SYBASE
An SAP Company

ユーザーズ・ガイド

Adaptive Server® Enterprise Sybase ODBC ドライバ

15.7

[Microsoft Windows および UNIX 版]

ドキュメント ID：DC00502-01-1570-01
改訂：2012 年 6 月

Copyright © 2012 by Sybase, Inc. All rights reserved.

このマニュアルは Sybase ソフトウェアの付属マニュアルであり、新しいマニュアルまたはテクニカル・ノートで特に示されないかぎりは、後続のリリースにも付属します。このマニュアルの内容は予告なしに変更されることがあります。このマニュアルに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、無断で使用することはできません。

このマニュアルの内容を弊社の書面による事前許可を得ずに、電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても、複製、転載、翻訳することを禁じます。

Sybase の商標は、the Sybase trademarks page (http://www.sybase.com/detail?id=1011207) で確認できます。Sybase およびこのリストに掲載されている商標は、米国法人 Sybase, Inc. の商標です。® は、米国における登録商標であることを示します。

このマニュアルに記載されている SAP、その他の SAP 製品、サービス、および関連するロゴは、ドイツおよびその他の国における SAP AG の商標または登録商標です。

Java および Java 関連の商標は、米国およびその他の国における Oracle およびその関連会社の商標または登録商標です。

Unicode と Unicode のロゴは、Unicode, Inc. の登録商標です。

このマニュアルに記載されている上記以外の社名および製品名は、当該各社の商標または登録商標の場合があります。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

目次

はじめに

対象読者

このマニュアルは、ODBC (Open Database Connectivity) を使用して、Microsoft Windows および UNIX プラットフォームで Adaptive Server® Enterprise のデータへアクセスする必要のあるアプリケーション開発者を対象としています。

このマニュアルの内容

このマニュアルは、次のように構成されています。

- 「第 1 章 ODBC プログラミングの概要」では、ODBC プログラミング・インタフェースを直接呼び出すアプリケーションの開発について説明します。
- 「第 2 章 データベースへの接続」では、クライアント・アプリケーションが ODBC を使用して Adaptive Server に接続する方法について説明します。
- 「第 3 章 サポートされている Adaptive Server の機能」では、Adaptive Server ODBC ドライバで使用できる Adaptive Server の機能について説明します。

関連マニュアル

詳細については、次のマニュアルを参照してください。

- 使用しているプラットフォームの『Software Developer’s Kit リリース・ノート』には、Adaptive Server ODBC ドライバおよび Software Developer’s Kit (SDK) に関する重要な最新情報が記載されています。
- 『Software Developer's Kit/Open Server インストール・ガイド』では、SDK および Adaptive Server ODBC ドライバのコンポーネントのインストールについて説明します。
- Adaptive Server Enterprise の『インストール・ガイド』には、Adaptive Server のインストールについて説明します。
- 使用しているプラットフォームの Adaptive Server Enterprise の『リリース・ノート』では、既知の問題および更新の詳細について説明します。

その他の情報

Sybase® Product Manuals Web サイトを使用して製品について学んでください。

- 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/Updates、Technical Documents、Case Management、Solved Cases、Newsgroups、Sybase Developer Network へのリンクもあります。

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

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品認定の最新情報にアクセスする**

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

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

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

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

❖ **コンポーネント認定の最新情報にアクセスする**

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

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

3 [Search] をクリックして、入手状況と認定レポートを表示します。

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

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

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

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

Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

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

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

3 製品を選択します。

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

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

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

表記規則

このマニュアルで使用されている表記規則は次のとおりです。

- 関数、コマンド名、コマンド・オプション名、プログラム名、プログラム・フラグ、プロパティ、キーワード、文、ストアド・プロシージャは次の形式で表記されます。

  SQLSetConnectAttr 関数を使用して、接続の詳細を制御する。たとえば、次の文では ODBC autocommit の動作がオフになる。

  ```
  sr = SQLSetConnectAttr(ConnectionHandle,
     SQL_ATTR_AUTOCOMMIT,
     (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
     SQL_IS_UINTEGER);
  ```

- 変数、パラメータ、ユーザが指定する語は、構文内と本文中では次のように斜体で表記されます。

  たとえば、次の文では、***stmt*** という名前の SQL_HANDLE_STMT ハンドルを、*dbc* という名前のハンドルを使用する接続に割り付けます。

- データベース、テーブル、カラム、データ型などのデータベース・オブジェクトの名前は、次のように表記されます。

  pubs2 オブジェクトの値。

- 関数の用途を示す例は、次のように表記されます。

```
retcode = SQLConnect( dbc,
     (SQLCHAR*) "MANGO",  SQL_NTS,
     (SQLCHAR* ) "sa", SQL_NTS,
     (SQLCHAR*) "", SQL_NTS );
```

次の表は、構文の表記規則をまとめたものです。

表 1 : 構文の表記規則

キー	定義
{ }	中カッコは、その中のオプションを 1 つ以上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。コマンドには角カッコは入力しない。
|	縦線は、中カッコまたは角カッコの中の複数のオプションのうち 1 つだけを選択できることを意味する。
,	カンマは、中カッコまたは角カッコの中の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。 カンマは他の構文内容で必須になることもある。
( )	このカッコはコマンドの一部として入力する。
...	省略記号 (...) は、直前の要素を必要な回数だけ繰り返し指定できることを意味する。省略記号はコマンドには入力しない。

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。

Software Developer’s Kit バージョン 15.7 と HTML 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。

この製品のオンライン・ヘルプは HTML でも提供され、スクリーン・リーダの読み上げで内容を理解できる機能があります。

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

ODBC プログラミングの概要

この章では、ODBC (Open Database Connectivity) プログラミング・インタフェースを直接呼び出すアプリケーションの開発について説明します。

ODBC アプリケーション開発に関する主要なマニュアルは、Microsoft ODBC SDK documentation (http://msdn.microsoft.com) です。この章では、Sybase が提供する Adaptive Server® Enterprise ODBC ドライバの概要と固有の機能について説明しますが、ODBC アプリケーション・プログラミングの詳細については説明しません。

ODBC の概要

ODBC インタフェースは、Microsoft Windows でのデータベース管理システムの標準インタフェースとして定義されている呼び出しベースのアプリケーション・プログラミング・インタフェースです。また、ODBC は、Linux など Windows 以外の多くのプラットフォームでも広く使用されています。

ソフトウェア要件

Adaptive Server Enterprise 用の ODBC アプリケーションを作成するには、次のソフトウェアが必要です。

- Adaptive Server Enterprise
- 環境に合ったプログラムを作成できる C コンパイラ
- ODBC Software Development Kit (SDK)
  - Windows プラットフォームでは、オペレーティング・システムと Visual Studio コンパイラで必要なコンポーネントがすべて提供されています。または、Microsoft Data Access Components (MDAC) をインストールします。
  - Windows 以外のプラットフォームでは、ODBC ドライバ・マネージャとヘッダー・ファイルを含んだ ODBC SDK を提供する、unixODBC や iODBC などの市販およびオープン・ソース・プロジェクトがあります。Linux オペレーティング・システムでは同様のオープン・ソース・ソフトウェアが提供されています。
  - HP HP-UX、IBM AIX、および Sun Solaris では、iAnywhere ODBC ドライバ・マネージャを使用でき、Adaptive Server ODBC ドライバのインストール・ファイルに含まれています。ODBC SDK の市販またはオープン・ソース・ソフトウェアを使用することもできますが、この場合それらを別途インストールする必要があります。

注意 iAnywhere ODBC ドライバ・マネージャでは、ODBC バージョン 1.0 および 2.0 への呼び出しを ODBC バージョン 3.x にマップすることはできません。iAnywhere ODBC ドライバ・マネージャを使用するアプリケーションでは、ODBC 機能セットの使用をバージョン 3.0 以降に制限する必要があります。

サポートされるプラットフォーム

Adaptive Server ODBC ドライバを使用できるプラットフォームのリストは、『Open Server および SDK 新機能』(Microsoft Windows、Linux、UNIX 版) を参照してください。

注意　このマニュアルの大部分では、Adaptive Server ODBC ドライバで ODBC 関数を使用してデータにアクセスするための C プログラムの作成について扱います。ODBC 接続を使用できるユーティリティ、プログラム、および 4GL RAD ツールがあります。たとえば、ODBC データ・ソースに接続する、PowerBuilder® アプリケーションまたは PHP Web ページを作成できます。ユーザは、Adaptive Server ODBC ドライバを使用したデータ・ソースのセットアップ方法さえ知っていればよいのです。データ・ソースを設定すると、これらのツールによって基になる ODBC 関数呼び出しが完全に抽象化されます。

ODBC への準拠

Adaptive Server ODBC ドライバは、ODBC 3.52 の仕様に準拠します。

ODBC 機能は、準拠のレベルに従って分類されます。機能は、コア、レベル 1、またはレベル 2 のいずれかで、レベル 2 が、ODBC サポートの中で最も詳細なレベルです。これらの機能については、*Microsoft* の『ODBC Programmer's Reference』にリストされています。

Adaptive Server ODBC ドライバは、次の例外を除き、レベル 2 に準拠します。

- **レベル 1 への準拠** Adaptive Server ODBC ドライバは、SQL_REFRESH を伴った SQLSetPos を除くすべてのレベル 1 機能をサポートします。
- **レベル 2 への準拠** Adaptive Server ODBC ドライバは、次の、SQLBulkOperations のブックマーク使用を除くすべてのレベル 2 機能をサポートします。
  (SQL_FETCH_BY_BOOKMARK、
  SQL_UPDATE_BY_BOOKMARK、
  SQL_DELEET_BY_BOOKMARK)。

古いバージョンの ODBC で開発されたアプリーションは、Adaptive Server ODBC ドライバおよび新しい ODBC ドライバ・マネージャでも機能します。新しい ODBC 機能は、古いアプリケーションでは使用できません。

ODBC ドライバ・マネージャ

ODBC ドライバ・マネージャは、ユーザ・アプリケーションと ODBC ドライバ間の通信を管理します。通常、ユーザ・アプリケーションは ODBC ドライバ・マネージャにリンクされ、ドライバ・マネージャは、アプリケーションに対応する ODBC ドライバのロードおよびアンロードのジョブを管理します。アプリケーションが ODBC ドライバ・マネージャに対して ODBC 呼び出しを行うと、ドライバ・マネージャは基本的なエラー・チェックを実行してから、これらの呼び出しを処理するか、基になる ODBC ドライバにこれらの呼び出しを渡します。

ODBC ドライバ・マネージャは必須コンポーネントではありませんが、ODBC アプリケーションの開発や展開に関する多くの問題を解決するために利用されます。ODBC ドライバ・マネージャを使用すると、次のような利点があります。

- ポータブル・データ・アクセス：別の DBMS を使用するためにアプリケーションを再構築する必要がない。
- データ・ソースへの実行時バインド。
- データ・ソースを簡単に変更可能。

ODBC ドライバ・マネージャを使用せずに Adaptive Server ODBC ドライバを使用するには、アプリケーションを Adaptive Server ODBC ドライバのライブラリに直接リンクします。結果の実行プログラムは Adaptive Server データ・ソースにだけ接続できます。

Adaptive Server ODBC ドライバは、次の ODBC ドライバ・マネージャでテスト済みです。

- Microsoft Windows の場合は、Microsoft Windows に含まれている Microsoft ODBC ドライバ・マネージャ
- Linux の場合は、Red Hat および SuSE に含まれている unixODBC ドライバ・マネージャ

- HP HP-UX、IBM AIX、および Sun Solaris の場合は、Adaptive Server ODBC ドライバのインストールに含まれている unixODBC Driver Manager バージョン 2.2.14 および Sybase iAnywhere ODBC ドライバ・マネージャ

注意　従来、Linux 64 ビット・プラットフォームの unixODBC ドライバ・マネージャは、ODBC ドライバから 4 バイトの SQLLEN を予期していました。バージョン 2.2.13 以降、unixODBC ドライバ・マネージャは 8 バイトの SQLLEN データ型を予期しています。Adaptive Server ODBC ドライバの 15.7 ESD #4 より、4 バイトおよび 8 バイト・バージョンの両方の SQLLEN ドライバがインストールに付属しています。4 バイト・バージョンがデフォルトとして設定されます。unixODBC ドライバ・マネージャのバージョンを確認し、2.2.13 以降の場合は、次のように ODBC ドライバのインストールを変更してください。

```
> cd ${SYBASE}/DataAccess64/ODBC/lib
> rm libsybdrvodb.so
> ln -s libsybdrvodb-sqllen8.so libsybdrvodb.so
```

Red Hat Enterprise Linux バージョン 6 以降は、unixODBC ドライバ・マネージャの 8 バイト・バージョンの SQLLEN を使用するため、前述の変更が必要です。

ODBC ドライバ・マネージャを使用したアプリケーションの構築

この項では、ODBC ドライバ・マネージャを使用したアプリケーションの構築方法について説明します。

Microsoft Windows

Microsoft ODBC ドライバ・マネージャには、*odbc32.dll* という名前の DLL または *odbc32.lib* という名前のインポート・ライブラリが含まれています。*odbc32.dll* ファイルは *%SystemRoot% ¥system32* にあります。*odbc32.lib* ファイルは、インストールした製品に応じて、複数のロケーションに存在する場合があります。Microsoft Visual Studio.NET を使用している場合、*odbc32.lib* は、*Microsoft Visual Studio% ¥ Vc7 ¥ PlatformSDK ¥Lib* の *%Install Path* にあります。

Microsoft ODBC ドライバ・マネージャに ODBC アプリケーションをリンクするには、*odbc32.lib* を使用します。

unixODBC ドライバ・マネージャを使用する UNIX

unixODBC ドライバ・マネージャには、*libodbc.so* という名前の共有ライブラリが含まれています。これは、*libodbc.so.1* という名前のライブラリへのソフト・リンクです。このファイルは通常 */usr/lib* ディレクトリにあります。

注意 一部の古いドライバ・マネージャでは、*libodbc.so.1* から *libodbc.so* へのソフト・リンクは作成されません。このリンクを手動で作成することをおすすめします。ODBC ドライバ・マネージャには、*libodbcinst.so.1* という名前の別の共有ライブラリも含まれています。このファイルから *libodbcinst.so* へのソフト・リンクも存在します。このソフト・リンクがシステムにない場合は、作成する必要があります。

unixODBC ドライバ・マネージャに ODBC アプリケーションをリンクするには、-lodbc フラグをリンカに渡します。

unixODBC ドライバ・マネージャが */usr/lib* ディレクトリにインストールされていない場合は、次もリンカに渡す必要があります。

```
-Ldir
```

ここで *dir* は、unixODBC ドライバ・マネージャの共有ライブラリがあるディレクトリです。

Sybase iAnywhere ODBC ドライバ・マネージャを使用する UNIX

Sybase iAnywhere ODBC ドライバ・マネージャに ODBC アプリケーションをリンクするには、-lodbc または -ldbodm フラグをリンカに渡します。また、-L*dir* フラグをリンカに渡す必要があります。ここで、*dir* は、Sybase iAnywhere ODBC ドライバ・マネージャの共有ライブラリが配置されたディレクトリです。

ODBC ドライバ・マネージャを使用しないアプリケーションの構築

アプリケーションは ODBC ドライバ・マネージャを使用しないで構築できます。Adaptive Server ODBC ドライバは、プラットフォーム固有の名前を持つ共有ダイナミック・ライブラリです。

プラットフォーム	ライブラリ・ファイル	ロケーション
Windows 32 ビット版	*sybdrvodb.dll*	*%SYBASE% ¥DataAccess ¥ODBC ¥dll*
Windows 64 ビット版	*sybdrvodb64.dl*	*%SYBASE% ¥DataAccess64 ¥ ODBC ¥dll*
UNIX 32 ビット版	*libsybdrvodb.so*	*$SYBASE/DataAccess/ODBC/lib*
UNIX 64 ビット版	*libsybdrvodb.so*	*$SYBASE/DataAccess64/ODBC/lib*

❖ **Windows の Adaptive Server ODBC ドライバと ODBC アプリケーションのリンク**

1　**リンカ / 入力** プロパティの [**追加の依存ファイル**] に *sybdrvodb.lib* を追加し、プロジェクトの **リンカ / 全般** プロパティの [**追加のライブラリ ディレクトリ**] に <*aseodbc_dir*> を追加します。

2　アプリケーションを配備するときには、Adaptive Server ODBC ドライバ共有ライブラリを含んだディレクトリの *%SYBASE%¥DataAccess¥ODBC¥dll* (32 ビット ODBC ドライバ用) または *%SYBASE%¥DataAccess64¥ODBC¥dll* (64 ビット ODBC ドライバ用) がシステム・パスに含まれていることを確認します。

❖ **UNIX の Adaptive Server ODBC ドライバと ODBC アプリケーションのリンク**

1　-lsybdrvodb フラグと -L<*aseodbc_dir*> フラグをリンカに渡します。

2　アプリケーションを配備するときには、Adaptive Server ODBC ドライバ共有ライブラリを含んだディレクトリの *$SYBASE/DataAccess/ODBC/lib* (32 ビット ODBC ドライバ用) または *$SYBASE/DataAccess64/ODBC/lib* (64 ビット ODBC ドライバ用) がライブラリ・パスに含まれていることを確認します。各プラットフォームのライブラリ・パス変数は次のとおりです。

- HP HP-UX Itanium: *SHLIB_PATH*
- IBM AIX：*LIBRARY_PATH*
- Linux および Solaris: *LD_LIBRARY_PATH*

Adaptive Server ODBC ドライバ・サンプルの使用

Adaptive Server ODBC ドライバ・サンプルは、次の場所にあります。

- 32 ビット Linux：*$SYBASE¥DataAccess¥ODBC¥samples*
- 64 ビット UNIX：*$SYBASE¥DataAccess64¥ODBC¥samples*
- Microsoft Windows：*%SYBASE%¥DataAccess¥ODBC¥samples* または *%SYBASE%¥DataAccess64¥ODBC¥samples*

各ディレクトリおよびサンプルには、次のサンプルの構築と実行に関する指示を含む *README* ファイルがあります。次のサンプルは、Microsoft Windows および UNIX で使用できます。

- *advanced*
- *asynchexec*

- *cursors*
- *odbcbatch*
- *odbcloblocator*
- *simple*

次のサンプルは、Microsoft Windows のみで使用できます。

- *adovbsample*
- *kerberos*

ODBC ハンドルの定義

ODBC アプリケーションは少数のハンドルのセットを使用して基本機能 (データベース接続や SQL 文など) を定義します。ハンドルは、32 ビットのプラットフォームで 32 ビット値、64 ビットのプラットフォームで 64 ビット値です。

