエラー	SYBSETSQLDA を使用する。「SYBSETSQLDA の使用」(81 ページ) を参照。
M_CONST_FETCH	エラー : CONST 記憶クラスの変数 *value* にフェッチしようとしました。	重大なエラー	定数データ型へのフェッチはできない。値をフェッチするには、宣言から定数修飾子を削除する。
M_DUP_HV	*file* の行 *line* に重複したホスト変数があります。	重大なエラー	同じブロック内で、すでに同じ名前のホスト変数が宣言されている。表示されたブロックの各変数がユニークであるかどうかを確認する。
M_DUP_STRUNION	*file* の行 *line* に重複した構造体／共用体があります。	重大なエラー	同じブロック内で、すでに同じ名前の構造体が宣言されている。表示されたブロックの各変数がユニークであるかどうかを確認する。
M_IDENT_OR_STRINGVAR	エラー : 項目は、SQL 識別子か文字列型変数でなければなりません。	重大なエラー	接続、カーソル、または文名が、文字列型または SQL 識別子になっていることを確認する。
M_ILL_LITERAL_USAGE	エラー : OUTPUT 修飾子を持つストアド・プロシージャにリテラル・パラメータを使用することは、無効です。	重大なエラー	リテラルを、ストアド・プロシージャに対する OUTPUT パラメータとして使用しないようにする。
M_ILL_PARAM_MODE	エラー: ファイル *file* 行 *line* でのリモート・プロシージャ・コールに呼び出しモードが混在しています。	重大なエラー	名前または位置を使って渡した引数を使用して、ストアド・プロシージャを呼び出す。これらのモードを同じ呼び出しで同時に使用すると無効になる。
M_INDICVAR	エラー : 項目は、インジケータ・タイプ変数でなければなりません。	重大なエラー	short integer を使用する。
M_INTVAR	エラー : 項目は、整数タイプ変数でなければなりません。	重大なエラー	整数を使用する。
M_INVLD_HV_BT	タイプ *value* の COBOL ホスト変数 *value* は、サポートされていません。	重大なエラー	ホスト変数のデータ型を確認する。サポートされていない型が検出された。

メッセージ ID	メッセージ	重大度	対処法
M_MISMATCHED_QUOTES	エラー: 16 進リテラル *value* にある引用符が正しくありません。	重大なエラー	左右の引用符の数を一致させる。
M_MULTIDIM_ARRAY	エラー : 行 *line*。多次元配列の変数は、サポートされていません。	重大なエラー	多次元の配列はサポートしていない。*m x n* 配列を分割して、それぞれ *n* 個の要素からなる *m* 配列にする。
M_MULTI_RESULTS	エラー: 行 *line* の Embedded Query が複数の結果セットを返します。	重大なエラー	クエリを複数のクエリに分割し、各クエリが 1 つの結果セットを返すようにする。または、テンポラリ・テーブルをすべての値で満たすようにクエリを書き直し、テンポラリ・テーブルから選択することによって、1 つの結果セットを返す。
M_NODCL_NONANSI	警告：非 ANSI モードで SQLCODE、SQLSTATE、または SQLCA のどれも宣言されていません。	警告	非 ANSI モードでは、SQLCA と SQLCODE のいずれか、または両方を宣言する。プログラム内の Embedded SQL 文すべてをスコープに適用できるかどうかを確認する。
M_NOLITERAL	エラー : 項目が、引用符で囲まれていない名前の可能性があります。	重大なエラー	引用符付きの名前またはホスト変数を使用する。
M_NOSQUOTE	エラー : 項目は、一重引用符で囲まれていてはいけません。二重引用符を使用してください。	重大なエラー	二重引用符を使用する。
M_NOT_AT_ABLE	“at” 句を使用できない文タイプで “at” 句が使用されています。これは、文 *value* で発生しました。	重大なエラー	指定した文から at 句を削除する。
M_NUMBER_OR_INDICVAR	エラー : 項目は、整数またはインジケータ・タイプ変数でなければなりません。	重大なエラー	リテラル integer、short integer、または CS_SMALLINT を使用する。
M_NUMBER_OR_INTVAR	エラー : 項目は、整定数か整数型変数でなければなりません。	重大なエラー	未使用。動的 SQL 文内のあるフィールド (MAX, Value *n* など) が、integer 型でも整数定数でもない場合に、このエラーを発生させるために使用する可能性がある。
M_PARAM_RESULTS	エラー: 行 *line* の Embedded Query が予期しないパラメータ結果セットを返します。	重大なエラー	オプションのサーバ構文の確認中にのみ発生する。クエリがパラメータを返している原因を調べ、クエリを書き直す。