ODBC プログラムに必要なハンドルのタイプは次のとおりです。

項目	ハンドル・タイプ
環境	SQLHENV
接続	SQLHDBC
文	SQLHSTMT
記述子	SQLHDESC

次のハンドルは、すべての ODBC アプリケーションで使用されます。

- **環境** 環境ハンドルは、データにアクセスするためのグローバル・コンテキストを提供します。すべての ODBC アプリケーションは起動時に必ず 1 つの環境ハンドルのみを割り付け、終了時にそのハンドルを解放する必要があります。

  環境ハンドルを割り付けるコードを次に示します。

```
SQLHENV env;
SQLRETURN rc;
rc = SQLAllocHandle( SQL_HANDLE_ENV,
       SQL_NULL_HANDLE, &env );
```

- **接続**　接続は、ODBC ドライバとデータ・ソースで指定されます。アプリケーションは、環境と関連付けられたいくつかの接続を持つことができます。接続ハンドルを割り付けても接続は確立されません。接続ハンドルが割り付けられてから、そのハンドルを使用して接続が確立されます。

  接続ハンドルを割り付けるコードを次に示します。

  ```
  SQLHDBC  dbc;
  SQLRETURN rc;
  rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
  ```

- **文**　文ハンドルは、SQL 文と SQL 文に関連する情報 (結果セットやパラメータなど) へのアクセスを提供します。各接続には、いくつかの文を含めることができます。文は、カーソル操作 (データのフェッチ) と単一文の実行 (INSERT、UPDATE、DELETE など) の両方で使用されます。

  接続ハンドルを割り付ける文を次に示します。

  ```
  SQLHSTMT stmt; SQLRETURN rc;
  rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
  ```

- **記述子**　記述子は、アプリケーションやドライバに見られる、SQL 文のパラメータまたは結果セットのカラムを記述するメタデータの集まりです。記述子は、次の 4 つの役割のいずれかになります。

  - アプリケーション・パラメータ記述子 (APD: Application Parameter Descriptor) － SQL 文内のパラメータにバインドされるアプリケーション・バッファに関する情報 (アドレス、長さ、C データ型など) を含む。
  - 実装パラメータ記述子 (IPD: Implementation Parameter Descriptor) － SQL 文内のパラメータに関する情報 (SQL データ型、長さ、null 入力可能性など) を含む。
  - アプリケーション・ロー記述子 (ARD: Application Row Descriptor) － 結果セット内のカラムにバインドされるアプリケーション・バッファに関する情報 (アドレス、長さ、C データ型など) を含む。
  - 実装ロー記述子 (IRD: Implementation Row Descriptor) － 結果セット内のカラムに関する情報 (SQL データ型、長さ、null 入力可能性など) を含む。

暗黙で割り付けられた記述子を取得する例を次に示します。

```
SQLRETURN rc;
SQLHDESC aparamdesc;
SQLHDESC iparamdesc;
SQLHDESC irowdesc;
SQLHDESC arowdesc;
rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_PARAM_DESC,
   &aparamdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
   &arowdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
   &iparamdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
   &irowdesc, SQL_IS_POINTER);
```

暗黙的な記述子は、文ハンドルが SQLFreeHandle(SQL_HANDLE_STMT, stmt) の呼び出しによって開放されると、自動的に解放されます。

ODBC ハンドルの割り付け

❖ **ODBC ハンドルの割り付け**

1 SQLAllocHandle 関数を呼び出し、次のパラメータを取得します。

- 割り付けられている項目のタイプの識別子
- 親項目のハンドル
- 割り付けられるハンドルのロケーションを指すポインタ

詳細については、*Microsoft* の『ODBC Programmer's Reference』で SQLAllocHandle を参照してください。

2 後続の関数呼び出しでこのハンドルを使用します。

3 SQLFreeHandle を使用してオブジェクトを解放し、次のパラメータを取得します。

- 解放されている項目のタイプの識別子
- 解放されている項目のハンドル

詳細については、*Microsoft* の『ODBC Programmer's Reference』で SQLFreeHandle を参照してください。

例　環境ハンドルを割り付けて解放するコードを次に示します。

```
SQLHENV env;
SQLRETURN retcode;
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
if ( retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO )
{
   // success:application code here
}
```

データ ソースへの接続

この項では、ODBC 関数を使用した Adaptive Server Enterprise データベースへの接続の確立方法について説明します。

注意　一般的に、この章で扱う例では、SQLConnect を使用します。

ODBC 接続関数の選択

ODBC は、一連の接続関数を提供します。次のどの関数を使用するかは、アプリケーションの展開および使用方法によって決まります。

- SQLConnect。最も単純な接続関数。

  SQLConnect は、データ・ソース名 (DSN: data source name)、およびオプションでユーザ ID とパスワードを使用します。データ・ソース名をアプリケーションにハードコードする場合は、SQLConnect を使用できます。

  詳細については、*Microsoft* の『ODBC Programmer's Reference』で SQLConnect を参照してください。

- SQLDriverConnect。接続文字列を使用してデータ・ソースに接続する。

SQLDriverConnect。アプリケーションは、データ・ソース外にある Adaptive Server Enterprise 固有の接続情報を使用できる。

注意 UNIX では、Adaptive Server ODBC ドライバは、SQL_DRIVER_NOPROMPT だけをサポートします。

SQLDriverConnect を使用して、データ・ソースを指定せずに接続することもできます。

詳細については、*Microsoft* の『ODBC Programmer's Reference』で SQLDriverConnect を参照してください。

- SQLBrowseConnect。SQLDriverConnect などの接続文字列を使用してデータ・ソースに接続する。

  SQLBrowseConnect を使用すると、アプリケーションは接続情報の入力を求める独自のダイアログ・ボックスを作成したり、特定のドライバ (この場合は Adaptive Server ODBC ドライバ) で使用されるデータ・ソースを参照できます。

  詳細については、*Microsoft* の『ODBC Programmer's Reference』で SQLBrowseConnect を参照してください。

接続文字列で使用できる接続パラメータの完全なリストについては、「第 2 章 データベースへの接続」を参照してください。

接続の確立

アプリケーションは、接続を確立してからデータベース操作を実行する必要があります。

❖ **ODBC 接続の確立**

1 次のように、ODBC 環境を割り付けます。

```
SQLHENV env;
SQLRETURN retcode;
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
```

2 ODBC バージョンを宣言します。

アプリケーションが ODBC バージョン 3 に従うことを宣言すると、SQLSTATE 値およびその他のバージョン依存の機能が適切な動作に設定されます。次に例を示します。

```
retcode = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION,
   (void*)SQL_OV_ODBC3, 0);
```

3　必要に応じて、データ・ソースまたは接続文字列をアセンブルします。

アプリケーションに応じて、ハードコードしたデータ・ソースまたは接続文字列を使用するか、それらを外部に保存して柔軟性を高めることができます。

4　次のように、ODBC 接続ハンドルを割り付けます。

```
retcode = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

5　接続する前に設定する必要がある接続属性を設定します (接続属性は接続を確立する前に設定する必要があるものと、確立する前後のどちらでも設定できるものがあります)。次に例を示します。

```
retcode = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_IS_UINTEGER);
```

6　次のように、ODBC 接続関数を呼び出します。

```
if (retcode == SQL_SUCCESS || retcode ==
   SQL_SUCCESS_WITH_INFO)
{
   printf( "dbc allocated¥n" );
   retcode = SQLConnect( dbc, (SQLCHAR*) "MANGO",
      SQL_NTS, (SQLCHAR*) "sa", SQL_NTS,
      (SQLCHAR*) "", SQL_NTS );
   if (retcode == SQL_SUCCESS || retcode ==
      SQL_SUCCESS_WITH_INFO)
      {
         // successfully connected.
      }
}
```

インストール・ディレクトリには、接続を確立するための詳細なサンプルがあります。

使用法についての注意

- ODBC に渡されるすべての文字列にはそれぞれ対応する長さがある。長さが不明の場合は、SQL_NTS を渡して、終了が null 文字 (¥0) でマークされる Null で終了する文字列であることを示す。
- SQLSetConnectAttr 関数を使用して、接続の詳細を制御する。たとえば、次の文では ODBC autocommit の動作がオフになる。

  ```
  retcode = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
     (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_IS_UINTEGER);
  ```

接続の多くの部分を、接続パラメータで制御できます。「第 2 章 データベースへの接続」を参照してください。

接続属性の一覧を含む詳細については、*Microsoft* の『ODBC Programmer's Reference』で SQLSetConnectAttr を参照してください。

ODBC アプリケーション内でのスレッドと接続の使用

Adaptive Server Enterprise 用のマルチスレッド ODBC アプリケーションを開発できます。スレッドごとに個別の接続を使用することをおすすめします。ただし、複数のスレッド間でオープンな接続を共有できます。

SQL 文の実行

ODBC には、SQL 文を実行するための関数がいくつか含まれます。

- **直接の実行** Adaptive Server は SQL 文を解析したうえで、実行プランを準備し、SQL 文を実行します。解析および実行プランの準備を文の準備といいます。
- **バインドされたパラメータの実行** バインドされたパラメータを使用して、実行時に文パラメータの値を設定するための SQL 文を構築および実行できます。複数回にわたり実行する文のパフォーマンスを向上させるために、バインド・パラメータと準備文を使用します。
- **準備された実行** 文の準備は、実行とは個別に行われます。繰り返し実行する文の場合、これにより準備の繰り返しが回避され、その結果としてパフォーマンスが向上します。

直接の文の実行

SQLExecDirect 関数は、SQL 文を準備、実行します。オプションで、文にパラメータを含めることができます。

次のコードは、パラメータを使用しない文の実行例を示したものです。SQLExecDirect 関数は引数として、文ハンドル、SQL 文字列、文字列サイズまたは終端インジケータ （この例では、null で終了する終端インジケータ）をとります。

❖ **ODBC アプリケーションでの SQL 文の実行**

1 SQLAllocHandle を使用して文のハンドルを割り付けます。

たとえば、次の文では、"stmt" という名前の SQL_HANDLE_STMT ハンドルを、"dbc" という名前のハンドルを使用する接続に割り付けます。

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

2　SQLExecDirect 関数を呼び出して、SQL 文を実行します。

文を宣言して実行するコードの例を次に示します。

```
SQLCHAR *deletestmt =
   "DELETE FROM department WHERE dept_id = 201";
SQLExecDirect( stmt, deletestmt, SQL_NTS) ;
```

Microsoft の『ODBC Programmer's Reference』で SQLExecDirect を参照してください。

バインドされたパラメータを持つ文の実行

この項では、バインドされたパラメータを使用して、実行時に文パラメータの値を設定するための SQL 文を構築および実行する方法を説明します。

❖ ODBC アプリケーションでバインドされたパラメータを伴う SQL 文の実行

1　SQLAllocHandle を使用して文のハンドルを割り付けます。

たとえば、次の文では、"stmt" という名前の SQL_HANDLE_STMT ハンドルを、"dbc" という名前のハンドルを使用する接続に割り付けます。

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

2　SQLBindParameter を使用して文のパラメータをバインドします。

たとえば、次の行では、部署 ID、部署名、およびマネージャ ID の値に加え、文の文字列そのものを格納する変数を宣言します。次に、"stmt" 文ハンドルを使用して実行された文の 1 つ目、2 つ目、および 3 つ目のパラメータにパラメータをバインドします。

```
#defined DEPT_NAME_LEN 20

SQLINTEGER cbDeptID = 0,
   cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptname[ DEPT_NAME_LEN ];
SQLSMALLINT deptID, managerID;
SQLCHAR *insertstmt =
   "INSERT INTO department "
   "( dept_id, dept_name, dept_head_id )"
   "VALUES (?, ?, ?,)";
SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,
   SQL_C_SSHORT, SQL_INTEGER, 0, 0,
   &deptID, 0, &cbDeptID);
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,
```

```
        SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,
        deptname, 0,&cbDeptName);
    SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,
        SQL_C_SSHORT, SQL_INTEGER, 0, 0,
        &managerID, 0, &cbManagerID);
```

3 パラメータに値を割り当てます。

手順 2 のコードのパラメータに値を割り当てるコード例を次に示します。

```
deptID = 201;
strcpy( (char * ) deptname, "Sales East" );
managerID = 902;
```

通常、これらの変数は、ユーザのアクションに反応して設定されます。

4 SQLExecDirect を使用して文を実行します。

たとえば、次の行では "stmt" 文ハンドルの "insertstmt" に格納されている文を実行します。

```
SQLExecDirect( stmt, insertstmt, SQL_NTS) ;
```

複数回にわたり実行する文のパフォーマンスを向上させるために、バインド・パラメータと準備文を使用します。

Microsoft の『ODBC Programmer's Reference』で SQLExecDirect を参照してください。

prepare 文の実行

Adaptive Server ODBC ドライバは、prepare 文を使用するための関数を一式提供します。これによって、繰り返し使用される文のパフォーマンスが向上します。

❖ 準備された SQL 文の実行

1 SQLPrepare を使用して文を準備します。

たとえば、次のコードは、insert 文を準備する例を示しています。

```
SQLRETURN retcode;
SQLHSTMT stmt;
retcode = SQLPrepare( stmt, "INSERT INTO department"
    "( dept_id, dept_name, dept_head_id )"
    "VALUES (?, ?, ?,)", SQL_NTS);
```

構文の説明は次のとおりです。

- *retcode* には、操作が成功または失敗するかどうかをテストするリターン・コードが格納されます。
- *stmt* は、文に対するハンドルを提供します。
- *?* は、文パラメータ・マーカです。

2 SQLBindParameter を使用して文のパラメータ値を設定します。

たとえば、次の関数呼び出しでは、*dept_id* 変数の値を設定します。

```
SQLBindParameter( stmt,
   1,
   SQL_PARAM_INPUT,
   SQL_C_SHORT,
   SQL_INTEGER,
   0,
   0,
   &sDeptID,
   0,
   &cbDeptID);
```

構文の説明は次のとおりです。

- *stmt* は文ハンドルです。
- *1* は、この呼び出しで最初のパラメータの値を設定することを示します。
- *SQL_PARAM_INPUT* は、パラメータが入力文であることを示します。
- *SQL_C_SHORT* は、アプリケーションで使用されている C データ型を示します。
- *SQL_INTEGER* は、データベースで使用されている SQL データ型を示します。
- *0* は、カラムの精度を示します。
- *0* は、小数桁数を示します。
- *&sDeptID* は、パラメータ値のバッファを指すポインタです。
- *0* はバッファの長さ (バイト数) を示します。
- *&cbDeptID* は、パラメータ値の長さのバッファを指すポインタです。

3 他の 2 つのパラメータをバインドし、sDeptId に値を割り当てます。

```
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,
   deptname, 0,&cbDeptName);

SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,
   SQL_C_SSHORT, SQL_INTEGER, 0, 0,
   &managerID, 0, &cbManagerID);
```

4 次のコマンドを実行します。

```
retcode = SQLExecute( stmt);
```

手順 2 ～ 4 を複数回繰り返します。

5 SQLFreeHandle を使用して文を削除します。

文を削除すると、その文に対応していたリソースが解放されます。

結果セットの使用

ODBC アプリケーションはカーソルを使用して、結果セットを操作および更新します。Adaptive Server ODBC ドライバは、さまざまなカーソルおよびカーソル操作を幅広くサポートします。

カーソル特性の選択

文を実行し、結果セットを操作する ODBC 関数は、そのタスクを実行するためにカーソルを使用します。アプリケーションでは、結果セットを返す文を実行する場合に、カーソルを暗黙的にオープンします。

結果セットを更新せず、結果セットを前方向にのみ移動するアプリケーションでは、カーソルの動作は比較的簡単になります。デフォルトでは、ODBC アプリケーションはこの動作を要求します。ODBC は、読み込み専用の前方向のみのカーソルを定義しています。また Adaptive Server ODBC ドライバは、この場合はパフォーマンスのために最適化されたカーソルを提供します。

必要な ODBC カーソル特性を設定するには、文属性を定義する SQLSetStmtAttr 関数を呼び出します。結果セットを返す文を実行する前に SQLSetStmtAttr を呼び出す必要があります。

SQLSetStmtAttr を使用して多くのカーソル特性を設定できます。Adaptive Server ODBC ドライバのカーソル・タイプを決定する特性は、SQL_ATTR_CONCURRENCY です。次の値のいずれかを設定できます。

- **SQL_CONCUR_READ_ONLY** 更新は許可されません。これがデフォルト値です。
- **SQL_CONCUR_LOCK** ローが更新できるかどうか確認するために必要なロックの最低レベルを使用します。

Microsoft の『ODBC Programmer's Reference』で SQLSetStmtAttr を参照してください。

例

次のコードでは、更新可能なカーソルを要求しています。

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CONCURRENCY,
   SQL_CONCUR_LOCK, 0 );
```

UseCursor 接続プロパティ

Adaptive Server ODBC ドライバは、結果セットを生成する SQL ステートメントが実行されるとき、次の 2 種類のカーソルのいずれかを作成できます。

- サーバ側カーソル - より多くのリソースを消費しますが、カーソルの全セマンティックのサポートが必要な場合に使用します。
- クライアント側カーソル - リソースの消費は多くなく、ほとんどの使用状況に適しています。

UseCursor 接続プロパティは、Adaptive Server ODBC ドライバによって生成されるカーソルの種類を決定するために使用します。

値：

- 0 － (デフォルト) サーバ側のカーソルが、結果セットを生成するすべての文に使用されます。
- 1 － サーバ側のカーソルが、結果セットを生成するすべての文に使用されます。

- 2 － SQLSetCursorName ODBC 関数が呼び出されたときにのみ、結果セットを生成する文にサーバ側のカーソルが使用されます。この設定は、サーバ側カーソルが必要な場合のみ、サーバ側カーソルを使用するように制限するために使用します。

注意 他のカーソル属性の設定によっては、サーバ側カーソルの要求が、Adaptive Server ODBC ドライバによって暗黙的にクライアント側カーソルに変更される場合があります。

データの検索

データベースからローを取得するには、SQLExecute または SQLExecDirect を使用して select 文を実行します。これにより、文のカーソルがオープンされます。次に、SQL_FETCH_NEXT オプションとともに SQLFetch または SQLFetchScroll を使用して、カーソルを介してローをフェッチします。アプリケーションで SQL_CLOSE オプションとともに SQLFreeStmt を使用して文を解放すると、カーソルがクローズされます。

カーソルから値をフェッチするには、アプリケーションで SQLBindCol または SQLGetData を使用します。

- SQLBindCol を使用すると、値は各フェッチで自動的に取得される。
- SQLGetData を使用する場合は、各フェッチの後にカラムごとに呼び出す必要がある。

SQLGetData は、LONG VARCHAR または LONG BINARY などのカラムの値を分割してフェッチするために使用されます。または、SQL_ATTR_MAX_LENGTH 文属性を、カラムの値全体を格納できる値に設定することもできます。SQL_ATTR_MAX_LENGTH の場合、デフォルト値は 32KB です。

simple サンプルの次のコードでは、クエリのカーソルをオープンし、そのカーソルによってデータを取得しています。コードを読みやすくするために、エラー・チェックは省略されています。

```
SQLExecDirect( stmt, "select au_fname from authors",
   SQL_NTS );
retcode = SQLBindCol( stmt, 1, SQL_C_CHAR, aufName,
   sizeof(aufName), &aufNameLen);
while(retcode == SQL_SUCCESS || retcode ==
   SQL_SUCCESS_WITH_INFO)
{
   retcode = SQLFetch( stmt );
}
```

カーソルを使用したローの更新と削除

更新および削除のためにカーソルをオープンするには、SQL_ATTR_CONCURRENCY という名前の文属性を SQL_CONCUR_LOCK に設定します。

```
SQLSetStmtAttr(stmt,SQL_ATTR_CONCURRENCY,(SQLPOINTER)
   SQL_CONCUR_LOCK,0);
```

cursor サンプルの次のコードは、更新および削除にカーソルを使用する例を示しています。簡略化のため、エラー・チェックは省略されています。

```
/* Set statement attribute for an updateable cursor */
SQLSetStmtAttr( stmt, SQL_ATTR_CONCURRENCY,
   (SQLPOINTER)SQL_CONCUR_LOCK, 0);
SQLSetCursorName(stmt1, "CustUpdate", SQL_NTS);
SQLExecDirect(stmt1, "select LastName
   from t_CursorTable ", SQL_NTS) ;
SQLFetch(stmt1);
SQLExecDirect(stmt2, "Update t_CursorTable"
   "set LastName='UpdateLastName'"
   "where current of CustUpdate", SQL_NTS) ;
```

完全なコードについては、*cursor.cpp* サンプルを参照してください。

スクロール可能なカーソルの使用

スクロール可能なカーソルは、前方にも後方にも移動できるので、スクリーンベースのアプリケーションをより簡単にサポートできます。ユーザが前後にスクロールすると、バックエンドが対応するデータを提供します。

UseCursor 接続プロパティの設定

クライアント側またはサーバ側のスクロール可能なカーソルが使用されているかどうかを特定するには、UseCursor プロパティを設定します。

- UseCursor 接続プロパティを 1 または 2 に設定すると、Adaptive Server バージョンが 15.0 以降の場合、サーバ側のスクロール可能なカーソルが使用されます。それ以前のバージョンの Adaptive Server の場合、サーバ側のスクロール可能なカーソルは使用できません。UseCursor 接続プロパティを 0 に設定すると、クライアント側のスクロール可能なカーソル (キャッシュされた結果セット) が Adaptive Server のバージョンにかかわらず使用されます。

  警告！ クライアント側のスクロール可能なカーソルを使用する場合、リソースを多量に必要とします。

- UseCursor 接続プロパティを 0 に設定すると、クライアント側のスクロール可能なカーソル (キャッシュされた結果セット) が Adaptive Server のバージョンにかかわらず使用されます。

  注意 「UseCursor 接続プロパティ」(19 ページ) を参照してください。

Static Insensitive スクロール可能カーソルのサポート

Adaptive Server ODBC ドライバは、Static Insensitive スクロール可能カーソルをサポートしています。ODBC SQLFetchScroll メソッドの実装により、ローの scroll と fetch を行います。SQLFetchScroll メソッドは MSDN ライブラリの『Microsoft Open Database Connectivity Software Development Kit Programmer’s Reference, Volume 2』に定義されている標準 ODBC メソッドです。

Adaptive Server ODBC ドライバは、次のようなスクロール・タイプをサポートします。

- SQL_FETCH_NEXT ― 次のローセットが返される。
- SQL_FETCH_PRIOR ― 前のローセットが返される。
- SQL_FETCH_RELATIVE ― 現在のローセットの開始から *n* 番目のローセットが返される。
- SQL_FETCH_FIRST ― 結果セットの最初のローセットが返される。
- SQL_FETCH_LAST ― 結果セットの最後の完全なローセットが返される。
- SQL_FETCH_ABSOLUTE ― ロー *n* から始まる rowset を返す。

スクロール可能なカーソル属性の設定

スクロール可能カーソルを使用するには、次の属性を設定する必要があります。

- SQL_ATTR_CURSOR_SCROLLABLE ― 使用しているスクロール可能カーソルのタイプ。これは SQL_SCROLLABLE の値に設定する必要がある。指定可能な値は、static、semi-sensitive、および insensitive。
- SQL_ATTR_CURSOR_SENSITIVITY ― このスクロール可能カーソルの sensitivity の値。サポートされる値は SQL_INSENSITIVE

  だけです。

スクロール可能カーソルを使用する際のオプション属性を次に示します。

- SQL_ATTR_ROW_ARRAY_SIZE ― SQLFetchScroll() メソッドの各呼び出しから返すローの数。この値を設定しない場合、デフォルト値である 1 つのローが使用されます。
- SQL_ATTR_CURSOR_TYPE ― 使用しているスクロール可能カーソルのタイプ。サポートされる値は、SQL_CURSOR_FORWARD_ONLY または SQL_CURSOR_STATIC のみです。
- SQL_ATTR_ROWS_FETCHED_PTR ― フェッチされたローの数が格納されるアドレス。SQL_ATTR_ROWS_FETCHED_PTR は、データ型 SQLUINTEGER の変数を指す。
- SQL_ATTR_ROW_STATUS_PTR ―ロー・ステータスが格納されるアドレス。SQL_ATTR_ROW_STATUS_PTR は、データ型 SQLUSMALLINT の変数を指す。

スクロール可能カーソルの実行

❖ **スクロール可能カーソルを実行するプログラムの設定**

1. 使用している環境に応じてスクロール可能カーソルの属性を設定します。

   詳細については、「スクロール可能なカーソル属性の設定」(23 ページ) を参照してください。

2. 結果をバインドします。たとえば、プログラムに次の行を追加します。

```
res=SQLBindCol(m_StatementHandle, 2, SQL_C_DOUBLE, price, 0, NULL);
res=SQLBindCol(m_StatementHandle, 3, SQL_C_LONG, quantity, 0, NULL);
```

3 SQLFetchScroll() を使用して、スクロールとフェッチを行います。たとえば、プログラムに次の行を追加します。

```
res = SQLSetStmtAttr(m_StatementHandle, SQL_ATTR_CURSOR_SCROLLABLE,
   (SQLPOINTER)SQL_SCROLLABLE,SQL_IS_INTEGER);

res = SQLSetStmtAttr(m_StatementHandle, SQL_ATTR_CURSOR_SENSITIVITY,
   (SQLPOINTER)SQL_INSENSITIVE, SQL_IS_INTEGER);

res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_NEXT,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_PRIOR,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_FIRST,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_LAST,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_ABSOLUTE,2);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_ABSOLUTE,-2);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_RELATIVE,1);
```

4 Select 文を実行します。たとえば、プログラムに次の行を追加します。

```
res = SQLExecDirect(m_StatementHandle,
   (SQLCHAR "select price, quantity from book" SQL_NTS);
```

5 結果セットとカーソルをクローズします。たとえば、プログラムに次の行を追加します。

```
res = SQLFreeStmt(m_StatementHandle,SQL_CLOSE);
```

結果の参照

スクロール可能なカーソルを実行してから、合計 *N* 個のローおよび rowset *m* として、次の結果を参照します。ここでは、*N* > *m* とします。

結果	解釈
Absolute 0	ローは返されませんでした。エラーです。
Absolute 1	*m* 件のローが返されました。
Absolute *N*	1 件のローが返されました。
Absolute *N*+1	ローは返されませんでした。エラーです。
First	最初の (1..*m*) 件のローが返されました。
Last	最後の (*N*-*m*+1 ..*N*) 件のローが返されました。
Next	SQLFetch() と同じ。
Prior	現在のローセットの前のローセットが返されました。

現在のカーソルがロー *k* および *k-a* > 0、*k* + *m* + *a* < *N*、*a*>=0 を指す場合、次のような結果になります。

結果	解釈
Relative *-a*	ロー (*k-a*、*k-a* + *m* -1) が返される。
Relative *a*	ロー (*k+a*、*k+a* + *m* -1) が返される。

スクロール可能カーソルの暗黙的な属性設定

アプリケーションで特定の属性を設定したときに暗黙的に設定される属性があります。ODBC がサポートする、暗黙的に設定されるスクロール可能カーソルの属性は次のとおりです。

アプリケーションで設定する属性	暗黙的に設定される他の属性
SQL_ATTR_CONCURRENCY = SQL_CONCUR_READ_ONLY	SQL_ATTR_CURSOR_SENSITIVITY = SQL_INSENSITIVE
SQL_ATTR_CONCURRENCY = SQL_CONCUR_LOCK	SQL_ATTR_CURSOR_SENSITIVITY = SQL_SENSITIVE
SQL_ATTR_CURSOR_SCROLLABLE = SQL_NONSCROLLABLE	SQL_ATTR_CURSOR_TYPE = SQL_CURSOR_FORWARD_ONLY
SQL_ATTR_CURSOR_SENSITIVITY = SQL_INSENSITIVE	SQL_ATTR_CONCURRENCY = SQL_CONCUR_READ_ONLY、SQL_ATTR_CURSOR_TYPE = SQL_CURSOR_STATIC
SQL_ATTR_CURSOR_TYPE = SQL_CURSOR_FORWARD_ONLY	SQL_ATTR_CURSOR_SCROLLABLE = SQL_NONSCROLLABLE
SQL_ATTR_CURSOR_TYPE = SQL_CURSOR_STATIC	SQL_ATTR_CURSOR_SCROLLABLE = SQL_SCROLLABLE

ストアド・プロシージャの呼び出し

この項では、ストアド・プロシージャを作成して呼び出し、ODBC アプリケーションから結果を処理する方法について説明します。

ストアド・プロシージャとトリガの詳細については、『ASE リファレンス・マニュアル』を参照してください。

プロシージャと結果セット

プロシージャには、2 つのタイプがあります。つまり、結果セットを返すプロシージャと、返さないプロシージャです。SQLNumResultCols を使用して次のように違いを区別できます。プロシージャが結果セットを返さない場合、結果カラムの数は 0 になります。結果セットがある場合は、他のカーソルと同様に SQLFetch または SQLFetchScroll を使用して値をフェッチできます。

パラメータは、パラメータ・マーカ (疑問符) を使用してプロシージャに渡します。SQLBindParameter を使用して各パラメータ・マーカに記憶領域を割り当てます。この場合、*INPUT*、*OUTPUT*、または *INOUT* パラメータのいずれでもかまいません。

例

advanced サンプルには、出力パラメータと戻り値を返すストアド・プロシージャと、複数の結果セットを返す別のストアド・プロシージャの例が含まれています。コードを読みやすくするために、エラー・チェックは省略されています。

```
/*
例 1:How to call a stored procedure and use input and output parameters
*/

SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &retVal, 0, SQL_NULL_HANDLE);
SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 4, 0, stor_id, sizeof(stor_id), SQL_NULL_HANDLE);
SQLBindParameter(stmt, 3, SQL_PARAM_OUTPUT, SQL_C_CHAR,
   SQL_VARCHAR, 20, 0, ord_num, sizeof(ord_num), &ordnumLen);
SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_VARCHAR, 40, 0, date, sizeof(date), &dateLen);

SQLExecDirect( stmt, "{ ?= call sp_selectsales(?,?,?)}", SQL_NTS);

/*
At this point retVal contains the return value as returned from the stored
procedure and the ord_num contains the order number as returned from the
stored procedure
*/


/*
例 2:How to call stored procedures returning multiple result sets
*/

SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 4, 0, stor_id, sizeof(stor_id), SQL_NULL_HANDLE);
```

```
SQLExecDirect(stmt, "{ call sp_multipleresults(?) }", SQL_NTS);
SQLBindCol( stmt, 1, SQL_C_CHAR, dbValue, sizeof(dbValue), &dbValueLen);
SQLSMALLINT count = 1;

while(retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
   retcode = SQLFetch( stmt );
   if (retcode == SQL_NO_DATA)
   {
     /*
     -- End of first result set --
     */
     if(count == 1)
     {
        retcode = SQLMoreResults(stmt);
        count ++;
     }
     /*
      At this point dbValue contains the value in the current row of the
      result
     */
   }
}
```

エラーの処理

ODBC のエラーは、各 ODBC 関数呼び出しと SQLGetDiagField 関数または SQLGetDiagRec 関数のいずれかの戻り値を使用して報告されます。SQLError 関数は ODBC バージョン 3 より前のバージョンで使用されていました。バージョン 3 で、SQLError 関数は SQLGetDiagRec および SQLGetDiagField 関数に置き換えられています。

すべての ODBC 関数は、次のステータス・コードのいずれかの SQLRETURN を返します。

ステータス・コード	説明
SQL_SUCCESS	エラーなし。
SQL_SUCCESS_WITH_INFO	関数は完了したが、SQLGetDiagRec の呼び出しが警告を示す。 このステータスの主な原因は、アプリケーションで提供されたバッファに対して返される値が長すぎることである。

ステータス・コード	説明
SQL_INVALID_HANDLE	無効な環境ハンドル、接続ハンドル、または文ハンドルがパラメータとして渡されている。 これは、ハンドルが解放されてから使用されている場合、またはハンドルが null ポインタである場合に頻繁に発生する。
SQL_NO_DATA	使用可能な情報がない。 このステータスが主に使用されるのはカーソルからフェッチする場合で、カーソルにこれ以上のローがないことを示す。
SQL_NEED_DATA	パラメータにデータが必要。 これは、ODBC Software Development Kit マニュアルの SQLParamData および SQLPutData の説明で取り上げられている高度な機能。

すべての環境ハンドル、接続ハンドル、および文ハンドルには、1 つまたは複数のエラーや警告を関連付けることができます。SQLGetDiagRec の呼び出しごとに 1 つのエラーに関する情報が返され、そのエラーの情報が削除されます。すべてのエラーを削除する SQLGetDiagRec を呼び出さない場合、エラーは、パラメータと同じハンドルを渡す次の関数呼び出しで削除されます。

SQLGetDiagRec の各呼び出しで、環境ハンドル、接続ハンドル、または文ハンドルのいずれかを渡すことができます。最初の呼び出しでは、SQL_HANDLE_DBC タイプのハンドルを渡して、接続に関連付けられたエラーを取得します。次の呼び出しでは、SQL_HANDLE_STMT タイプのハンドルを渡して、実行した文に関連付けられたエラーを取得します。

SQLGetDiagRec は、報告するエラーがない場合 (SQL_ERROR ではない場合)、SQL_SUCCESS を返し、報告するエラーがこれ以上ない場合は、SQL_NO_DATA_FOUND を返します。

例 1

次のコードでは、SQLGetDiagRec を使用して、コードを返しています。

```
retcode = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt );
if( retcode == SQL_ERROR )
{
   SQLGetDiagRec(SQL_HANDLE_DBC,dbc, 1, NULL, NULL,
     errmsg, 100, NULL);
   /* Assume that print_error is defined */
   print_error( "Allocation failed", errmsg );
```

```
    return;
}
```

例 2

```
retcode = SQLExecDirect( stmt,
    "delete from sales_order_items where id=2015",
    SQL_NTS );
if( retcode == SQL_ERROR )
{
    SQLGetDiagRec(SQL_HANDLE_STMT,stmt, 1, NULL, NULL,
        errmsg, 100, NULL);
    /* Assume that print_error is defined */
    print_error( "Failed to delete items", errmsg );
    return;
}
```

データ型のマッピング

表 1-1 に、Adaptive Server ODBC ドライバのデータ型のマッピングを示します。

表 1-1：データ型のマッピング

ASE のデータ型	ODBC SQL のデータ型	ODBC bind データ型
bigdatetime	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP または SQL_C_CHAR
bigtime	SQL_TYPE_TIME	SQL_C_TYPE_TIME または SQL_C_CHAR
bigint	SQL_BIGINT	SQL_C_BIGINT
binary	SQL_BINARY	SQL_C_BINARY
bit	SQL_BIT	SQL_C_BIT
char	SQL_CHAR	SQL_C_CHAR
date	SQL_TYPE_DATE	SQL_C_TYPE_DATE または SQL_C_CHAR
datetime	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP または SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_NUMERIC または SQL_C_CHAR
double	SQL_DOUBLE	SQL_C_DOUBLE
float (<16)	SQL_REAL	SQL_C_FLOAT

ASE のデータ型	ODBC SQL のデータ型	ODBC bind データ型
float (>=16)	SQL_DOUBLE	SQL_C_DOUBLE
image	SQL_LONGVARBINARY	SQL_C_BINARY
image_locator	SQL_IMAGE_LOCATOR	SQL_C_ IMAGE_LOCATOR
int[eger]	SQL_INTEGER	SQL_C_LONG
money	SQL_DECIMAL	SQL_C_NUMERIC または SQL_C_CHAR
nchar	SQL_CHAR	SQL_C_CHAR
nvarchar	SQL_VARCHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_NUMERIC または SQL_C_CHAR
real	SQL_REAL	SQL_C_FLOAT
smalldatetime	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP または SQL_C_CHAR
smallint	SQL_SMALLINT	SQL_C_SHORT
smallmoney	SQL_DECIMAL	SQL_C_NUMERIC または SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
text_locator	SQL_TEXT_LOCATOR	SQL_C_TEXT_LOCATOR
time	SQL_TYPE_TIME	SQL_C_TYPE_TIME または SQL_C_CHAR
timestamp	SQL_BINARY	SQL_C_BINARY
tinyint	SQL_TINYINT	SQL_C_TINYINT
unichar	SQL_WCHAR	SQL_C_CHAR
unitext	SQL_WLONGVARCHAR	SQL_C_CHAR
unitext_locator	SQL_UNITEXT_LOCATOR	SQL_C_ UNITEXT_LOCATOR
univarchar	SQL_WVARCHAR	SQL_C_CHAR
unsignedbigint	SQL_BIGINT	SQL_C_UBIGINT
unsignedint	SQL_INTEGER	SQL_C_ULONG
unsignedsmallint	SQL_SMALLINT	SQL_C_USHORT
varbinary	SQL_VARBINARY	SQL_C_BINARY
varchar	SQL_VARCHAR	SQL_C_CHAR

unichar、varchar、および unitext についての特別な指示

Adaptive Server データ型の unichar、univarchar、および unitext を使用し、それらのいずれかを SQL_C_CHAR にバインドする場合は、Adaptive Server ODBC ドライバでデータを Unicode からマルチバイトあるいはその逆に変換する必要があります。この変換を行うためには、*$SYBASE* ディレクトリに SYBASE 文字セットをインストールする必要があります。インストール・プログラムには、これらの文字セット・ファイルをインストールするためのオプションがあります。