メッセージ ID	メッセージ	重大度	対処法
M_PASS1_ERR	ファイル *file*：Pass 1 の構文エラー：Pass 2 は処理されていません。	情報メッセージ	パス 1 で発生したエラーのため、プリコンパイルがアボートされた。パス 1 のエラーを修正し、処理を進める。
M_PTR_IN_DEC_SEC	警告：宣言部分でのポインタは、サポートされていません。	警告	なし
M_QSTRING_OR_STRINGVAR	エラー : 項目は、引用文字列か文字列型変数でなければなりません。	重大なエラー	サーバ名、ユーザ名、パスワードが、二重引用符で囲まれた文字列または文字列データ型になっているかどうかを確認する。
M_SCALAR_CHAR	エラー : *line* 行で、配列のない文字列変数 *value* がホスト変数として不正に使用されています。	重大なエラー	文字配列を使用する。
M_SQLCA_IGNR	警告 : SQLCODE と SQLCA の両方が宣言されています。SQLCA は無視されます。	警告	2 つの宣言のうちの 1 つを削除する。
M_SQLCA_WARN	警告：ANSI モードで、INCLUDE SQLCA がありました。SQLCA は無視されます。	警告	なし
M_SQLCODE_UNDCL	警告：ANSI モードで、SQLCODE が宣言されていません。	警告	SQLCODE を宣言する。
M_STATE_CODE	警告：SQLSTATE と SQLCODE の両方が宣言されています。SQLCODE は無視されます。	警告	2 つの宣言のうちの 1 つを削除する。
M_STATE_SQLCA	警告 : SQLSTATE と SQLCA の両方が宣言されています。SQLCA は無視されます。	警告	2 つの宣言のうちの 1 つを削除する。
M_STATUS_RESULTS	エラー: 行 *line* の Embedded Query が予期しないステータス結果セットを返します。	重大なエラー	オプションのサーバ構文の確認中にのみ発生する。クエリがステータス結果を返している原因を調べ、クエリを書き直す。
M_STICKY_AUTOVAR	警告：*line* 行で、スティッキ・バインドとともに自動変数 *value* が使用されています。これにより、実行時に正しくない結果またはエラーが発生する可能性があります。	警告	この場合、使用しているプログラムの論理がエラーを許可しないことを確認する。または、静的変数またはグローバル変数を使用する。
M_STICKY_REGVAR	エラー :*line* 行で、スティッキ・バインドとともにレジスタ変数 *value* を使用できません。	重大なエラー	レジスタ修飾子を削除する。

メッセージ ID	メッセージ	重大度	対処法
M_STRUCT_NOTFOUND	*file* の行 *line* のスコープに構造体／共用体定義がありません。	重大なエラー	構造体や共用体の定義が指定された行のスコープ内であるかどうかを確認する。
M_SYNTAX_PARSE	ファイル *file* の行 *line* の付近に構文エラーがあります。	重大なエラー	表示された行番号の Embedded SQL の文法に構文エラーがないかどうかを確認する。
M_UNBALANCED_DQ	区切られた識別子に対になっていない引用符があります。	重大なエラー	左右の引用符の数を一致させる。
M_UNDEF_ELM	エラー *value*：構造体／共用体の要素が不正です。	重大なエラー	構造体の指定された要素が構造体の定義に入っていない。定義を修正する。
M_UNDEF_HV	ホスト変数 *value* は定義されていません。	重大なエラー	適切な場所でホスト変数を定義する。
M_UNDEF_IV	インジケータ変数 *value* は定義されていません。	重大なエラー	適切な場所でインジケータ変数を定義する。
M_UNDEF_STR	エラー: 構造体 *value* は定義されていません。	重大なエラー	未定義の構造体が指定した行にある。適切なスコープ内で構造体を定義する。
M_UNSUP	*value* 機能は、このバージョンではサポートされていません。	致命的なエラー	この機能はサポートされていない。

表 *A-3:* 第 2 パス・パーサ・メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_CURSOR_RD	ファイル *file* の行 *line* で、カーソル *value* が再定義されています。	警告	同じ名前のカーソルがすでに宣言されている。違う名前を使用する。
M_HOSTVAR_MULTIBIND	警告：ホスト変数 *value* がバインド変数として 1 つの文に 2 度以上使用されました。	警告	1 つのフェッチ文内で複数回ホスト変数を使用してはならない。複数の結果を 1 つのロケーションへフェッチすることはできない。Client-Library は、最後にフェッチした値を変数に代入する。
M_INVTYPE_IV	インジケータ変数のタイプが正しくありません。	重大なエラー	インジケータ変数は、CS_SMALLINT 型または INDICATOR 型にする。
M_PARSE_INTERNAL	行 *line* で内部パーサ・エラーです。サイベースにお問い合わせください。	致命的なエラー	この内部一貫性パーサ・エラーが発生した場合は、ただちに Sybase 製品の保守契約を結んでいるサポート・センタに連絡する。
M_SQLCANF	'INCLUDE SQLCA' 文が見つかりません。	警告	文を追加する。
M_TAB_IN_LIT	警告：引用符で囲まれた文字列の TAB 文字がスペースに変換されました。(この警告は、1 度しか表示されません。)	警告	これが問題の場合、手動で使用しているクエリ内の引用符付きの <tabs> をスペースに拡張する。
M_WHEN_ERROR	SQL 文 'WHENEVER SQLERROR' がありません。	警告	WHENEVER SQLERROR 文を追加するか、コマンド・ライン・オプションを使用して、警告と INTO メッセージを出さないようにする (『Open Client/Server プログラマーズ・ガイド補足』参照)。
M_WHEN_NF	SQL 文 'WHENEVER NOT FOUND' がありません。	警告	WHENEVER NOT FOUND 文を追加するか、コマンド・ライン・オプションを使用して、警告と INTO メッセージを出さないようにする (『Open Client/Server プログラマーズ・ガイド補足』参照)。
M_WHEN_WARN	SQL 文 'WHENEVER NOT FOUND' がありません。	警告	WHENEVER WARNING 文を追加するか、コマンド・ライン・オプションを使用して、警告と INTO メッセージを出さないようにする (『Open Client/Server プログラマーズ・ガイド補足』参照)。