ドライバで文字セットが検出されない場合や、*$SYBASE* 環境変数が設定されていない場合は、対応するエラーがアプリケーションに伝達されます。SYBASE 文字セットをインストールするには、ODBC ドライバを再インストールする必要があります。詳細については、使用しているプラットフォームの『Software Developer’s Kit/Open Server インストール・ガイド』を参照してください。

注意　Adaptive Server ODBC ドライバでは、古いアプリケーションをサポートするために、unitext、univarchar、または unichar カラムが SQL_C_DEFAULTDriver としてバインドされた場合、デフォルトの型を SQL_C_CHAR と仮定します。アプリケーションで unicode としてバインドするには、明示的に SQL_C_WCHAR のバインド型を使用する必要があります。

---

bigint についての特別な指示

Adaptive Server テーブルで、bigint 型の列を識別子として使用している場合 (ID やプライマリ・キーなど)、Microsoft Access などのアプリケーションが Adaptive Server ODBC ドライバ経由でこのテーブルにアクセスすると、カラムの値が “#deleted” と表示され、そのテーブルでそれ以上操作できません。この対処方法として、CHANGEBIGINTDEFAULT を 1 に設定します。

CHANGEBIGINTDEFAULT の値は次のとおりです。

- 0 －デフォルト値。SQL_C_DEFAULT を SQL_C_BIGINT にバインドします。
- 1 － SQL_C_DEFAULT を SQL_C_CHAR にバインドします。この設定は、Microsoft Access や Microsoft Excel などのアプリケーションから bigint 識別子によって Adaptive Server テーブルにアクセスする場合に使用します。
- 2 － SQL_C_DEFAULT を SQL_C_WCHAR. にバインドします。

第 2 章

データベースへの接続

この章では、クライアント・アプリケーションが ODBC を使用して Sybase Adaptive Server Enterprise に接続する方法について説明します。

接続の概要

Adaptive Server Enterprise を使用するクライアント・アプリケーションは、まず、Adaptive Server への接続を確立する必要があります。接続は、クライアント・アプリケーションのすべてのアクティビティを実行するチャネルとなります。たとえば、ユーザがデータベース上で実行できる操作はユーザ ID によって決定されますが、このユーザ ID は接続確立要求の一部として送信され、データベース・サーバに渡されます。Adaptive Server ODBC ドライバは、クライアント・アプリケーションからの呼び出しに含まれている接続情報を、初期化ファイル内のディスクに格納されている情報と併用して、必要なデータベースを実行している Adaptive Server サーバを見つけて接続します。

ODBC メタデータ・ストアド・プロシージャのインストール

ODBC メタデータ・ストアド・プロシージャは、ODBC の機能が想定どおりに動作することを保証します。接続する必要があるすべての Adaptive Server サーバで、ODBC ドライバを使用して ODBC メタデータ・ストアド・プロシージャのバージョンを確認し、必要に応じて更新することをおすすめします。

注意 「ODBC ドライバのバージョン情報ユーティリティ」(54 ページ) を -connect で使用して、メタデータ・ストアド・プロシージャが最新か、更新が必要であるかを確認してください。

❖ メタデータ・ストアド・プロシージャのインストール

この手順は、sybsystemprocs に ODBC メタデータ・ストアド・プロシージャをインストールします。

このスクリプトを正常に実行するには、sybsystemprocs にストアド・プロシージャを作成するパーミッションを持っている必要があります。

1 Adaptive Server ODBC ドライバのインストール・ディレクトリの下の *sp* ディレクトリに移動します。

- Adaptive Server ODBC ドライバ Microsoft Windows 32 ビット版：*%SYBASE% ¥DataAccess ¥ODBC ¥sp*
- Adaptive Server ODBC ドライバ Microsoft Windows 64 ビット版：*%SYBASE% ¥DataAccess64 ¥ODBC ¥sp*
- Adaptive Server ODBC ドライバ Linux 32 ビット版 *$SYBASE ¥DataAccess ¥ODBC ¥sp*
- Adaptive Server ODBC ドライバ UNIX 64 ビット版 *$SYBASE ¥DataAccess64 ¥ODBC ¥sp*

2 *install_odbc_sprocs* スクリプトを実行します。

- Adaptive Server ODBC ドライバ Microsoft Windows 版：

```
install_odbc_sprocs ServerName username
   [password]
```

- Adaptive Server ODBC ドライバ UNIX 版：

```
./install_odbc_sprocs ServerName username
   [password]
```

各パラメータの意味は次のとおりです。

- *ServerName* は、Adaptive Server の名前。
- *username* は、サーバに接続するユーザの名前。
- *[password]* は、ユーザ名のパスワード。値が null の場合は、パラメータを空にする。

接続パラメータの構造

アプリケーションからデータベースに接続する場合は、一連の接続パラメータを使用して接続を定義します。接続パラメータには、サーバ名、データベース名、ユーザ ID などの情報が含まれています。各接続パラメータは、キーワードと値の組み合わせとして、parameter=value という形式で指定されます。たとえば、ユーザ ID の接続パラメータは、次のように指定します。

```
UID=sa
```

接続文字列として渡される接続パラメータ

接続パラメータは、Adaptive Server ODBC ドライバに接続文字列として渡され、次のようにセミコロンで区切られます。

```
parameter1=value1;parameter2=value2;...
```

通常、アプリケーションによって構築され、ドライバに渡される接続文字列は、ユーザが情報を入力する方法と直接対応していません。代わりに、ユーザがダイアログ・ボックスに入力するか、アプリケーションが初期化ファイルから接続情報を読み込むことができます。

文字セット

CharSet 接続プロパティは Adaptive Server に文字データを送信するためにドライバが使用する文字セットを定義しますが、ClientCharset 接続プロパティはクライアント・アプリケーションが使用する文字セットを定義します。

CharSet の有効な値は次のとおりです。

- ServerDefault －この値の指定時には、Adaptive Server ODBC ドライバが Adaptive Server のデフォルト文字セットを使用してサーバと通信します。クライアントとサーバで異なる文字セットが使用されている場合、Adaptive Server ODBC ドライバによってクライアント用に文字データが変換されます。
- ClientDefault －この値の指定時には、Adaptive Server ODBC ドライバがクライアントで指定された文字セットを使用して Adaptive Server と通信します。Adaptive Server のデフォルトの文字セットがクライアントと異なる場合、Adaptive Server によって文字データがクライアントの文字セットに変換されます。Adaptive Server で文字セットの変換が実行される場合には、より多くのリソースが使用されます。
- NoConversions －この値の指定時には、Adaptive Server ODBC ドライバはクライアントの文字セットを無視して、文字データを変換しません。この設定では、クライアント・アプリケーションで、クライアントの文字セットと Adaptive Server のデフォルトの文字セットの間で正しくデータを変換する必要があります。この値は、特殊な状況のみで使用します。たとえば、Adaptive Server に保存された文字データをカスタマイズされた文字セットの変換ロジックを使用してクライアント・アプリケーションで変更する必要がある場合などです。

注意 Microsoft Windows ODBC データ・ソース・アドミニストレータでは、[詳細設定] ウィンドウの [サーバ・デフォルト]、[クライアント文字セット]、および [変換なし] フィールドがそれぞれ、CharSet 値の ServerDefault、ClientDefault、および NoConversions に対応します。

Adaptive Server ODBC ドライバは、次のようにプラットフォームに応じてクライアントの文字セットを決定します。

- Microsoft Windows では、ログイン・セッションで選択されるデフォルトのクライアント文字セットが ANSI コード・ページになります。有効なコード・ページのタイプは、ANSI、OEM、および Other です。Other を選択した場合は、有効な Windows のコード・ページ値を入力する必要があります。

- UNIX では、Adaptive Server ODBC ドライバがデフォルトで、LC_CTYPE および LANG 環境変数を確認します。それらが設定されていない場合、ドライバのデフォルトによって ISO 8859-1 に設定される。これらの環境変数の 1 つが設定されている場合、ドライバは *$SYBASE/locales/locales.dat* ディレクトリで *locales.dat* を検索して、対応する Adaptive Server 文字セットを選択する。ファイルが見つからない場合は、メモリ内の独自のマップを参照して、対応する Adaptive Server 文字セットを検索する。

Adaptive Server ODBC ドライバの設定

ODBC アプリケーションがデータベースに接続するときには、通常は ODBC データ・ソースを使用します。ODBC データ・ソースは接続パラメータの集まりであり、レジストリまたはファイルに保存されます。Windows 以外のプラットフォームの ODBC データ・ソースは、通常 *ini* ファイル内にあります。ほとんどの ODBC ドライバ・マネージャには、ODBC ドライバとデータ・ソースを設定するための GUI ツールが用意されています。

Microsoft Windows

Sybase SDK インストール・プログラムを使用して Adaptive Server ODBC ドライバをインストールすると、ドライバはローカル・マシンに登録されます。regsvr32 ユーティリティを使用して Microsoft Windows に Adaptive Server ODBC ドライバを手動で登録できます。

Adaptive Server ODBC ドライバの登録

注意　Sybase SDK インストール・プログラムを使用して Adaptive Server ODBC ドライバをインストールしている場合は、Adaptive Server ODBC ドライバを手動で登録する必要はありません。

❖ **Microsoft Windows x86 32 ビット版への Adaptive Server ODBC ドライバ 32 ビットの手動登録**

1 *%SYBASE%¥DataAccess¥ODBC¥dll* ディレクトリに移動します。このディレクトリには、Adaptive Server ODBC ドライバの DLL が含まれています。

2 次の regsvr32 ユーティリティを実行して、*HKEY_LOCAL_MACHINE¥SOFTWARE¥ODBC¥ODBCINST.INI* キーにレジストリ・エントリを作成します。

```
regsvr32 sybdrvodb.dll
```

❖ **Microsoft Windows x86-64 64 ビット版への Adaptive Server ODBC ドライバ 64 ビットの手動登録**

1 *%SYBASE%¥DataAccess64¥ODBC¥dll* ディレクトリに移動します。このディレクトリには、Adaptive Server ODBC ドライバの DLL が含まれています。

2 次の regsvr32 ユーティリティを実行して、*HKEY_LOCAL_MACHINE¥SOFTWARE¥ODBC¥ODBCINST.INI* キーにレジストリ・エントリを作成します。

```
regsvr32 sybdrvodb64.dll
```

❖ **Microsoft Windows x86-64 64 ビット版への Adaptive Server ODBC ドライバ 32 ビットの手動登録**

1 *%SYBASE%¥DataAccess¥ODBC¥dll* ディレクトリに移動します。このディレクトリには、Adaptive Server ODBC ドライバの DLL が含まれています。

2 次の regsvr32 ユーティリティを実行して、*HKEY_LOCAL_MACHINE¥SOFTWARE¥Wow6432Node¥ODBC¥ODBCINST.INI* キーにレジストリ・エントリを作成します。

```
regsvr32 sybdrvodb.dll
```

注意 Microsoft Windows x86-64 64 ビット版で Adaptive Server ODBC ドライバ 32 ビットを使用してデータ・ソースを設定するには、32 ビット版の ODBC データ・ソース・アドミニストレータ odbcad32.exe (C:¥WINDOWS¥SysWOW64¥ にあります) を使用します。

データ ソースの設定

❖ データ ソースの設定

1　ODBC アドミニストレータを起動します。詳細な手順については、使用している Microsoft Windows オペレーティング・システムのオンライン・ヘルプを参照してください。

2　[ユーザー DSN] タブを選択します。[追加] をクリックします。

3　ドライバのリストから "Adaptive Server Enterprise" を選択します。

4　[完了] をクリックします。

5　[一般] タブを選択します。次のフィールドに値を入力します。

- データ・ソース名－データ・ソースの名前
- 説明－データ・ソースの説明
- サーバ名－ Adaptive Server Enterprise のホスト名
- サーバ・ポート－ Adaptive Server Enterprise のポート番号
- データベース名－データベース名
- ログイン ID － Adaptive Server Enterprise データベースにログインするユーザ名

6　すべての select 文でカーソルをオープンする場合は、[カーソルの使用] を選択します。

7　必要に応じて、[接続] タブと [詳細] タブに入力します。

8　[OK] をクリックして変更を保存します。

注意　接続パラメータの詳細については、「接続パラメータの使用」(43 ページ) を参照してください。

UNIX

unixODBC ドライバ・マネージャでは、GUI およびコマンド・ラインからドライバとデータ・ソースを設定できます。GUI ツールとコマンド・ライン構文での設定方法については、ODBC ドライバ・マネージャのマニュアルを参照してください。

注意 このドライバを使用する Adaptive Server ODBC ドライバとデータ・ソースは、unixODBC ドライバ・マネージャから GUI ツールを使用して設定できません。コマンド・ライン・インタフェースを使用する必要があります。

---

unixODBC ドライバ・マネージャのコマンド・ライン・ツールを使用してドライバとデータ・ソースを設定する場合は、テンプレート・ファイルを指定する必要があります。サンプルのテンプレートについては、次の項で説明します。これらのテンプレートは、次の場所にもあります。

- Adaptive Server ODBC ドライバ 32 ビット：
  $SYBASE/DataAccess/ODBC/samples
- Adaptive Server ODBC ドライバ 64 ビット：
  $SYBASE/DataAccess64/ODBC/samples

次に、ドライバ・テンプレート・ファイルの例を示します。

```
[Adaptive Server Enterprise]
Description=Sybase ODBC Driver
Driver=/install dir/driver library name
FileUsage=-1
```

各パラメータの意味は次のとおりです。

- *install dir* は、Adaptive Server ODBC ドライバ・インストールへのパス。
- *driver library name* は、ドライバ・ライブラリの名前。

Adaptive Server ODBC ドライバのインストール

unixODBC コマンド・ライン・ツールを使用して Adaptive Server ODBC ドライバをインストールするには、次のコマンドを実行します。

```
# odbcinst -i -d -f driver template file
```

ここで、*driver template file* は、Adaptive Server ODBC ドライバ・テンプレート・ファイルへの完全なパスです。

注意　多くの場合、このコマンドはルート・ユーザとして実行する必要があります。このコマンドは、ルートが所有している *odbcinst.ini* ファイルを修正するからです。

データ ソースの設定

unixODBC ドライバ・マネージャ

次に、データ・ソース・テンプレートを示します。

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
```

unixODBC コマンド・ライン・ツールを使用して Adaptive Server ODBC ドライバのデータ・ソースを設定するには、次のコマンドを実行します。

```
# odbcinst -i -s -f dsn template file
```

ここで、*dsn template file* は、Adaptive Server ODBC データ・ソース・テンプレート・ファイルへの完全なパスです。これによって、*odbc.ini* ファイルにデータ・ソースのエントリが作成されます。

注意　ODBC データ・ソースの設定に必要な正確なコマンドは、使用している ODBC ドライバ・マネージャによって決まります。

Sybase iAnywhere ODBC ドライバ・マネージャ

Sybase iAnywhere ODBC ドライバ・マネージャは、*odbc.ini* で提供される情報を使用して、ドライバやその他の接続情報を検出します。ODBCINI 変数は、*odbc.ini* ファイルの場所を定義します。

❖ ODBC ドライバとデータ・ソースの手動設定

1　次の *odbc.ini* ファイルを作成します。

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
```

```
Password=
Driver=complete_path_to_libsybdrvodb.so
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
```

2 ODBCINI 環境変数に *odbc.ini* ファイルへの完全なパスを設定します。

ODBC *ini* ファイル

ODBC ドライバ・マネージャは、*ini* ファイルまたはシステム・レジストリ内にドライバおよびデータ・ソースの情報を保存します。

注意　これらの *ini* ファイルの正確なパスについては、ODBC ドライバ・マネージャのマニュアルを参照してください。

Microsoft Windows

odbc.ini および *odbcinst.ini* ファイルは、*c:¥winnt* ディレクトリにあります。Microsoft ODBC ドライバ・マネージャは、実行時にアプリケーションがデータ・ソースへの接続を要求すると、これらのファイルまたはレジストリを調べます。

UNIX

システムにインストールされている ODBC ドライバについての情報は、*odbcinst.ini* ファイルに保存されます。このファイルは通常 */etc/odbcinst.ini* にあります。

データ・ソースについての情報は、次の 2 つのファイルのどちらかに保存されます。

- ユーザだけが利用できるユーザ・データ・ソース情報は、*$HOME/.odbc.ini* ファイルに保存される。ここで *$HOME* は、ユーザのホーム・ディレクトリ。
- システムのどのユーザでも利用できるシステム・データ・ソース情報は、通常 */etc/odbc.ini* ファイルに保存される。同じデータ・ソースが両方のファイルで定義されている場合、ユーザ・データ・ソースが優先される。

ODBC ドライバ・マネージャは、実行時にアプリケーションがデータ・ソースへの接続を要求すると、これらのファイルを調べます。これらの *ini* ファイルの正確なパスについては、ODBC ドライバ・マネージャのマニュアルを参照してください。ドライバ・マネージャによっては、別のロケーションを使用する場合もあります。

アプリケーションが ODBC ドライバ・マネージャを使用せず、Adaptive Server ODBC ドライバを直接使用している場合、*ini* ファイルは別の方法で検索されます。Adaptive Server ODBC ドライバは最初に現在の作業ディレクトリにある *odbc.ini* という名前のファイルを検索し、ファイルが見つからない場合や、ファイル内でデータ・ソースが見つからない場合は、*$SYBASE/odbc.ini* を検索します。

アプリケーションで、Sybase iAnywhere ODBC Driver Manager を使用している場合、ODBCINI 環境変数に *odbc.ini* ファイルへの完全なパスを設定します。デフォルトでは、*odbc.ini* は *$SYBASE* の配下にあります。

データ・ソースを使用した接続

ODBC アプリケーションは通常、接続先の各データベースのクライアント・コンピュータのデータ・ソースを使用します。一連の Adaptive Server Enterprise 接続パラメータを、ODBC データ・ソースという形でシステム・レジストリまたは *ini* ファイルに保存できます。データ・ソースがある場合、DataSourceName (DSN) 接続パラメータを使用して、接続文字列には目的のデータ・ソースを指定するだけで済みます。

```
DSN=my data source
```