表 *A-4:* コード生成メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_INCLUDE_PATHLEN	インクルードまたはコピーされたパスが長すぎます。生成されたファイル名 *value* からパスを取り除いてください。	警告	リンクを使用するか、ファイルを短いパスに移動する。
M_WRITE_ISQL	ISQL ファイルに書き込みできません。リターン・コード : *value*	致命的なエラー	isql ファイルとディレクトリに対する作成および書き込みパーミッションを確認する。また、ファイル・システムが満杯でないことを確認する。
M_WRITE_TARGT	ターゲット・ファイルに書き込みできません。リターン・コード : *value*	致命的なエラー	ターゲット・ファイルに書き込みできません。プリコンパイラがターゲット・ファイルを生成するディレクトリに対するファイルの作成および書き込みパーミッションがあるかどうかを確認する。また、ファイル・システムが満杯でないことを確認する。

表 *A-5: FIPS* フラグ・メッセージ

メッセージ ID	メッセージ	重大度	ANSI 拡張機能
M_FIPS_ARRAY	FIPS-FLAGGER 警告 : ANSI 拡張子ポインタ型 (行 *line*)	情報メッセージ	配列。ANSI 標準に準拠する必要がある場合、一切の FIPS メッセージでこの機能を使用してはならない。
M_FIPS_DATAINIT	FIPS-FLAGGER 警告 : ANSI 拡張子データ初期化 (行 *line*)	情報	データ初期化。
M_FIPS_GPITEM	FIPS-FLAGGER 警告 : ANSI 拡張子グループ項目構文 (行 *line*)	情報	
M_FIPS_HASHDEF	FIPS-FLAGGER 警告 : ANSI 拡張機能 "#DEFINE" *line*。	情報	DECLARE セクション内で #DEFIN を使用している。
M_FIPS_LABEL	FIPS-FLAGGER 警告 : ANSI 拡張、WHENEVER 句内の ':' とラベル	情報メッセージ	WHENEVER 句内でラベルに ":" を許可している。
M_FIPS_POINTER	FIPS-FLAGGER 警告 : ANSI 拡張子ポインタ型 (行 *line*)	情報メッセージ	POINTER 型。
M_FIPS_SQLDA	FIPS-FLAGGER 警告 : ANSI 拡張子 sqlda.(行 *line*)	情報メッセージ	SQLDA 構造体。
M_FIPS_STMT	FIPS-FLAGGER 警告 : ANSI 拡張子文 (行 *line*)	情報メッセージ	この行にある文は拡張機能である。
M_FIPS_SYBTYPE	FIPS-FLAGGER 警告 : ANSI 拡張子 Sybase SQL 型 (行 *line*)	情報メッセージ	Sybase 固有のデータ型である。

メッセージ ID	メッセージ	重大度	ANSI 拡張機能
M_FIPS_TYPE	FIPS-FLAGGER 警告 : ANSI 拡張子データ型 (行 *line*)	情報メッセージ	指定された構文は、ANSI 準拠ではない。
M_FIPS_TYPEDEF	FIPS-FLAGGER 警告 : ANSI 拡張機能 TYPEDEF *line*	情報メッセージ	TYPEDEF。
M_FIPS_VOID	FIPS-FLAGGER 警告 : ANSI 拡張子 void 型 (行 *line*)	情報メッセージ	VOID 型。

表 *A-6:* 内部エラー・メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_ALC_MEMORY	メモリのブロックを割り当てできません。	致命的なエラー	システム・リソースを確認する。
M_FILE_STACK_OVFL	ファイル・スタック・オーバーフロー : 最大許容ネスティングは *value* です。	致命的なエラー	ネストした INCLUDE 文の処理中に、ファイル・スタックがオーバフローした。ネストの深さは、最大値 32 を超えてはならない。
M_INTERNAL_ERROR	致命的な内部エラーがファイル *file* の行 *line* で発生しました。引数不一致エラーです。サイベースにお問い合わせください。	致命的なエラー	内部エラー。Sybase の担当者に連絡する。