接続パラメータの使用

表 2-1 に、Adaptive Server ODBC ドライバに提供できる DSN パラメータ以外の接続パラメータのリストを示します。

表 2-1：接続パラメータ

プロパティ名	説明	必須	デフォルト値
AlternateServers	カンマで区切られた host:port の組み合わせのリスト (server1:port1,server2:port2,...,serverN:portN など)。接続を確立しているとき、Adaptive Server ODBC ドライバは、AlternateServers のリストにあるホストとポートを試す前に、まず Server プロパティと Port プロパティで指定されたホストとポートに接続する。 高可用性環境で AlternateServers を使用する方法は、「サポートされている Adaptive Server クラスタ・エディションの機能」(59 ページ) を参照。	いいえ	空
AnsiNull	ODBC に厳格に準拠し、"= NULL" を使用不可とする。代わりに "IsNull" を使用する。	いいえ	1
ApplicationName	クライアント・アプリケーションを識別するために Adaptive Server が使用する名前。	いいえ	空
AuthenticationClient	Kerberos Authentication で使用されるクライアント・ライブラリの種類。有効な値は次のとおり。 • activedirectory • cybersafekerberos • mitkerberos	いいえ	空
AutoCommit	autocommit の状態を設定する。有効な値は次のとおり。 • 0 ー autocommit はオフ (SQL_ATTR_AUTOCOMMIT を SQL_AUTOCOMMIT_OFF に設定するのと同じ) • 1 ー (デフォルト) autocommit はオン (SQL_ATTR_AUTOCOMMIT を SQL_AUTOCOMMIT_ON に設定するのと同じ)	いいえ	1

プロパティ名	説明	必須	デフォルト値
BackEndType	定義しているデータ・ソースのターゲット・タイプ。Adaptive Server ODBC ドライバは、Adaptive Server などのデータベース・システムや Sybase 以外のデータベース・システムへのゲートウェイなど、複数のターゲット・オブジェクトと通信できる。有効な値は次のとおり。 • ASE • DC DB2 Access Service • DC TRS • MFC Gatewayless • Replication Server 詳細については、「z/OS オプションに対する Mainframe Connect および DirectConnect のサポート」(72 ページ) を参照。	いいえ	ASE
BufferCacheSize	入力／出力バッファをプール内に保持する。結果が大きくなる場合、この値を大きくするとパフォーマンスが向上する。	いいえ	20
ChangeBigIntDefault	bigint カラムのデフォルトの C 言語型を指定する。有効な値は次のとおり。 • 0 － SQL_C_SBIGINT/SQL_CUBIGINT • 1 － SQL_C_CHAR • 2 － SQL_C_WCHAR	なし	0
CharSet	Adaptive Server との通信に使用する文字セットを指定する。有効値は、ServerDefault、ClientDefault、NoConversions。 「文字セット」(35 ページ) を参照。	いいえ	ServerDefault
ClientCharset	クライアント文字セットを指定する。 「文字セット」(35 ページ) を参照。	いいえ	オペレーティング・システムが現在使用する文字セット。
ClientHostName	ログイン・レコードでサーバに渡されたクライアント・ホストの名前。	いいえ	空
ClientHostProc	ログイン・レコードでサーバに渡されたこのホスト・マシン上のクライアント・プロセスの ID。	いいえ	空
CodePageType	使用する文字コードの種類を指定する。有効な値は、ANSI、OEM、および Other。	いいえ	ANSI

プロパティ名	説明	必須	デフォルト値
CommandTimeOut	クライアントがコマンドの実行を待機する時間 (秒単位)。指定した時間内にコマンドが実行されない場合、クライアントはそのコマンドをキャンセルしてエラーを生成する。	いいえ	0. 値が 0 の場合は時間制限がなく、クライアントが無制限にコマンドを実行できることを意味する。
ConnectionTimeOut	クライアントが通信の確立を待機する時間 (秒単位)。指定した時間内に通信が確立されない場合、クライアントはその試行をキャンセルしてエラーを生成する。	いいえ	0. 値が 0 の場合は時間制限がなく、ODBC はデータベース接続が確立するまで無制限に待機することを意味する。
CRC	ストアド・プロシージャで複数の update 文が実行されたとき、ドライバはデフォルトで、更新されたレコード総数を返す。このカウントには、update や insert に設定されたトリガで実行されるすべての更新操作も含まれる。 ドライバが最後の更新カウントだけを返すようにする場合は、このプロパティを 0 に設定する。	いいえ	1
Database	接続するデータベース。	いいえ	空
DataIntegrity	Kerberos のデータの整合性を有効にする。	いいえ	0 (無効)
DistributedTransaction Protocol	分散トランザクションに使用するプロトコルを設定。有効値は、XA (デフォルト) および OLE。	いいえ	XA
DSPassword	LDAP サーバで匿名アクセスが許可されない場合に、LDAP サーバでの認証に使用するパスワード。パスワードは DSURL (Directory Service URL) でも指定可能。	いいえ	空
DSPrincipal	LDAP サーバで匿名アクセスが許可されない場合に、LDAP サーバでの認証に使用するユーザ名。プリンシパルは DSURL でも指定可能。	いいえ	空
DSURL	LDAP サーバへの URL。	いいえ	空

プロパティ名	説明	必須	デフォルト値
DTCProtocol (Microsoft Windows のみ)	分散トランザクションを使用する場合に、ドライバで XA プロトコルまたは OleNative プロトコルのどちらかを使用することを許可する。詳細については、「第 3 章 サポートされている Adaptive Server の機能」の「分散トランザクションの使用」(63 ページ) を参照。	いいえ	XA
DynamicPrepare	1 に設定した場合、ドライバはコンパイルまたは準備のために SQLPrepare 呼び出しを Adaptive Server に送信する。これにより、同じクエリを繰り返し使用する場合に、パフォーマンスを向上させることができる。	いいえ	0
EnableBulkLoad	バルク・ロードのサポートを有効化するかどうかを指定。 • 0 ー バルク・ロードのサポートは無効。 • 1 ー**配列挿入**を使用したバルク・ロードが有効。 • 2 ー**バルク・コピー**・インタフェースを使用したバルク・ロードが有効。 • 3 ー高速ログ出力**バルク・コピー**・インタフェースを使用したバルク・ロードが有効。 「バルク・ロードのサポート」(70 ページ) を参照。 EnableBulkLoad をプログラムによって設定するには、Sybase 専用の SQL_ATTR_ENABLE_BULK_LOAD 接続属性を使用する。この属性は、EnableBulkLoad と同じ値を取る。	はい	0
EnableLOBLocator	ラージ・オブジェクト (LOB) ロケータのサポートを有効化するかどうかを指定。 • 0 ー LOB ロケータのサポートは無効。 • 1 ー LOB ロケータのサポートは有効。 「ラージ・オブジェクト (LOB) のロケータのサポート」(108 ページ) を参照。	いいえ	0

プロパティ名	説明	必須	デフォルト値
EnableMDACheck	サーバにインストールされた MDA スクリプトのチェック・モードを設定。有効な値は次のとおり。 • 0 − MDA スクリプトのチェックを無効化。 • 1 − MDA スクリプトのバージョンがドライバのバージョンよりも古く、接続を継続する場合に警告を発行。 • 2 − MDA スクリプトのバージョンがドライバのバージョンよりも古く、接続に失敗した場合にエラーを発行。	いいえ	0
EnableServerPacketSize	最適なパケット・サイズを選択するために、Adaptive Server サーバ・バージョン 15.0 以降を許可する。	いいえ	1
Encryption	指定されている暗号化方式。指定できる値：ssl。	いいえ	空
EncryptPassword	パスワードが暗号化フォーマットで転送されるかどうかを指定する。 • 0 − プレーン・テキスト形式のパスワードを使用します。 • 1 − 暗号化されたパスワードを使用します。サポートされていない場合、エラー・メッセージを返します。 • 2 − 暗号化されたパスワードを使用します。サポートされていない場合、プレーン・テキスト形式のパスワードを使用する。 **注意** パスワードの暗号化が有効になっていて、サーバが非対称暗号化をサポートしている場合、非対称暗号化が対称暗号化の代わりに使用される。	いいえ	0
Escape	ODBC エスケープ文字を設定。	いいえ	‘¥’
FetchArraySize	サーバから結果をフェッチする場合にドライバが取得するロー数を指定する。	いいえ	25
HASession	高可用性を有効にするかどうかを指定する。0 は高可用性を無効にし、1 は高可用性を有効にする。	いいえ	0

プロパティ名	説明	必須	デフォルト値
HomogeneousBatch	パラメータのバッチ処理モードを指定。 • 0 ー バラメータのバッチ処理を無効化。 • 1 ー Adaptive Server のパラメータ・バッチ処理プロトコルを有効化。 • 2 ー Adaptive Server のバルク挿入プロトコルを有効化。 HomogeneousBatch をプログラムによって設定するには、Sybase 専用の SQL_ATTR_HOMOGENEOUS_BATCH 接続属性を使用する。この属性は、HomogeneousBatch と同じ値を取る。	いいえ	0
IgnoreErrorsIfRS Pending	エラー・メッセージが表示された場合に、ドライバの処理を続行するかどうかを指定する。1 に設定した場合、ドライバでエラーが無視され、サーバからさらに結果が取得可能な場合は結果の処理が続行される。0 に設定した場合、ドライバではエラーが発生したときに保留中の結果があっても結果の処理を停止する。	いいえ	0
InitializationString	ログイン時に実行する Transact-SQL 文を設定します。	いいえ	空
Isolation	接続の初期の独立性レベルを指定。有効な値は次のとおり。 • 0 ー読み取りはコミットされない。 • 1 ー読み取りがコミットされる。 • 2 ー繰り返し可能読み出し • 3 ー直列化可能	いいえ	0
Language	Adaptive Server が返すエラー・メッセージの言語。	いいえ	空 - デフォルトでは英語を使用。
LoginTimeOut	アプリケーションへ戻る前に、ログインの試行を待機する秒数。0 に設定するとタイムアウトが無効になり、接続の試行を永久的に待機する。	いいえ	15

プロパティ名	説明	必須	デフォルト値
NormalizeWCharParams	Unicode 文字列正規化を有効にするかどうかを指定する。Adaptive Server の設定オプション enable unicode normalization が 0 に設定されているとき、このプロパティを 1 に設定する。有効な値は次のとおり。 • 0 – Unicode 文字列の正規化を無効にする。 • 1 – Unicode 文字列の正規化を有効にする。	いいえ	0
OldPassword	現在のパスワード。OldPassword に null や空の文字列以外の値が含まれている場合、現在のパスワードは PWD に含まれる値に変更される。	いいえ	空
PacketSize	Adaptive Server とクライアント間で送受信されるネットワーク・パケット 1 つあたりのバイト数。	いいえ	ドライバが Adaptive Server 15.0 以降に接続したときに判別されたサーバ。より古いバージョンでは、デフォルトは 512 になる。
ParamsetsBeforeThread	バッチ処理の間、応答スレッドを開始するまでに送信するパラメータ・セットの数を指定。	いいえ	50
Port	Adaptive Server のポート番号。	はい	空
ProgName	ログイン時に使用する progname の値を設定する。 指定する値は、30 文字でトランケートされる。	いいえ	空
ProtocolCapture	このプロパティは、ODBC アプリケーションとサーバ間でデバッグの目的で交換される TDS パケットを取得するために使用する。このプロパティは、取得ファイル・プレフィクスを指定することで有効になる。 「TDS プロトコルの取得」(94 ページ) を参照。	いいえ	空

プロパティ名	説明	必須	デフォルト値
PWD、Password	パスワードの値が含まれる。通常のログインを実行すると、OldPassword が設定されず PWD に現在のパスワードの値が含まれる。パスワードを変更すると、OldPassword に現在のパスワードが設定され、PWD には新しいパスワードの値が含まれる。	いいえ (ユーザ名にパスワードが不要な場合)	空
QuotedIdentifier	Adaptive Server で二重引用符で囲まれた文字列を識別子として処理するかどうかを指定する。 • 0 － 二重引用符で囲まれた識別子を有効にしない。 • 1 － 引用符で囲まれた識別子を有効にする。	いいえ	0
ReadWriteUnknown	設定すると、更新されないカラムが読み込み / 書き込み不明としてマークされる。	いいえ	0
ReleaseLocksOnCursorClose	カーソルがクローズされた場合に Adaptive Server が独立性レベル 2 および 3 で共有読み込み専用カーソルのロックを解除するかを指定する。 • 0 －クローズされた共有カーソルロックの解除を無効にする。 • 1 －クローズされた共有カーソルロックの解除を有効にする。	いいえ	0
RemotePwd	サーバに `servername,password;servername,password;...` のフォーマットでリモート・パスワードを設定する。	いいえ	空
ReplayDetection	Kerberos リプレイ検出を有効にする。	いいえ	0
RestrictMaximumPacketSize	EnableServerPacketSize に 1 を設定した場合にメモリに制約があるときは、このプロパティに 512 の倍数 (最大 65536) の int 値を設定する。	いいえ	0
RetryCount、RetryDelay	再試行の動作を制御する。 RetryCount は、接続の失敗をレポートするまでにサーバへの接続を試行する回数。再試行の間、ドライバは RetryDelay の秒数だけ遅延する。 デフォルトでは、ODBC アプリケーションは接続を再試行しない。	いいえ	0

プロパティ名	説明	必須	デフォルト値
SecondaryPort	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server のポート番号。	はい (HASession が 1 に設定されている場合)	空
SecondaryServer	アクティブ／アクティブ設定またはアクティブ／パッシブ設定でフェールオーバ・サーバとして機能する Adaptive Server の名前または IP アドレス。	はい (HASession が 1 に設定されている場合)	空
サーバ	Adaptive Server の名前または IP アドレス。	はい	空
ServerInitiated Transactions	SQL_ATTR_AUTOCOMMIT に 1 を設定した場合、Adaptive Server は必要に応じてトランザクションの管理を開始する。ドライバは、接続に対して set chained on コマンドを発行する。古い ODBC ドライバはこの機能を使用せず、begin tran を呼び出してトランザクションを開始するジョブの管理もしない。古い動作を管理するか、または接続で「連鎖」トランザクション・モードを使用しないことを要求するには、このプロパティに 0 を設定する。	いいえ	1
ServerPrincipal	KDC (Key Distribution Center) 内で設定された論理名または Adaptive Server 名。Adaptive Server ODBC ドライバはこの情報を使用して、設定された KDC および Adaptive Server サーバと Kerberos 認証をネゴシエートする。	いいえ	空
ServiceName	ホストとの接続に使用するサービス名を指定する。ServiceName には任意の文字列値を指定できます。 「z/OS オプションに対する Mainframe Connect および DirectConnect のサポート」 (72 ページ) を参照。	いいえ	空

プロパティ名	説明	必須	デフォルト値
SuppressParamFormat	セッションで準備文が再実行されるとき、パラメータ・フォーマット・メタデータを抑制するように Adaptive Server を指定する。 値： • 0 – パラメータ・フォーマット・メタデータは抑制されない。 • 1 – デフォルト値。Adaptive Server は、可能な場合にパラメータ・フォーマット・メタデータを送信しない。 「パラメータ・フォーマット・メタデータの抑制」(104 ページ) を参照。	いいえ	1
SuppressRowFormat	セッションで再実行されるクエリに対して、ロー・フォーマット・メタデータ (TDS_ROWFMT または TDS_ROWFMT2) を抑制するように Adaptive Server を指定する。 値： • 0 – ロー・フォーマット・メタデータは抑制されない。 • 1 – デフォルト値。Adaptive Server は、可能な場合にロー・フォーマット・メタデータを送信しない。 「ロー・フォーマット・メタデータの抑制」(104 ページ) を参照。	いいえ	1
SuppressRowFormat2	Adaptive Server が可能な場合には TDS_ROWFMT2 バイト・シーケンスではなく TDS_ROWFMT バイト・シーケンスを使用してデータを送信することを指定する。 値： • 0 – デフォルト値。TDS_ROWFMT2 は抑制されない。 • 1 – サーバがデータをできる限り TDS_ROWFMT で送信するように強制する。 「追加のロー・フォーマットの抑制に関する情報」(103 ページ) を参照。	いいえ	0
SupressTDSControl Tokens	設定すると、サーバは TDS コントロール・トークンを送信しない。	いいえ	0

プロパティ名	説明	必須	デフォルト値
TextSize	ネットワークで送信可能なバイナリまたはテキスト・データの最大サイズ。	いいえ	空 - Adaptive Server のデフォルトは 32 K。
TightlyCoupled Transaction (Microsoft Windows のみ)	分散トランザクションを使用するときに、同一の Adaptive Server サーバに接続している 2 つの DSN を使用している場合は、このプロパティを 1 に設定する。「第 3 章 サポートされている Adaptive Server の機能」の「分散トランザクションの使用」(63 ページ) を参照。	いいえ	0
TrustedFile	Encryption を ssl に設定した場合は、このプロパティに信頼されたファイルへのパスを設定する。	いいえ	空
UID、UserID	Adaptive Server への接続に必要なユーザ ID。大文字と小文字を区別する。	はい	空
UseCursor	結果セットを生成する SQL 文にどのカーソル・タイプを使用するかを指定する。 • 0 ーすべての状況でサーバ側カーソルを使用。 • 1 ーすべての状況でクライアント側カーソルを使用。 • 2 ー SQLSetCursorName ODBC 関数が呼び出された場合にのみサーバ側カーソルを使用。 「UseCursor 接続プロパティ」(19 ページ) を参照。	いいえ	0

ODBC ドライバのバージョン情報ユーティリティ

odbcversion ユーティリティは、ODBC ドライバに関する情報を表示します。

構文

odbcversion
-version |
-fullversion |
-connect *dsn userid password*

パラメータ

-version

ODBC ドライバの単純な数値のバージョン文字列を表示します。

-fullversion

ODBC ドライバの冗長バージョン文字列を表示します。

-connect *dsn userid password*

Adaptive Server のバージョンと、その Adaptive Server にインストールされている ODBC および OLEDB MDA スクリプトのバージョンを表示します。このパラメータには 3 つの変数が必要です。これらは、Adaptive Server のデータ・ソース名である *dsn* と、Adaptive Server への接続に使用されるユーザ ID およびパスワードです。

例

Adaptive Server への接続に使用される ODBC ドライバの単純な数値のバージョン文字列を取得します。

```
odbcversion -version
```