表 *A-7:* プラットフォームと言語メッセージ

メッセージ ID	メッセージ	重大度	対処法
M_LONGLINE	印刷している行が長すぎて中断できません。	警告	出力される行を短くする。

表 *A-8: Sybase/Client-Library* のメッセージ

メッセージ ID	メッセージ	重大度	対処法
M_COLMCNT	*bind variable count* のバインド・カウントと *result set* のカラム・カウントは矛盾します。	警告	戻りカラムの数が、バインド変数のデータ型や数とともに戻された結果のカラム数と異なっている。
M_COLVARLM	ホスト変数 *name* の長さ *value* が *value* のカラム長より小さいです。	警告	ホスト変数がフェッチされたカラムを保持できない。カラムの長さを確認し、それに従ってホスト変数の長さを調整する。
M_COLVARPS	ホスト変数 *name* の精度と位取り *value* が、カラムの精度 *value* および位取り *value* と異なっています。	警告	ホスト変数の精度と位取りがフェッチまたは挿入されたカラムと異なる。位取りと精度を適合させる。
M_COLVARTM	Open Client は、ホスト変数 *value* のタイプ *value* をタイプ *value* に変換できません。	警告	無効なタイプ。Open Client がデフォルトで変換できないときは、cs_convert を使用する。

メッセージ ID	メッセージ	重大度	対処法
M_CTMSG	Client Library メッセージ : *value*	情報	なし。必要があれば、Sybase の保守契約を結んでいるサポート・センタに連絡する。
M_OCAPI	Open Client API の *value* を実行中にエラーが発生しました。エラー : *value*	警告	この警告が発生したコンテキストによっては、先へ進む前に修正が必要になることがある。
M_OPERSYS	オペレーティング・システム・エラー : Open Client API の実行中に *value* が発生しました。	警告	オペレーティング・システムでエラーが発生した。システム管理者に連絡する。
M_PRECLINE	行 *value* のクエリーをチェック中の警告	情報	クエリに問題がないかどうかを調べる。
M_SYBSERV	Sybase サーバ エラー。サーバ：*value* メッセージ：name	警告	エラーが発生したサーバへ送った文の構文を確認する。SQL 文を処理するときに Adaptive Server Enterprise 内ですべてのリソースが利用できることを確認する。

表 *A-9:* 実行時メッセージ

SQLSTATE コード	メッセージ	重大度	対処法
ZZ000	リカバリできないエラーが発生しました。	致命的なエラー	Sybase 製品の保守契約を結んでいるサポート・センタに連絡する。
ZA000	内部エラーが発生しました。	致命的なエラー	Sybase 製品の保守契約を結んでいるサポート・センタに連絡する。
ZD000	予期しない CS_COMPUTE_RESULT を受け取りました。	重大なエラー	Embedded SQL が計算結果を取得できない。エラーを返さないようにクエリを書き換える。
ZE000	予期しない CS_CURSOR_RESULT を受け取りました。	重大なエラー	CS_LIBRARY ルーチンから返された値が有効かどうかを確認する。詳細は、CS-Library のマニュアルを参照。
ZF000	予期しない CS_PARAM_RESULT を受け取りました。	重大なエラー	CS_LIBRARY ルーチンから返された値が有効かどうかを確認する。詳細は、CS-Library のマニュアルを参照。
ZG000	予期しない CS_ROW_RESULT を受け取りました。	重大なエラー	CS_LIBRARY ルーチンから返された値が有効かどうかを確認する。詳細は、CS-Library のマニュアルを参照。
ZB000	sqlca、SQLCODE または SQLSTATE にメッセージが返されません。	情報	情報メッセージ。対処不要。
ZC000	まだ接続が定義されていません。	重大なエラー	有効な connect 文を入力する。

SQLSTATE コード	メッセージ	重大度	対処法
ZH000	予想外の CS_STATUS_RESULT を受信しました。	重大なエラー	CS_LIBRARY ルーチンから返された値が有効かどうかを確認する。詳細は、CS-Library のマニュアルを参照。
ZI000	予期しない CS_DESCRIBE_RESULT を受け取りました。	重大なエラー	CS_LIBRARY ルーチンから返された値が有効かどうかを確認する。詳細は、CS-Library のマニュアルを参照。
22005	データ例外 -- 項目記述子の割り当てエラー - タイプ	重大なエラー	正しい記述子型を入力する。
ZJ000	メモリ割り当てに失敗しました。	重大なエラー	この作業を行うにはメモリ不足である。
ZK000	Adaptive Server Enterprise のバージョンは 10 以上でなければなりません。	重大なエラー	Adaptive Server Enterprise 10.0 以降がインストールされていることを確認する。Adaptive Server Enterprise 10.0 以降がない場合、インストール担当者は Sybase 製品の保守契約を結んでいるサポート・センタに連絡する。
ZM000	クライアント・ライブラリ初期化時のエラー	重大なエラー	*$SYBASE* の設定を確認する。
ZN000	mutex の取得時または解放時のエラー	重大なエラー	使用されていない。
08002	接続名が使用中です。	重大なエラー	使用しているプログラムの論理を確認する。オープンしている接続を再度オープンしていないか確認する。または、2 番目の接続に新しい名前を使用する。 **注意** DEFAULT 接続を 2 つ持つことはできない。

用語解説

Adaptive Server Enterprise

Sybase のクライアント／サーバ・アーキテクチャにおけるサーバ。Adaptive Server Enterprise は、複数のデータベースと複数のユーザを管理します。ディスク上にあるデータの実際のロケーションを監視し、論理データ記述から物理データ記憶領域へのマッピングを管理します。メモリ内のデータ・キャッシュとプロシージャ・キャッシュの保守も行います。

Client-Library

Open Client の一部で、クライアント・アプリケーションを記述するためのルーチンの集まり。Client-Library は、Sybase 製品ラインのカーソルや他の高度な機能を取り込むように設計されたライブラリです。

CS-Library

Client-Library と Server-Library のアプリケーションの両方で役立つユーティリティ・ルーチンの集まり。Open Client および Open Server の両方に含まれています。

DB-Library

Open Client の一部で、クライアント・アプリケーションを記述するためのルーチンの集まり。

FIPS

Federal Information Processing Standards (連邦情報処理標準) の略。FIPS フラグが有効なとき、Adaptive Server Enterprise および Embedded SQL プリコンパイラは、標準でない拡張された SQL 文を検出すると警告を発行します。

interfaces ファイル (interfaces file)

サーバ名をトランスポート・アドレスにマップするファイル。クライアント・アプリケーションが、サーバに接続するために ct_connect または dbopen を呼び出すと、Client-Library または DB-Library が interfaces ファイルからサーバのアドレスを検索します。ただし、すべてのプラットフォームが interfaces ファイルを使用するわけではありません。interfaces ファイルを使わないプラットフォームでは、別のメカニズムでクライアントにサーバ・アドレスを知らせます。

isql スクリプト・ファイル (isql script file)

Embedded SQL において、プリコンパイラが生成できる 3 つのファイルのうちの 1 つ。isql スクリプト・ファイルには、プリコンパイラが生成したストアド・プロシージャが含まれます。このストアド・プロシージャは、Transact-SQL で記述されます。

mutex

相互排他セマフォ。Open Server アプリケーションが、共有オブジェクトへ排他アクセスをするために使用する論理オブジェクトのことです。

NULL

NULL は、明示的に割り当てられた値を持ちません。NULL は 0 でもブランクでもありません。NULL の値は、他の値と比べて大きいとも、小さいとも、同じであるともみなされません。他の NULL との比較においても同様です。

Open Server
カスタム・サーバを作成するためのツールとインタフェースを提供する Sybase 製品。

Open Server アプリケーション (Open Server application)
Open Server で構築されたカスタム・サーバ。

Server-Library
Open Server アプリケーションの記述に使用するルーチンの集まり。

SQLCA
1. Embedded SQL アプリケーションにおいて、Adaptive Server Enterprise とアプリケーション・プログラム間の通信パスを提供する構造体。Adaptive Server Enterprise は、各 SQL 文を実行したあと、SQLCA 内のリターン・コードを格納します。

2. Client-Library アプリケーションにおいて、アプリケーションが、Client-Library およびサーバのエラー・メッセージと情報メッセージを取得するのに使用する構造体。

SQLCODE
1. Embedded SQL アプリケーションにおいて、Adaptive Server Enterprise とアプリケーション・プログラム間の通信パスを提供する構造体。Adaptive Server Enterprise は、各 SQL 文を実行したあと、SQLCODE 内のリターン・コードを格納します。SQLCODE は、独立して存在することも、SQLCA 構造体内の変数として存在することもできます。

2. Client-Library アプリケーションにおいて、アプリケーションが Client-Library およびサーバのエラー・メッセージと情報メッセージ・コードを取得するときに使用する構造体。

TDS
Sybase のクライアントとサーバが通信に使用するアプリケーション・レベルのプロトコル。TDS は、コマンドと結果を記述します。

Transact-SQL
データベース言語 SQL の機能拡張バージョン。アプリケーションは、Transact-SQL を使用して、Adaptive Server Enterprise と通信できます。

イベント (event)
Open Server アプリケーションに何らかの動作を行うよう要求するオカレンス。クライアント・コマンドおよび Open Server アプリケーション・コードの特定のコマンドは、イベントをトリガできます。イベントが発生すると、Open Server は、サーバ・アプリケーション・コード内の適切なイベント処理ルーチン、または適切なデフォルト・イベント・ハンドラのどちらかを呼び出します。

イベント・ハンドラ (event handler)
Open Server におけるイベントを処理するルーチン。Open Server アプリケーションは、Open Server が提供するデフォルト・ハンドラを使用することができます。また、カスタマイズしたイベント・ハンドラをインストールすることもできます。