数値バージョン文字列が返されます。

```
15.05.00.1015
```

使用法

パラメータが指定されていない場合、odbcversion ユーティリティは有効なパラメータのリストを表示します。

第 3 章

サポートされている Adaptive Server の機能

この章では、Adaptive Server ODBC ドライバで使用できる Adaptive Server の高度な機能について説明します。

マイクロ秒の精度の time データ

Adaptive Server ODBC ドライバは、SQL データ型の bigdatetime と bigtime をサポートすることで、マイクロ秒レベルの精度の time データを提供します。

bigdatetime と bigtime は同様に機能し、SQL データ型の datetime および time とデータ・マッピングが同じです。

- bigdatetime は、Adaptive Server のデータ型 bigdatetime に対応し、0000 年 1 月 1 日の 00:00:00.000000 から経過したマイクロ秒数を示します。有効な bigdatetime 値の範囲は、0001 年 1 月 1 日の 0:00:00.000000 から 9999 年 12 月 31 日の 23:59:59.999999 までです。
- bigtime は、Adaptive Server のデータ型 bigtime に対応し、当日の午前 0 時ちょうどから経過したマイクロ秒数を示します。有効な bigtime 値の範囲は、00:00:00.000000 から 23:59:59.999999 までです。

使用法

- Adaptive Server 15.5 への接続時に、Adaptive Server ODBC ドライバは bigdatetime および bigtime データ型を使用してデータを転送します。受信した Adaptive Server カラムが datetime および time として定義されている場合でも同様です。

  これは、Adaptive Server は、Adaptive Server カラムに合わせるために、Adaptive Server ODBC ドライバから取得した値を暗黙的にトランケートする可能性があることを意味します。たとえば、bigtime の値 23:59:59.999999 は、time データ型の Adaptive Server カラムに 23:59:59.996 として保存されます。
- Adaptive Server 15.0.x 以前のバージョンへの接続時には、Adaptive Server ODBC ドライバは datetime および time データ型を使用してデータを転送します。

ODBC での非同期実行

デフォルトでは、ドライバは同期をとりながら ODBC 関数を実行します。つまり、アプリケーションが関数を呼び出し、実行が完了するとドライバからアプリケーションに制御が戻ります。非同期実行では、最小限の処理の後、実行が完了する前にドライバからアプリケーションに制御が戻ります。これによって、アプリケーションでは、最初の関数がまだ実行中であるときに他の関数を並列に実行できます。非同期実行は、タスクが複雑で実行にかなりの時間を要する場合に効果的です。

Sybase 製 Adaptive Server ODBC ドライバは、非同期モードで最大 1 つの同時文をサポートします。サーバ側カーソルが使用されている場合、または接続の自動コミットが無効になっている場合は、同期か非同期かにかかわらず、同時文を 1 つだけ実行できます。

Sybase 製 Adaptive Server ODBC ドライバで接続レベルの非同期実行を使用するには、SQLSetConnectAttr を呼び出し、*SQL_ATTR_ASYNC_ENABLE* を SQL_ASYNC_ENABLE_ON に設定します。

非同期実行と非同期実行アプリケーションの詳細については、『ODBC Programmer’s Reference』(Microsoft Developers Network (http://msdn.microsoft.com/) にあります) を参照してください。

注意　何も処理が実行されていないときに SQLCancel を呼び出した場合、関連するカーソルは閉じられません。ODBC アプリケーションでは、SQLFreeStmt または SQLCloseCursor を明示的に呼び出してカーソルを閉じる必要があります。

サポートされている Adaptive Server クラスタ・エディションの機能

この項では、クラスタ・エディション環境をサポートする Adaptive Server ODBC ドライバの機能について説明します。クラスタ・エディション環境では、複数の Adaptive Server が共有ディスクのセットと高速プライベート相互接続に接続します。この場合、複数の物理ホストと論理ホストを使用して、Adaptive Server を拡張できます。

Adaptive Server Enterprise の『クラスタ・ユーザ・ガイド』を参照してください。

ログインのリダイレクト

クラスタ・エディション環境では一般に、常にサーバ間で処理負荷の不均衡が発生しています。ビジー状態のサーバに対してクライアント・アプリケーションが接続を試みた場合、ログインのリダイレクト機能によって、サーバの負荷バランスが調整されます。具体的には、クラスタ内の負荷が少ない別サーバに対して、クライアント接続がリダイレクトされます。ログインのリダイレクトが発生するのはログイン・シーケンス中であり、リダイレクトが発生したことは、クライアント・アプリケーションには通知されません。ログインのリダイレクト機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意 クライアントをリダイレクトするように設定されているサーバに対してクライアント・アプリケーションが接続すると、ログインに時間がかかる場合があります。これは、クライアント接続が別サーバにリダイレクトされるたびに、ログイン・プロセスが再開されるからです。

接続マイグレーション

接続マイグレーション機能を使用すると、クラスタ・エディション環境内のサーバは動的に負荷を分散できます。さらに、既存のクライアント接続とそのコンテキストをクラスタ内の別サーバにシームレスにマイグレートできます。この機能によって、クラスタ・エディション環境では、最適なリソース配分と処理時間の短縮が実現します。サーバ間のマイグレーションはシームレスに行われるので、接続マイグレーション機能は、可用性が高い (HA: High Availability)「ダウン時間ゼロ」の環境を構築する場合にも役立ちます。接続マイグレーション機能をサポートしているサーバに対してクライアント・アプリケーションが接続した時点で、この機能は自動的に有効になります。

注意 接続マイグレーション中には、コマンドの実行に時間がかかる場合があります。状況に応じて、コマンドのタイムアウト値を増やすことをおすすめします。

クラスタ・エディションの接続フェールオーバ

接続フェールオーバ機能を使用すると、停電やソケットの障害など、予想外の原因でプライマリ・サーバが使用不可になった場合に、クライアント・アプリケーションは接続先を別の Adaptive Server に切り替えることができます。Adaptive Server クラスタ・エディションでは、クライアント・アプリケーションは動的なフェールオーバ・アドレスを使用して、複数のサーバに対して何度もフェールオーバできます。

高可用性に対応したシステムでは、フェールオーバ・ターゲットの候補をクライアント・アプリケーションにあらかじめ設定しておく必要はありません。Adaptive Server は、クラスタ・メンバシップ、論理クラスタの使用状況、負荷分散などに基づいて、最適なフェールオーバ・リストを常にクライアントに提供します。クライアントは、フェールオーバ時にフェールオーバ・リストの順序付けを参照して、再接続を試みます。ドライバがサーバに正常に接続した場合は、返されたリストに基づいて、ホスト値のリストが内部的に更新されます。それ以外の場合は、接続失敗例外が発生します。

「高可用性システムでのフェールオーバの使用」(83 ページ) を参照してください。

クラスタ・エディションの接続フェールオーバの有効化

Adaptive Server ODBC ドライバのユーザ・インタフェースの使用 (Windows のみ)

Adaptive Server ODBC ドライバでのクラスタ・エディション接続フェールオーバは、そのユーザ・インタフェースを使用して有効にできます。

❖ **ユーザ・インタフェースによる拡張フェールオーバの有効化**

1. [Adaptive Server Enterprise] ダイアログ・ボックスを表示します。
2. [接続] タブに移動します。
3. [高可用性の有効化] を選択します。
4. (オプション) [代替サーバ] フィールドに次の形式で代替サーバとポートを入力します。

```
server1:port1,server2:port2,...,serverN:portN;
```

接続を確立するとき、Adaptive Server ODBC ドライバは最初に、[Adaptive Server Enterprise] ダイアログ・ボックスの [一般] タブで定義されているプライマリ・ホストとポートに接続を試みます。Adaptive Server ODBC ドライバが接続を確立できない場合、[代替サーバ] フィールドのホストとポートのリストを検索し、次に指定されているホストとポートを試みます。

Adaptive Server ODBC ドライバの接続文字列の使用

接続文字列を使用して Adaptive Server ODBC ドライバの接続フェールオーバを有効にするには、HASession 接続文字列プロパティを 1 に設定します。SQLDriverConnect を使用して、接続文字列を指定できます。次に例を示します。

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;HASession=1;
AlternateServers=server2:port2,...,serverN:portN;
```

この例では、server1 と port1 をプライマリ・サーバとポートに定義します。Adaptive Server ODBC ドライバがプライマリ・サーバへの接続に失敗し、代替サーバが定義されている場合、[代替サーバ] フィールドで指定されたサーバとポートの順序リスト全体を検索し、接続を確立するかリストの最後になるまでサーバとポートを試みます。

unixODBC ドライバ・マネージャの使用 (UNIX のみ)

unixODBC ドライバ・マネージャにリンクしている場合は、unixODBC コマンド・ライン・ツールを使用して Adaptive Server ODBC データ・ソース・テンプレート *odbc.ini* を編集し、データ・ソースを再インストールします。

```
# odbcinst -i -s -f dsn_template_file
```

ここで、*dsn_template_file* は、Adaptive Server ODBC データ・ソース・テンプレート・ファイルへの完全なパスです。

Adaptive Server ODBC ドライバまたは Sybase iAnywhere ODBC ドライバ・マネージャの使用

Adaptive Server ODBC ドライバまたは Sybase iAnywhere ODBC ドライバ・マネージャに直接リンクする場合は、代替サーバを追加するために *odbc.ini* ファイルを変更します。次に例を示します。

```
ODBC Data Source UserID=sa
Password= Driver=Adaptive
Server Enterprise Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
HASession=1
AlternateServers=server2:port2,server3:port3;
```

注意 GUI または接続文字列で指定された代替サーバのリストは、初期接続時にのみ使用されます。使用可能なインスタンスとの接続の確立後、高可用性をサポートしているクライアントは、最適なフェールオーバ・ターゲットを含む最新のリストをサーバから受信します。この新しいリストは、指定されたリストを上書きします。

分散トランザクションの使用

ここでは、Adaptive Server ODBC ドライバを使用し、それを 2 フェーズ・コミット・トランザクションに含める方法について説明します。この機能は Microsoft Windows でのみサポートされ、2 フェーズ・コミットを管理するトランザクション・コーディネータとして Microsoft 分散トランザクション・コーディネータ (MS DTC) が使用されている必要があります。

Sybase は次のすべてのプログラミング・モデルをサポートしています。

- MS DTC を直接使用するアプリケーション
- Sybase EAServer を使用するアプリケーション
- MTS (Microsoft Transaction Server) または COM+ を使用するアプリケーション

MS DTC のプログラミング

❖ **Microsoft 分散トランザクション・コーディネータ (MS DTC) を使用したプログラミング**

1. DtcGetTransactionManager 関数を使用して MS DTC に接続します。MS DTC の詳細については、Microsoft 分散トランザクション・コーディネータのマニュアルを参照してください。
2. 確立する Adaptive Server の接続ごとに 1 回、SQLDriverConnect または SQLConnect を呼び出します。
3. ITransactionDispenser::BeginTransaction 関数を呼び出して MS DTC トランザクションを開始し、そのトランザクションを表す OLE トランザクション・オブジェクトを取得します。
4. MS DTC トランザクションに登録する ODBC 接続ごとに 1 回または複数回、SQLSetConnectAttr を呼び出します。SQLSetConnectAttr の呼び出し時には、手順 3 で取得したトランザクション・オブジェクトの SQL_ATTR_ENLIST_IN_DTC および ValuePtr 属性を指定する必要があります。
5. insert または update SQL 文ごとに 1 回または複数回、SQLExecDirect を呼び出します。

6 ITransaction::Commit 関数を呼び出して MS DTC トランザクションをコミットします。これでトランザクション・オブジェクトは無効になります。

一連の MS DTC トランザクションを実行するには、手順 3 ～ 6 を繰り返します。

トランザクション・オブジェクトへの参照を解放するには、ITransaction::Release 関数を呼び出します。

MS DTC トランザクションに使用している ODBC 接続をローカルの Adaptive Server トランザクションでも使用するには、SQL_DTC_DONE の ValuePtr を指定して SQLSetConnectAttr を呼び出し、トランザクションから接続の登録を解除します。

注意 手順 4 と 5 で示した呼び出し方法の代わりに、Adaptive Server ごとに別々に SQLSetConnectAttr と SQLExecDirect を呼び出すこともできます。

Sybase EAServer、MTS、または COM+ に展開されるコンポーネントのプログラミング

次の手順は、Sybase EAServer、MTS、または COM+ で分散トランザクションに関与するコンポーネントの作成方法を説明しています。

❖ **Sybase EAServer、MTS、または COM+ に展開されるコンポーネントのプログラミング**

1 確立する Adaptive Server 接続ごとに 1 回、SQLDriverConnect を呼び出します。

2 insert または update SQL 文ごとに 1 回、SQLExecDirect を呼び出します。

3 コンポーネントを MTS に展開し、必要に応じてトランザクション属性を設定します。

トランザクション・コーディネータは必要に応じて分散トランザクションを作成し、Adaptive Server ODBC ドライバを使用するコンポーネントがグローバル・トランザクションに自動登録されます。次に、分散トランザクションがコミットまたはロールバックされます。

分散トランザクションでの接続プロパティのサポート

ここでは、接続プロパティについて説明します。

- 分散トランザクション・プロトコル (DistributedTransactionProtocol) － 分散トランザクションをサポートするために使用されるプロトコルを指定するには、XA インタフェース標準または MS DTC OLE ネイティブ・プロトコルのいずれかを使用し、[ODBC データ・ソース] ダイアログで [**分散トランザクション・プロトコル**] を選択するか、接続文字列にプロパティ DistributedTransactionProtocol = *OLE* ネイティブ・プロトコルを設定します。デフォルトは *XA* です。
- 密結合トランザクション (TightlyCoupledTransaction) － 2 つのリソース・マネージャを使用する分散トランザクションで同一の Adaptive Server サーバを指定すると、「密結合トランザクション」になります。この場合、このプロパティを 1 に設定していないと分散トランザクションが失敗することがあります。

つまり、同一の Adaptive Server に対して 2 つのデータベース接続をオープンしてから、オープンした接続を同一の分散トランザクションに登録する場合は、TightlyCoupledTransaction=1 を設定する必要があります。このプロパティを設定するには、[ODBC データ・ソース] ダイアログ・ボックスで [密結合トランザクション] を選択するか、接続文字列でプロパティ TightlyCoupledTransaction=1 を渡します。

警告！ SQLSetConnectAttr に SQL_AUTOCOMMIT_OFF を指定して実行するか、SQLExecDirect を使用して BEGIN TRANSACTION 文を明示的に実行することにより、その接続で既にローカル・トランザクションを開始している場合、SQLSetConnectAttr を登録すると、SQL_ERROR が返されます。

ディレクトリ・サービスの使用

ディレクトリ・サービスを使用すると、Adaptive Server ODBC ドライバは中央にある LDAP サーバから接続やその他の情報を取得し、これらの情報を使用して Adaptive Server に接続できます。ここでは、DSURL (Directory Service URL) というプロパティを使用して、データを取得する LDAP サーバを示します。

ディレクトリ・サービスとしての LDAP

LDAP (Lightweight Directory Access Protocol) は、ディレクトリ・サービスへの業界標準のアクセス方法です。ディレクトリ・サービスを使用すると、コンポーネントは LDAP サーバから情報を DN (識別名) で検索できます。LDAP サーバは、企業またはネットワーク上で使用されるサーバ、ユーザ、ソフトウェアの情報を格納したり管理したりします。

LDAP は、クライアントとサーバが交換するメッセージの通信プロトコルと内容を定義します。LDAP サーバに格納され、取得が可能な情報は、次のとおりです。

- Adaptive Server に関する情報 (IP アドレス、ポート番号、ネットワーク・プロトコルなど)
- セキュリティ・メカニズムとフィルタ
- 高可用性コンパニオン・サーバ名

詳細については、Adaptive Server Enterprise の『システム管理ガイド』を参照してください。

LDAP サーバの設定時に、次のアクセス制限を使用できます。

- 匿名認証 − すべてのユーザがあらゆる情報にアクセスできます。
- ユーザ名とパスワードによる認証 − Adaptive Server は、次のファイルで指定されているデフォルトのユーザ名とパスワードを使用します。

ユーザ名とパスワードによる認証のプロパティによって、LDAP サーバとのセッション接続が確立され、終了します。

注意 LDAP サーバと Adaptive Server やクライアントのプラットフォームは異なっていてもかまいません。

ディレクトリ・サービスの使用

ディレクトリ・サービスを使用するには、ConnectString に次のプロパティを追加します。

```
DSURL=ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

URL は LDAP URL で、LDAP ライブラリを使用して URL を解決します。

LDAP サーバの高可用性をサポートするため、DSURL はセミコロンで区切られた複数の URL を受け入れます。

```
DSURL={ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO};
```

プロバイダは、LDAP サーバからプロパティを指定された順序で取得しようとします。次に例を示します。

```
ldap://hostport/dn[?attrs[?scope[?filter[?userdn?userp
ass]]]]
```

各パラメータの意味は次のとおりです。

- *hostport* は、ホスト名とオプションの `portnumber` です。たとえば、`SYBLDAP1:389` となります。
- *dn* は、`dc=sybase,dc-com` などの検索ベースです。
- *attrs* は、LDAP に要求される属性のカンマ区切りリストです。これはブランクにします。Data Provider はすべての属性を必要とします。
- *scope* は、次の 3 つの文字列のいずれかになります。
  - *base* (デフォルト) - ベースを検索する。
  - *one* - 直下の子を検索する。
  - *sub* ー サブツリーを検索する。
- *filter* は検索フィルタであり、通常は sybaseServername です。検索フィルタをブランクにする場合は、データ・ソースまたは ConnectionString のサーバ名のプロパティを設定します。
- *userdn* は、ユーザの識別名 (DN: Distinguished Name) です。LDAP サーバが匿名ログインをサポートしていない場合は、ここでユーザの DN を設定するか、ConnectionString の DSPrincipal プロパティを設定します。
- *userpass* はパスワードです。LDAP サーバが匿名ログインをサポートしていない場合は、ここでパスワードを設定するか、ConnectionString の DSPassword プロパティを設定します。

URL に *sybaseServername* を組み込むことも、Server Name プロパティを LDAP Sybase サーバ・オブジェクトのサービス名に設定することもできます。

次のプロパティは、ディレクトリ・サービスを使用する場合に役立ちます。

- DSURL － LDAP URL に設定します。デフォルトは空の文字列です。
- Server - LDAP Sybase サーバ・オブジェクトのサービス名。デフォルトは空の文字列です。
- DSPrincipal - LDAP サーバが DSURL の一部ではなく匿名アクセスが許可されない場合に、このサーバにログインするユーザ名。
- DSPassword または Directory Service Password - LDAP が DSURL の一部ではなく匿名アクセスが許可されない場合に、このサーバで認証するパスワード。

ディレクトリ・サービスの有効化

この項では、使用しているプラットフォームでディレクトリ・サービスを有効にする方法について説明します。

Microsoft Windows

❖ **Microsoft Windows でのディレクトリ・サービスの有効化**

1 ODBC データ・ソース・アドミニストレータを起動します。

2 使用するデータ・ソースを選択し、[設定] を選択します。

3 [接続] タブをクリックします。

4 [ディレクトリ・サービス情報] グループで、[URL] フィールドに完全な URL を指定します。また、LDAP サーバにログインするためのユーザ名を [ユーザ ID] フィールドに、LDAP サービス名を [サービス名] フィールドに入力することもできます。