インジケータ変数 (indicator variable)
他の変数の値やフェッチしたデータについての特別な条件を示す変数。

Embedded SQL ホスト変数とともに使用すると、インジケータ変数は、データベースの値が null である箇所を示します。

エラー・メッセージ (error message)
Open Client/Server 製品がエラー状態を検出したときに発行するメッセージ。

カーソル (cursor)
SQL 文に関連付けられた記号名。

Embedded SQL において、カーソルは複数ローのデータをホスト・プログラムに渡すデータ・セレクタです。このローの受け渡しは、一度に 1 つずつ行われます。

隠し構造体 (hidden structure)
内部が Open Client/Server プログラマに対して隠されている構造体。Open Client/Serever プログラマは、隠し構造体の割り付け、操作、割り付け解除を行うために、Open Client/Server ルーチンを使用しなければなりません。隠し構造体には、CS_CONTEXT 構造体などがあります。

拡張トランザクション (extended transaction)
Embedded SQL における、複数の Embedded SQL 文からなるトランザクション。

キー (key)
ローをユニークに識別するロー・データのサブセット。キー・データは、オープンされたカーソル内の「現在のロー」をユニークに記述します。

キーワード (keyword)
Transact-SQL または Embedded SQL で排他的に利用するように予約されているワードまたはフレーズ。「予約語」ともいいます。

機能 (capability)
クライアント／サーバ接続で許可されるクライアントの要求とサーバの応答を決定します。

クエリ (query)
1. データの検索要求。通常は select 文です。

2. データを操作する任意の SQL 文。

クライアント (client)
クライアント／サーバ・システムにおいて、サーバへ要求を送り、この要求に対する結果に対して処理を行う部分。

ゲートウェイ (gateway)
直接通信できないクライアントとサーバとの仲介として動作するアプリケーション。ゲートウェイ・アプリケーションは、クライアントとサーバの両方として動作します。クライアントからの要求をサーバに渡し、サーバからの結果をクライアントに返します。

結果変数 (result variable)
Embedded SQL において、select または fetch 文の結果を受け取る変数。

現在のロー (current row)
カーソルに関連して、カーソルが置かれているロー。fetch は、カーソルに対する現在のローを取得します。

公開された構造体 (exposed structure)
内部が Open Client/Server プログラマに公開されている構造体。Open Client/Server プログラマは、公開された構造体の宣言、操作、割り付け解除を直接行うことができます。公開された構造体には、CS_DATAFMT 構造体などがあります。

コード・セット (code set)
「文字セット *(character set)*」を参照してください。

コールバック・イベント (callback event)
Open Client と Open Server におけるコールバック・ルーチンをトリガするイベント。

コールバック・ルーチン (callback routine)
コールバック・イベントと呼ばれるトリガ・イベントに応答して、Open Client または Open Server が呼び出すルーチン。

コマンド構造体 (command structure)
Client-Library アプリケーションがコマンドの送信と結果の処理に使用する Client-Library の隠し構造体 (CS_COMMAND)。

コマンド (command)
Client-Library において、ct_command、ct_dynamic、または ct_cursor に対するアプリケーションの呼び出しで始まり、ct_send に対するアプリケーションの呼び出しで終了するサーバ要求。

コンテキスト構造体 (context structure)
Client-Library または Open Server アプリケーション内でアプリケーション「コンテキスト」または操作環境を定義する CS-Library の隠し構造体 (CS_CONTEXT)。CS-Library ルーチンの cs_ctx_alloc と cs_ctx_drop は、それぞれコンテキスト構造体の割り付けと削除を行います。

サーバ (server)
クライアント／サーバ・システムにおけるクライアント要求を処理し、結果をクライアントに返す部分。

システム・プロシージャ (system procedure)
Adaptive Server Enterprise が、システム管理のために提供するストアド・プロシージャ。これらのプロシージャは、システム・テーブルからの情報の取得を簡単にし、データベース管理とシステム・テーブルの更新などを可能にします。

システム・レジスタード・プロシージャ (system registered procedures)
Open Server が、レジスタード・プロシージャのノーティフィケーション (通知) の管理とステータスの監視のために提供する内部レジスタード・プロシージャ。

システム管理者 (System Administrator)
Adaptive Server Enterprise システムにおいて、ユーザ・アカウントの作成、パーミッションの割り当て、新しいデータベースの作成などのシステム管理を担当するユーザ。Adaptive Server Enterprise では、システム管理者のログイン名は “sa” です。

システム記述子 (system descriptor)
Embedded SQL において、動的 SQL 文で使用される変数の記述を保持するメモリの領域。

出力変数 (output variable)
Embedded SQL において、ストアド・プロシージャからアプリケーション・プログラムにデータを渡す変数。

照合順 (collating sequence)
「ソート順 *(sort order)*」を参照してください。

ステータス変数 (status variable)
Embedded SQL において、ストアド・プロシージャのリターン・ステータス値を受け取ることによって、プロシージャの成功または失敗を示す変数。