Linux

❖ **Linux でのディレクトリ・サービスの有効化**

次のパッケージをインストールします。

- openldap-2.0 (ランタイム)
- openldap-devel-2.0

Adaptive Server ODBC ドライバは、*libldap.so* という名前のファイルをロードしようとしますが、このファイルでシンボリック・リンクを作成するには、openldap-devel パッケージをインストールする必要があります。openldap ランタイム・パッケージではシンボリック・リンクは作成されません。

unixODBC ドライバ・マネージャにリンクしている場合は、次の手順に従います。

1　Adaptive Server ODBC データ・ソース・テンプレート *odbc.ini* を編集します。

2　unixODBC コマンド・ライン・ツールを使用してデータ・ソースを再インストールします。

```
# odbcinst -i -s -f <dsn template file>
```

ここで、*dsn template file* は、Adaptive Server ODBC データ・ソース・テンプレート・ファイルへの完全なパスです。

Adaptive Server ODBC ドライバに直接リンクしている場合は、*odbc.ini* ファイルを修正します。次に例を示します。

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password= Driver=Adaptive
Server Enterprise Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
DSURL=ldap://SYBLDAP1:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

ブックマークとバルクのサポート

Sybase では、ODBC ドライバに対して、ブックマーク・オペレーションと SQL バルク・オペレーションをサポートします。

SQLBulkOperations を使用するバルク挿入では、SQL_ADD オプションが使用され、SQLSetPos (SQL_UPDATE、SQL_DELETE、SQL_POSITION) を使用してカーソルの位置付け更新と削除が行われます。SQL_ADD および SQLSetPos の使用方法については、*ODBC Programmer's Reference* (Microsoft Developer Network library (http://msdn.microsoft.com) にあります) を参照してください。

バルク・ロードのサポート

Adaptive Server ODBC ドライバでは、バルク・ロード・インタフェースがサポートされており、Adaptive Server への多数のローの高速な挿入が可能です。このインタフェースは、SQLBulkOperations が SQL_ADD オプションとともに使用され、EnableBulkLoad 接続プロパティが設定されているときに呼び出されます。次の 2 つのタイプのバルク・ロードがサポートされています。

- 配列挿入ーこのタイプのバルク・ロードは、単一または複数文トランザクション内で使用できます。データベース接続は、autocommit off に設定できます。
- バルク・コピーーこれは単一文トランザクションのみでサポートされ、次の条件を満たす必要があります。
  - データベース接続を autocommit on に設定する。
  - Adaptive Server の select into/bulkcopy オプションをオンにする。

  ターゲット・テーブルが高速バージョンの**バルク・コピー**の条件を満たす場合、Adaptive Server はこのバージョンの**バルク・コピー**を使用してローを挿入します。

注意 select into/bulkcopy オプションを有効にして、バルク・コピー・モードを使用する場合、データベースのリカバリ性に影響します。管理者は、バルク・コピーの処理が完了したら、今後のリカバリ性を保証するためにデータベースをダンプする必要があります。

次の表は、使用すべきバルク・ロードについて説明しています。

表 3-1：バルク・ロード・オプションの使用方法

使用状況	その他の注意事項	使用するバルク・ロード・オプション	注意
1 つまたは少数のローの挿入。		なし	

使用状況	その他の注意事項	使用するバルク・ロード・オプション	注意
多数のローを含むバッチの挿入。	バッチは、複数文トランザクションに含まれる。	配列挿入	バルク・ロードが無効な場合よりも、ローが高速に挿入される。
	Adaptive Server の select into または bulkcopy オプションは、リカバリ性の考慮により選択できない。	配列挿入	バルク・ロードが無効な場合よりも、ローが高速に挿入される。
	バッチは単一トランザクションで、Adaptive Server select into/bulkcopy オプションは有効化されている。	バルク・コピー	Adaptive Server で、配列挿入よりも速い、高速バルク・コピーを使用可能。バルク・コピーのパフォーマンスは、高速バルク・コピーを使用しない場合でも、配列挿入よりもわずかに高速。

select into/bulkcopy を有効化した場合の影響と、高速またはログ出力**バルク・コピー**を使用するための条件については Adaptive Server Enterprise の『ユーティリティ・ガイド』を参照してください。

EnableBulkLoad 接続プロパティ

次の EnableBulkLoad 接続プロパティを使用して、バルク・ロード・サポートを有効化または無効化します。

- 0 ― デフォルト値。バルク・ロードを無効化します。
- 1 ―**配列挿入**を使用するバルク・ロードを有効化します。
- 2 ―**バルク・コピー**・インタフェースを使用するバルク・ロードを有効化します。
- 3 ―高速ログ出力**バルク・コピー**・インタフェースを使用するバルク・ロードを有効化します。

または、Sybase 固有の SQL_ATTR_ENABLE_BULK_LOAD 接続属性を使用して、EnableBulkLoad をプログラムによって設定します。この属性は EnableBulkLoad と同じ値を受け入れます。次に例を示します。

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_ENABLE_BULK_LOAD,
(SQLPOINTER)3, SQL_IS_INTEGER);
```

パフォーマンスの考慮事項

この機能ではサーバを設定する必要は特にありませんが、より大きなページ・サイズやネットワーク・パケット・サイズにより、パフォーマンスは大幅に向上します。クライアント・メモリに応じてバッチ・サイズを大きくすることでもパフォーマンスが向上します。

制限事項

トリガは、バルク・ロード用に選択されたテーブルでは無視されます。

バルク・ロードの有効化

- **ODBC データ・ソース・アドミニストレータのユーザ・インタフェースを使用したバルク・ロードの有効化**

  1 ODBC データ・ソース・アドミニストレータから、データ・ソース名 (DSN) の設定ウィンドウを開きます。

  2 [詳細設定] タブを選択します。

  3 [バルク・ロードの有効化] の下の適切なオプションを選択します。

  EnableBulkLoad の接続プロパティのデフォルト値は 0 です。これは、insert が使用されることを意味します。

- **ODBC 接続文字列を使用したバルク・ロードの有効化**

  1 SQLDriverConnect を使用して接続文字列を指定します。

  2 EnableBulkLoad 接続文字列のプロパティを 0、1、2、または 3 の任意の値に設定します。次に例を示します。

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;EnableBulkLoad=1;
```

z/OS オプションに対する Mainframe Connect および DirectConnect のサポート

Sybase の Adaptive Server ODBC ドライバは、ServiceName および BackEndType 接続プロパティによって、z/OS オプションに対して Mainframe Connect DirectConnect™ をサポートします。

ServiceName 接続プロパティ

ServiceName プロパティは、ホストに接続するために使用するサービス名を指定します。ServiceName には任意の文字列値を指定できます。デフォルト値は空の文字列 ("") です。

BackEndType 接続プロパティ

BackEndType 接続プロパティは、定義している DSN のターゲットの種類を指定します。ODBC ドライバは、Adaptive Server などのデータベース・システムや Sybase 以外のデータベース・システムへのゲートウェイなど、複数のターゲットと通信できます。現在、Adaptive Server ODBC ドライバは次のバックエンドの種類をサポートします。

- ASE (デフォルト)
- MFC Gatewayless
- DC DB2 Access Service
- DC TRS
- Replication Server

Replication Server の接続サポート

Adaptive Server Enterprise ODBC ドライバを Replication Server に接続し、このサーバの設定および管理を行うことができます。ODBC ドライバから送信される有効な Replication Server 管理コマンドのみが Replication Server でサポートされます。Replication Server 接続のために、BackEndType 接続プロパティを `Replication Server` に設定します。

DSN マイグレーション・ツール

ODBC DSN マイグレーション・ツールを使用して、Data Direct ODBC ドライバから Sybase 製 Adaptive Server ODBC ドライバにデータ・ソースをマイグレートできます。

マイグレーション・ツールの使用

dsnmigrate ツールでは、どの DSN をマイグレートするかを制御するスイッチを使用します。コマンド・ラインで次のように入力します。

```
dsnmigrate.exe [/?|/help] [l|/ul|/sl][/a|/ua|/sa]
   [[/dsn|/udsn|/sdsn]=dsn] [/suffix=suffix]
```

変換されるすべての DSN には、変換が完了する前に "<dsn>-backup" という名前が付けられます。新しい Sybase DSN が作成され、変換が完了すると、名前が "<dsn>" に変更されます。これにより、既存のアプリケーションを変更せずに継続して実行できます。

変換スイッチ

表 3-2 に、変換で使用されるスイッチを示し、各スイッチについて説明します。

表 3-2：変換スイッチ

スイッチ	結果の説明
/?、/h、/help	スイッチとその説明のリストを示す。コマンド・ライン引数を指定しないで dsnmigrate を呼び出したときにも、同じリストが表示される。
/l	Sybase Data Direct のすべてのユーザ DSN とシステム DSN を一覧表示する。
/ul	Sybase Data Direct のすべてのユーザ DSN を一覧表示する。
/sl	Sybase Data Direct のすべてのシステム DSN を一覧表示する。
/a	Sybase Data Direct のすべてのユーザ DSN とシステム DSN を変換する。
/ua	Sybase Data Direct のすべてのユーザ DSN を変換する。
/sa	Sybase Data Direct のすべてのシステム DSN を変換する。
/dsn	Sybase Data Direct の特定のユーザ DSN またはシステム DSN を変換する。
/udsn	Sybase Data Direct の特定のユーザ DSN を変換する。
/sdsn	Sybase Data Direct の特定のシステム DSN を変換する。
dsn	変換される DSN の名前。
/suffix	DSN に名前を付ける方法を変更するオプション・スイッチ。このスイッチを使用すると、元の DSN が保持され、新しい DSN に "<dsn>-<suffix>" という名前が付けられる。
suffix	新しい DSN に名前を付けるために使用されるサフィックス。

パスワードの暗号化

Adaptive Server ODBC ドライバはデフォルトで、ネットワークを介してプレーン・テキストのパスワードを Adaptive Server に送信して認証を求めます。ただし、Adaptive Server ODBC ドライバは、パスワードの対称／非対称暗号化もサポートしています。これによってデフォルトの動作を変更し、パスワードを暗号化してからネットワークに送信できます。

対称暗号化メカニズムでは、パスワードの暗号化と復号化に同じキーが使用されます。これに対して、非対称暗号化メカニズムでは、暗号化にはパブリック・キー、復号化には別のプライベート・キーが使用されます。プライベート・キーはネットワークを介して共有されないため、非対称暗号化の方が対称暗号化よりも安全であると考えられます。パスワードの暗号化が有効になっていて、サーバが非対称暗号化をサポートしている場合、非対称暗号化が対称暗号化の代わりに使用されます。

Sybase CSI (Common Security Infrastructure) を使用して、ログイン・パスワードとリモート・パスワードを暗号化できます。CSI 2.6 は、連邦情報処理標準 (FIPS: Federal Information Processing Standard) 140-2 に準拠しています。

パスワードの暗号化の有効化

パスワードの暗号化を有効にするには、EncryptPassword 接続プロパティを設定する必要があります。この接続プロパティでは、パスワードが暗号化フォーマットで転送されるかどうかを指定します。パスワードの暗号化が有効になっていると、ログインがネゴシエートされた場合にのみ、パスワードはネットワークに送信されます。パスワードは最初に暗号化されてから送信されます。EncryptPassword の値は次のとおりです。

- 0 ― プレーン・テキスト形式のパスワードを使用します。これはデフォルトの値です。
- 1 ― 暗号化されたパスワードを使用します。暗号化がサポートされていない場合、エラー・メッセージを返します。

- 2 – 暗号化されたパスワードを使用します。暗号化がサポートされていない場合、プレーン・テキスト形式のパスワードを使用します。

注意 パスワードの暗号化機能を使用するには、パスワードの暗号化をサポートするサーバ (Adaptive Server 15.0.2 など) が必要です。非対称暗号化では、追加の処理時間が必要になり、ログインに若干の遅延が発生する可能性があります。

Microsoft Windows でのパスワード暗号化

❖ **Microsoft Windows でのパスワードの暗号化**

1 ODBC データ・ソース・アドミニストレータを起動します。

2 使用するデータ・ソースを選択し、[設定] を選択します。

3 [詳細設定] タブをクリックします。

4 EncryptPassword を選択します。

`SQLDriverConnect` に対する呼び出しで EncryptPassword 接続プロパティを使用できます。

注意 ユーザ・インタフェースを使用して EncryptPassword を 0 または 1 に設定します。EncryptPassword を 2 に設定するには、接続文字列を使用します。

UNIX でのパスワード暗号化

unixODBC ドライバ・マネージャにリンクするには、unixODBC コマンド・ライン・ツールを使用してデータ・ソース・テンプレートを編集し、データ・ソースを再インストールします。

```
# odbcinst -i -s -f dsn template file
```

ここで、*dsn template file* は、Adaptive Server ODBC データ・ソース・テンプレート・ファイルへの完全なパスです。

Adaptive Server ODBC ドライバまたは Sybase iAnywhere ODBC ドライバ・マネージャに直接リンクしている場合は、*odbc.ini* ファイルを修正します。

次に、*odbc.ini* データ・ソース・テンプレート・ファイルの例を示します。

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
EncryptPassword=1
```

パスワード有効期限の処理

各企業は、自社のデータベース・システム用に独自のパスワード・ポリシーを設定しています。ポリシーに応じて、パスワードは特定の日時で期限切れになります。パスワードがリセットされない限り、データベースに接続した Adaptive Server ODBC ドライバはパスワード期限切れエラーを発生させ、isql を使用してパスワードを変更するようユーザに要求します。パスワード有効期限の処理機能によって、Adaptive Server ODBC ドライバを使用して期限切れのパスワードを変更できます。

接続文字列プロパティによるパスワードの変更

次の 2 つの接続文字列プロパティを設定します。

- OldPassword － 現在のパスワード。OldPassword に null や空の文字列以外の値が含まれている場合、現在のパスワードは PWD に含まれる値に変更される。
- PWD －パスワードの値が含まれる。OldPassword が設定されて null 以外の場合、PWD には現在のパスワード値が含まれる。OldPassword が存在しないか null の場合、PWD には新しいパスワード値が含まれる。

ダイアログ・ボックスによるパスワードの変更

[パスワードの変更] ダイアログは、"SQLDriverConnect with SQL_DRIVER_PROMPT" が true に設定されているときに有効になります。このダイアログで、現在のパスワードと新しいパスワードを入力します。

SSL の使用

SSL (Secure Sockets Layer) は、クライアントとサーバ間、およびサーバ同士の接続において、ワイヤ・レベルまたはソケット・レベルで暗号化されたデータを送信する業界標準です。サーバとクライアントが、安全な暗号化セッションをネゴシエートして合意してから、SSL 接続が確立されます。これは、"SSL ハンドシェイク " と呼ばれています。

注意　安全なセッションの確立に追加のオーバヘッドが必要です。データを暗号化するとサイズが増え、情報の暗号化と復号化に追加の計算も必要になるためです。通常の状況では、SSL ハンドシェイク中に生じる I/O の増加によって、ユーザ・ログインにかかる時間が 10 ～ 20 倍になることがあります

SSL ハンドシェイク

クライアント・アプリケーションが接続を要求すると、SSL 対応サーバは証明書を提示し、ID を証明してから、データを送信します。基本的に、SSL ハンドシェイクは次の手順によって構成されています。

1　クライアントがサーバに接続要求を送信します。要求には、クライアントがサポートしている SSL (または TLS: Transport Layer Security) オプションが含まれています。

2　サーバは、自身の証明書と、サポートされている暗号スイートのリストを返す。このリストには、SSL/TLS サポート・オプション、キー交換で使用するアルゴリズム、デジタル署名が含まれます。暗号スイートは、SSL プロトコルで使用されるキー交換アルゴリズム、ハッシュ方式、暗号化方式の優先順位リストです。

3　クライアントとサーバの両者が 1 つの暗号スイートについて合意すると、安全で暗号化されたセッションが確立されます。

暗号スイート

SSL ハンドシェイク中に、クライアントとサーバは、暗号スイートを介して共通のセキュリティ・プロトコルをネゴシエートします。

デフォルトでは、クライアントとサーバの両方がサポートしている最も強力な暗号スイートは、SSL ベースのセッションに使用される暗号スイートです。サーバ接続属性は、接続文字列か、LDAP などのディレクトリ・サービスによって指定されます。

Adaptive Server ODBC ドライバと Adaptive Server は、SSL Plus ライブラリ API と暗号エンジンである Security Builder (両方とも Certicom 製) で使用可能な暗号スイートをサポートしています。

注意　次のリストの暗号スイートは、TLS (Transport Layer Security) の仕様に準拠しています。TLS は、SSL 3.0 を拡張したものであり、SSL バージョン 3.0 CipherSuite の別名です。

次に、Adaptive Server ODBC ドライバでサポートされる暗号スイートを最も強力なものから順に示します。

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_RC4_128_MD5
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_RC4_128_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_DHE_DSS_WITH_DES_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

SSL ハンドシェイクと SSL/TLS プロトコルの詳細については、Internet Engineering Task Force Web site (http://www.ietf.org) を参照してください。

暗号スイートの詳細については、IETF organization Web site (http://www.ietf.org/rfc/rfc2246.txt) を参照してください。

Adaptive Server ODBC ドライバの SSL セキュリティ・レベル

Adaptive Server ODBC ドライバでは、SSL は次のセキュリティ・レベルを提供します。

- SSL セッションが確立されると、ユーザ名とパスワードが暗号化された安全な接続によって送信されます。
- SSL 対応サーバへの接続を確立すると、サーバは接続対象のサーバであることを自己認証し、暗号化された SSL セッションが開始され、データが送信されます。
- サーバ証明書のデジタル署名を比較して、サーバから受信したデータが転送中に変更されたかどうかを判断します。

証明書によるサーバの検証

Adaptive Server ODBC ドライバが SSL 対応サーバにクライアント接続する場合、サーバは証明書ファイルが必要です。これは、サーバの証明書と暗号化されたプライベート・キーで構成されます。また、証明書は署名 / 認証局 (CA: Certification Authority) によってデジタル署名されている必要もあります。Adaptive Server ODBC ドライバのクライアント・アプリケーションが Adaptive Server へのソケット接続を確立する方法は、既存のクライアント接続の確立方法と似ています。ネットワーク・トランスポート・レベルの接続コールがクライアント側で完了し、承認コールがサーバ側で完了すると、ソケット上で SSL ハンドシェイクが行われ、その後でユーザ・データが送信されます。

SSL 対応サーバに正しく接続するには、次の手順に従ってください。

1 クライアント・アプリケーションが接続要求を行った場合、SSL 対応サーバは証明書を提示しなければなりません。

2 クライアント・アプリケーションは、証明書に署名した CA を認識しなければなりません。「信頼された」CA すべてを含んだリストは、「信頼されたルート・ファイル」にあります。

信頼されたルート・ファイル

既知で信頼された CA のリストは、信頼されたルート・ファイルに保管されています。エンティティ (クライアント・アプリケーション、サーバ、ネットワーク・リソースなど) に既知の CA の証明書がある以外は、信頼されたルート・ファイルは証明書ファイルのフォーマットと同じです。システム・セキュリティ担当者が、標準 ASCII テキスト・エディタを使って、信頼された CA を追加したり、削除したりします。

アプリケーション・プログラムでは、ConnectString の TrustedFile=*trusted file path* プロパティを使用して、信頼されたルート・ファイルの位置を指定します。最も一般的に使用される CA (thawte、Entrust、Baltimore、VeriSign、RSA) が記載された信頼されたルート・ファイルは *$SYBASE/config/trusted.txt* にインストールされています。

証明書の詳細については、『Open Client Client-Library C リファレンス・マニュアル』を参照してください。

SSL 接続の有効化

Adaptive Server ODBC ドライバで SSL を有効化するには、ConnectString に Encryption=ssl と TrustedFile=<*filename*> を追加します (*filename* は*信頼されたルート・ファイル*へのパスです)。これで、Adaptive Server ODBC ドライバが Adaptive Server と SSL 接続をネゴシエートするようになります。

注意　SSL を使用するように、Adaptive Server を設定してください。SSL の詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。

Microsoft Windows

SSL を有効にする前に、接続文字列内の TrustedFile プロパティに信頼されたルート・ファイルのファイル名を設定する必要があります。ファイル名には、そのファイルへのパスも含める必要があります。

❖ **SSL 接続の有効化**

1　接続文字列内の Encryption プロパティを ssl に設定します。

2　ODBC データ・ソース・アドミニストレータを起動します。

3　使用するデータ・ソース名 (DSN) を選択し、[設定] を選択します。

4　[接続] タブをクリックします。

5　Secure Socket Layer グループで [SSL 暗号化の使用] を選択します。

6　TrustedFile フィールドに、信頼されたルート・ファイルの完全なパスを指定します。

UNIX

❖ **SSL 接続の有効化**

1 unixODBC ドライバ・マネージャの odbcinst ユーティリティを起動します。

2 既存のデータ・ソース・テンプレートを開くか、新しいテンプレートを作成します。

3 データ・ソース・テンプレートに次を追加します。