ストアド・プロシージャ (stored procedure)
Adaptive Server Enterprise における、名前を付けて保管された SQL 文とオプションのフロー制御文の集まり。Adaptive Server Enterprise が提供するストアド・プロシージャは、「システム・プロシージャ (system procedure)」と呼ばれます。

スレッド (thread)
Open Server アプリケーションからライブラリ・コードまでの実行のパス。また、スタック領域、ステータス情報およびイベント・ハンドラに対応するパス。

接続構造体 (connection structure)
コンテキスト内にクライアント／サーバ接続を定義する Client-Library の隠し構造体 (CS_CONNECTION)。

ソート順 (sort order)
文字データをソートするときの順序の決定に使用されます。「照合順 *(collating sequence)*」とも呼ばれます。

ターゲット・ファイル (target file)
Embedded SQL において、プリコンパイラが生成できる 3 つのファイルのうちの 1 つ。ターゲット・ファイルは元の入力ファイルと似ていますが、すべての SQL 文が Client-Library の関数呼び出しに変換されています。

データ型 (datatype)
変数に有効な値と演算を表す定義属性。

データベース (database)
特別な目的のために組織化された、関連するデータ・テーブルとその他のデータベース・オブジェクトの集まり。

デッドロック (deadlock)
データにロックを保持している 2 人のユーザが、互いに他方のデータのロックを獲得しようとしたときに起こる状態。Adaptive Server Enterprise がデッドロックを検出すると、片方のユーザのプロセスを強制終了することによってこの状態を解決します。

デフォルト・データベース (default database)
ユーザがデータベース・サーバにログインしたとき、デフォルトで指定されるデータベース。

デフォルト言語 (default language)
1. アプリケーションに対して明示的にローカライゼーションの指定を行わないとき、Open Client/Server 製品が使用する言語。デフォルト言語は、ロケール・ファイルの “default” エントリにより決定されます。

2. ユーザが明示的に言語を選択しなかったとき、Adaptive Server Enterprise がメッセージとプロンプトに使用する言語。

デフォルト (default)
明示的に何も指定されなかったときに、Open Client/Server 製品が使用する値、オプション、または動作。

動的 SQL (Dynamic SQL)
動的 SQL によって、Embedded SQL または Client-Library アプリケーションは、実行時に値の決まる変数を含む SQL 文を実行できます。

トランザクション・モード (transaction mode)
Adaptive Server Enterprise がトランザクションを管理する方法。Adaptive Server Enterprise は、2 つのトランザクション・モードをサポートしています。Transact-SQL モード (「非連鎖トランザクション」とも呼ばれる) と ANSI モード (「連鎖トランザクション」とも呼ばれる) です。

トランザクション (transaction)
バックアップまたはリカバリのために 1 つの単位として扱われる、1 つ以上のサーバ・コマンド。トランザクション内のコマンドは、1 つのグループとしてコミットされます。したがって、すべてのコマンドがコミットされるか、すべてロールバックされるかのどちらかとなります。

入力変数 (input variable)
情報をルーチン、ストアド・プロシージャ、または Adaptive Server Enterprise に渡すときに使用する変数。

配列 (array)
複数の同じタイプの変数からなる構造体。各変数は、個々にアドレッシングされます。

配列バインド (array binding)
結果カラムを配列変数にバインドする処理。フェッチのときは、複数のロー分のカラムが変数にコピーされます。

パススルー・モード (passthrough mode)

パススルー・モードのとき、ゲートウェイは、クライアントとリモート・データ・ソース間の TDS (Tabular Data Stream™) パケットを、そのパケットの内容をアンパックすることなく中継します。

バッチ (batch)

コマンドまたは文の集まり。

Client-Library のコマンド・バッチは、ct_send へのアプリケーションの呼び出しで終了する、1 つ以上の Client-Library のコマンドです。たとえば、アプリケーションは、カーソルに対する宣言、ローの選択、オープンを実行する複数のコマンドをまとめてバッチ処理できます。

Transact-SQL 文バッチは、1 つの Client-Library コマンドまたは Embedded SQL 文によって Adaptive Server Enterprise に送信される 1 つ以上の Transact-SQL 文です。

パラメータ (parameter)

1. データをルーチンに渡すとき、およびルーチンからデータを取得するときに使用する変数。

2. ストアド・プロシージャへの引数。

バルク・コピー (bulk copy)

データベースからデータをコピーしたり、データベースへデータをコピーしたりするコピー・ユーティリティ。bcp とも呼ばれます。

ブラウズ・モード (browse mode)

DB-Library と Client-Library アプリケーションが、一度に 1 つのローの値を更新しながらデータベース・ローをブラウズする方法。同様の機能を果たすカーソルの方が、一般に扱いやすくなっています。

プロパティ (property)

構造体に格納される名前付きの値。コンテキスト構造体、接続構造体、スレッド構造体、コマンド構造体は、プロパティを持ちます。構造体のプロパティは、構造体の動作を決めます。

文 (statement)

Transact-SQL または Embedded SQL におけるキーワードで始まる命令。キーワード名は、基本オペレーションまたは実行するコマンドを表します。

変換 (conversion)

「文字セット変換 (character set conversion)」 を参照してください。

ホスト・プログラム (host program)

Embedded SQL における、Embedded SQL コードを含むアプリケーション・プログラム。

ホスト言語 (host language)

アプリケーションを記述するときに使われるプログラミング言語。

ホスト変数 (host variable)

Embedded SQL における、Adaptive Server Enterprise とアプリケーション・プログラム間のデータ転送を可能にする変数。「インジケータ変数 (indicator variable)」「入力変数 (input variable)」「出力変数 (output variable)」「結果変数 (result variable)」および「ステータス変数 (status variable)」 を参照してください。

マルチバイト文字セット (multi-byte character set)

複数のバイトを使用してコード化された文字を含む文字セット。マルチバイト文字セットには、EUC JIS、シフト JIS などがあります。

メッセージ・キュー (message queue)

Open Server において、スレッドが通信するとき使用するメッセージ・ポインタのリンク・リスト。スレッドは、キューにメッセージを書き込んだり、キューからメッセージを読み込んだりすることができます。

メッセージ番号 (message number)

エラー・メッセージをユニークに識別する番号。

文字セット (character set)

各文字をユニークに定義するコード化スキームを持つ特定の (通常、標準化された) 文字の集まり。ASCII と ISO 8859-1 (Latin 1) は、よく使用される文字セットです。

文字セット変換 (character set conversion)

サーバへ入出力するときの文字セットのコード化スキームの変換。サーバとクライアントが異なる文字セットを使って通信するとき、変換が行われます。たとえば、Adaptive Server Enterprise が ISO 8859-1 を使用し、クライアントが Code Page 850 を使用する場合、文字セット変換をオンにして、サーバとクライアントが、受け渡しされるデータを同じように解釈するようにします。

ユーザ名 (user name)

「ログイン名 (login name)」を参照してください。

リスティング・ファイル (listing file)

Embedded SQL において、プリコンパイラが生成できる 3 つのファイルのうちの 1 つ。リスティング・ファイルには、入力ファイルのソース文と、情報、警告、エラーなどのメッセージが含まれます。

リモート・プロシージャ・コール (remote procedure call)

1. クライアント・アプリケーションが Adaptive Server Enterprise ストアド・プロシージャを実行する 2 つの方法のうちの 1 つ (もう 1 つの方法では、Transact-SQL の execute 文を使用します)。Client-Library のアプリケーションは、ct_command を呼び出すことによって、リモート・プロシージャ・コール・コマンドを開始します。DB-Library アプリケーションは、dbrpcinit を呼び出すことによって、リモート・プロシージャ・コール・コマンドを開始します。

2. クライアントが Open Server アプリケーションを使用して利用できる要求のタイプの 1 つ。これに応答して Open Server は、対応するレジスタード・プロシージャを実行するか、または Open Server アプリケーションの RPC イベント・ハンドラを呼び出します。

3. ユーザが接続しているサーバと異なるサーバで実行されるストアド・プロシージャ。

レジスタード・プロシージャ (registered procedure)

Open Server において、名前を付けて保管される C で記述された文の集まり。Open Server が提供するレジスタード・プロシージャは、「システム・レジスタード・プロシージャ (system registered procedure)」と呼ばれます。

ローカライゼーション (localization)

アプリケーションを特定の言語環境で使用するために設定する処理のこと。ローカライズされたアプリケーションは、通常、各国の言語と文字セットでメッセージを作成し、その国の日付、時刻、日時表記フォーマットを使用します。

ロケール名 (locale name)

言語と文字セットのペアを表す文字列。ロケール名は、「ロケール・ファイル」にリストされています。Sybase があらかじめ定義しているロケール名の他に、システム管理者が別のロケール名を定義し、ロケール・ファイルに追加することもできます。

ロケール・ファイル (locales file)

ロケール名を言語と文字セットのペアにマッピングするファイル。Open Client/Server 製品は、ローカライゼーション情報をロードするときにロケール・ファイルを検索します。

ロケール構造体 (locale structure)

Client-Library または Open Server アプリケーションのためのカスタム・ローカライゼーション値を定義する CS-Library の隠し構造体 (CS_LOCALE)。アプリケーションは、CS_LOCALE を使用して、使用される言語、文字セット、日付順、ソート順を定義できます。ロケール構造体の割り付けと割り付け解除には、CS-Library ルーチン cs_loc_alloc および cs_loc_drop を使用します。

ログイン名 (login name)

ユーザが、サーバにログインするとき使用する名前。Adaptive Server Enterprise のログイン名が有効となるのは、Adaptive Server Enterprise がシステム・テーブル *syslogins* にそのユーザのエントリを持つ場合です。

索引

記号

数字

A

B

C

D

E

F

G

I

N

O

P

R

S

T

U

W

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

て

と

な

に

ね

は

ふ

へ

ほ

ま

む

も

よ

ら

り

る

れ

ろ