```
Encryption=ssl

TrustedFile=<filename>line
```

4 次のコマンドでデータ・ソースを再インストールします。

```
# odbcinst -i -s -f dsn template file
```

ここで、*dsn template file* は、Adaptive Server ODBC データ・ソース・テンプレート・ファイルへの完全なパスです。

Adaptive Server ODBC ドライバまたは Sybase iAnywhere ODBC ドライバ・マネージャに直接リンクしている場合は、*odbc.ini* ファイルを修正します。

次に、*odbc.ini* データ・ソース・テンプレート・ファイルの例を示します。

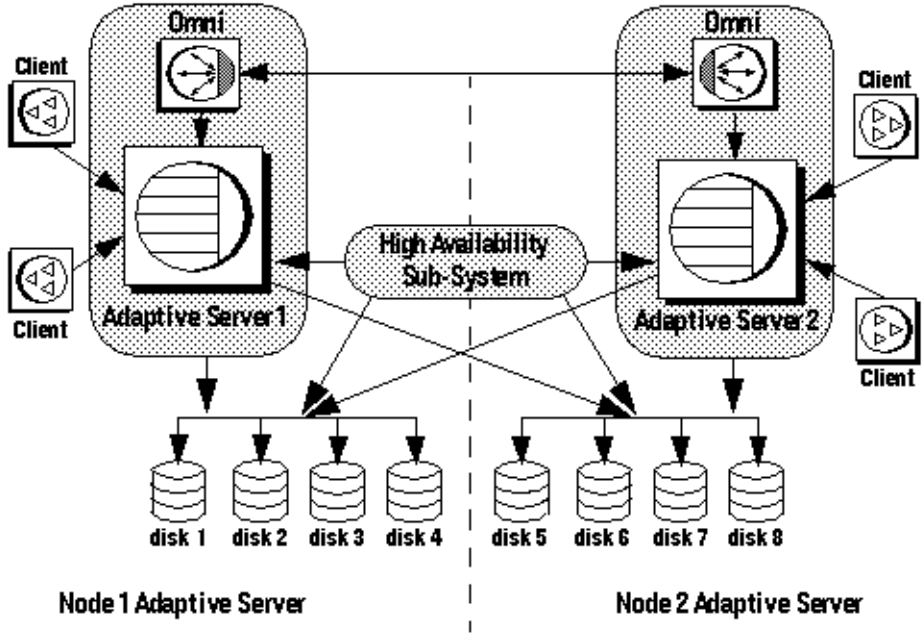
```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
Encryption=ssl
TrustedFile=<SYBASE>/config/trusted.txt
```

高可用性システムでのフェールオーバの使用

高可用性クラスタには、2 つ以上のマシンが含まれます。これらのマシンは、1 つのマシン (またはアプリケーション) が中断した場合にもう 1 つのマシンが両方のマシンの負荷を処理するように設定されています。このようなマシンのそれぞれを、高可用性クラスタのノードといいます。高可用性クラスタはシステムが常に稼働していなければならないような環境で使用します。たとえば、クライアントが 1 年 365 日絶えず接続する銀行のシステムなどです。

図 3-1 のマシンは、他のマシンのディスクを各マシンで読み込めるように設定されています。ただし、同時読み込みはできません (フェールオーバで使用するすべてのディスクは共有ディスクに設定してください)。

図 3-1: フェールオーバを使用する高可用性クラスタ



たとえば、プライマリ・コンパニオン・サーバである Adaptive Server 1 が失敗した場合、セカンダリ・コンパニオンである Adaptive Server 2 は、Adaptive Server 1 を再起動できるようになるまでそのディスク (ディスク 1 ～ 4) を読み込んで、ディスク上のデータベースすべてを管理します。Adaptive Server 1 に接続していたクライアントは、自動的に Adaptive Server 2 に接続されます。

フェールオーバによって、Adaptive Server をアクティブ／アクティブ設定またはアクティブ／パッシブ設定の高可用性クラスタで運用できます。

フェールオーバが発生すると、プライマリ・コンパニオンに接続していたクライアントは、フェールオーバ・プロパティを使用して、自動的にセカンダリ・コンパニオンへのネットワーク接続を再確立します。フェールオーバを有効にするには、接続プロパティ HASession を "1" (デフォルト値は "0") に設定します。このプロパティを設定しないと、サーバでフェールオーバが設定されていても、セッションではフェールオーバが行われません。SecondaryServer (セカンダリ Adaptive Server の IP アドレスまたはマシン名) と SecondaryPort (セカンダリ Adaptive Server サーバのポート番号) のプロパティも設定する必要があります。使用するシステムの高可用性設定の詳細については、Adaptive Server Enterprise のマニュアル『高可用性システムにおける Sybase フェールオーバの使用』を参照してください。

Adaptive Server ODBC ドライバでプライマリ Adaptive Server サーバの接続エラーが検出されると、最初にプライマリ・サーバへの再接続が試行されます。再接続できない場合はフェールオーバが行われたと見なされます。次に、SecondaryServer と SecondaryPort に設定された接続プロパティを使用して、セカンダリ Adaptive Server への接続が自動的に試行されます。

フェールオーバの成功を確認する方法

セカンダリ・サーバへの接続が確立されると、Adaptive Server ODBC ドライバは関数のリターン・コードの SQL_ERROR を返します。フェールオーバが成功したかどうか確認するには、SQLState の値が "08S01"、NativeError メッセージの値が "30130" になっているかを調べます。このようなフェールオーバでは、次のようなエラー・メッセージが返されます。

```
"Connection to Sybase server has been lost, you have
been successfully connected to the next available HA
server. All active transactions have been rolled back."
```

これらの値にアクセスするには、StatementHandle で SQLGetDiagRec を呼び出します。クライアントは、新しい接続を使用して、失敗したトランザクションを再適用しなければなりません。トランザクションのオープン中にフェールオーバが発生した場合、フェールオーバ前にデータベースにコミットされた変更のみが保持されます。

フェールオーバの失敗の確認

セカンダリ・サーバへの接続が確立されない場合、Adaptive Server ODBC ドライバは関数のリターン・コードの SQL_ERROR を返します。フェールオーバが行われていないことを確認するには、SQLState の値が “08S01”、NativeError の値が “30131” になっているか調べる必要がありますフェールオーバが失敗した場合、次のようなエラー・メッセージが返されます。

```
"Connection to Sybase server has been lost,
connection to the next available HA server
also failed.All active transactions
have been rolled back".
```

これらの値にアクセスするには、StatementHandle で SQLGetDiagRec を呼び出します。

次にフェールオーバのコーディング方法について示します。

```
/* Declare required variables */
....
/* Open Database connection */
....
/* Perform a transaction */
...
/* Check return code and handle failover */
if( retcode == SQL_ERROR )
{
   retcode = SQLGetDiagRec(stmt, 1,
      sqlstate,&NativeError, errmsg,100, NULL );
   if(retcode == SQL_SUCCESS ||
      retcode == SQL_SUCCESS_WITH_INFO)
   {
        if(NativeError == 30130 )
        {
        /* Successful failover retry transaction*/
        ...
        }
     else if (NativeError == 30131)
     {
        /* Failover failed.Return error */
        ...
     }
   }
}
```

Microsoft Windows

❖ **Microsoft Windows でのフェールオーバの使用**

1 ODBC データ・ソース・アドミニストレータを起動します。

2 使用するデータ・ソースを選択し、[設定] を選択します。

3 [接続] タブをクリックします。

4 [高可用性情報] グループで [高可用性の有効化] を選択します。

5 [サーバ名] フィールドでフェールオーバ・サーバ名を指定します。

6 [サーバ・ポート] フィールドでフェールオーバ・ポートを指定します。

UNIX

unixODBC ドライバ・マネージャにリンクしている場合は、unixODBC コマンド・ライン・ツールを使用してデータ・ソース・テンプレートを編集し、データ・ソースを再インストールします。

```
# odbcinst -i -s -f dsn template file
```

ここで、*dsn template file* は、Adaptive Server ODBC データ・ソース・テンプレート・ファイルへの完全なパスです。

Adaptive Server ODBC ドライバまたは Sybase iAnywhere ODBC ドライバ・マネージャに直接リンクしている場合は、*odbc.ini* ファイルを修正します。

次に、*odbc.ini* データ・ソース・テンプレート・ファイルの例を示します。

```
[sampledsn]
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
UserID=sa
Password=
Database=pubs2
HASession=1
SecondaryHost=failoverserver
SecondaryPort=5000
```

Kerberos による認証

Kerberos は、簡単なログイン認証と相互のログイン認証を提供する業界標準のネットワーク認証システムです。Kerberos を使用して、さまざまなアプリケーションにわたるシングル・サインオンを非常に安全な環境内で行えます。ネットワークの各所でパスワードを渡す代わりに、Kerberos サーバがユーザのパスワードの暗号化されたバージョンと使用可能なサービスの情報を保持します。

さらに Kerberos では、機密性とデータの整合性を維持するために暗号化を使用します。

Adaptive Server と Adaptive Server ODBC ドライバは、Kerberos 接続をサポートします。Adaptive Server ODBC ドライバは特に、MIT、CyberSafe、Active Directory の KDC (Key Distribution Center) をサポートします。

プロセスの概要

Kerberos 認証プロセスは次のように機能します。

1　クライアント・アプリケーションは、特定のサービスにアクセスするための「チケット」を Kerberos サーバに要求します。

2　Kerberos サーバは、2 つのパケットを含むチケットをクライアントに返します。第 1 のパケットはユーザ・パスワードにより暗号化されます。第 2 のパケットはサービス・パスワードにより暗号化されます。これらの各パケット内に「セッション・キー」が含まれます。

3　クライアントは、セッション・キーを取得するためにユーザ・パケットを復号化します。

4　クライアントは新しい認証パケットを作成し、それをセッション・キーにより暗号化します。

5　クライアントは、認証パケットとサービス・パケットをサービスに送信します。

6　サービスは、セッション・キーを取得するためにサービス・パケットを復号化し、ユーザ情報を取得するために認証パケットを復号化します。

7　サービスは、認証パケットからのユーザ情報と、サービス・パケットにも含まれているユーザ情報を比較します。両者が一致する場合、ユーザは認証済みです。

8 サービスは、認証パケットに含まれる検証データに加えてサービス固有の情報を含む確認パケットを作成します。

9 サービスは、このデータをセッション・キーとともに暗号化し、それをクライアントに返します。

10 クライアントは、パケットを復号化するために Kerberos から受信したユーザ・パケット内のセッション・キーを使用し、サービスがそれ自身の主張に一致しているかどうかを検証します。

こうした方法で、ユーザとサービスは相互に認証されます。以後、クライアントとサービス (この場合は Adaptive Server データベース・サーバ) の間の通信はすべて、セッション・キーにより暗号化されます。これにより、サービスとクライアント間で送信されるすべてのデータが望ましくない閲覧者から正しく保護されます。

稼働条件

認証システムとして Kerberos を使用するには、Kerberos に認証を委任するように Adaptive Server Enterprise を設定します。詳細については、Adaptive Server Enterprise の『システム管理ガイド』を参照してください。

Adaptive Server で Kerberos を使用するように設定されている場合、Adaptive Server と通信するすべてのクライアントが Kerberos クライアント・ライブラリをインストールする必要があります。これは、各種オペレーティング・システムのベンダごとに異なります。

- Microsoft Windows では、Active Directory クライアント・ライブラリがオペレーティング・システムとともにインストールされます。
- Microsoft Windows および Linux では、CyberSafe と MIT のクライアント・ライブラリを使用できます。

詳細については、ベンダのマニュアルを参照してください。

Kerberos 認証の有効化

Adaptive Server ODBC ドライバに対して Kerberos 認証を有効にするには、次の接続プロパティを追加します。

```
AuthenticationClient=<one of 'mitkerberos'
or 'cybersafekerberos' or 'activedirectory'>
and ServerPrincipal=<Adaptive Server name>
```

ここで <*Adaptive Server name*> は、KDC (Key Distribution Center) 内で設定された論理名またはプリンシパルです。Adaptive Server ODBC ドライバはこの情報を使用して、設定された KDC および Adaptive Server サーバと Kerberos 認証をネゴシエートします。

Kerberos クライアント・ライブラリは、さまざまな KDC 間で互換性を持ちます。たとえば、Linux では、KDC が Microsoft Active Directory であっても、mitkerberos と同じ AuthenticationClient を設定できます。

Kerberos クライアントで別のキャッシュ内の TGT (Ticket Granting Ticket) を検索する必要がある場合は、userprincipal プロパティを指定できます。

SQL_DRIVER_NOPROMPT とともに SQLDriverConnect を使用する場合、ConnectString は次のようになります。

```
"Driver=Adaptive Server Enterprise;UID=sa;
PWD='';Server=sampleserver;
Port=4100;Database=pubs2;
AuthenticationClient=mitkerberos;
ServerPrincipal=MANGO;"
```

Microsoft Windows

❖ **Microsoft Windows のログイン認証での Kerberos の有効化**

1. Microsoft Windows ODBC データ・ソース・アドミニストレータを起動します。
2. Sybase Adaptive Server Enterprise ODBC ドライバを選択します。
3. [ユーザー DSN] ／ [システム DSN] タブを選択し、変更するデータ・ソースをクリックするか、[追加] を選択します。
4. [セキュリティ] タブの [認証クライアント] で [Use Active Directory] を選択します。
5. [サーバのプリンシパル] 編集ボックスにサーバ・プリンシパルの名前を入力します。この名前は、KDC に設定された Adaptive Server の名前と一致する必要があります。

UNIX

❖ UNIX のログイン認証での Kerberos の有効化

UNIX ODBC ドライバ・マネージャにリンクしている場合は、次の手順に従います。

1 既存のデータ・ソースを開くか、新しいデータ・ソース・テンプレートを作成します。

2 次をデータ・ソース・テンプレートに追加します。

```
Authentication= mitkerberos
(or cybersafekerberos) ServerPrincipal=<MANGO>
to enable Kerberos Login Authentication.
```

各パラメータの意味は次のとおりです。*<MANGO>* は、サインオンの認証のために使用されるプリンシパル・サーバの名前です。

3 コマンド・ラインで odbcinst ユーティリティを使用してデータ・ソースを再インストールします。

```
odbcinst -i-s  -f ${datasourcetemplatefile}
```

Adaptive Server ODBC ドライバまたは Sybase iAnywhere ODBC ドライバ・マネージャに直接リンクしている場合は、*odbc.ini* ファイルを直接修正します。

次に、*odbc.ini* データ・ソース・テンプレート・ファイルを編集後の例を示します。

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
AuthenticationClient=mitkerberos
ServerPrincipal=MANGO
```

Key Distribution Center からの初期チケットの取得

Kerberos 認証を使用するには、Key Distribution Center から TGT (Ticket Granted Ticket) と呼ばれる初期チケットを生成します。このチケットを取得する手順は、使用する Kerberos ライブラリに応じて異なります。詳細については、ベンダのマニュアルを参照してください。

❖ **MIT Kerberos クライアント・ライブラリ用の TGT の生成**

1 コマンド・ラインに次のように入力して kinit ユーティリティを開始します。

```
% kinit
```

2 *your_name*@YOUR.REALM などの kinit ユーザ名を入力します。

3 my_password など、*your_name*@YOUR.REALM のパスワードを入力します。パスワードを入力すると、kinit ユーティリティにより TGT に対する要求が認証サーバに送信されます。

このパスワードは、キーの計算のために使用されます。そのキーは、応答の一部を復号化するために使用されます。この応答には、セッション・キーに加えて要求の確認が含まれます。パスワードを正しく入力していれば、この段階で TGT が取得されています。

4 コマンド・ラインに次のように入力して TGT が取得されていることを確認します。

```
% klist
```

klist コマンドの結果は次のようになるはずです。

```
Ticket cache:/var/tmp/krb5cc_1234
Default principal:your_name@YOUR.REALM
Valid starting       Expires              Service principal
24-Jul-95 12:58:02   24-Jul-95 20:58:15   krbtgt/YOUR.REALM@YOUR.REALM
```

結果の説明

チケット・キャッシュ チケット・キャッシュ・フィールドにより、どのファイルにクレデンシャル・キャッシュが含まれているかがわかります。

デフォルトのプリンシパル デフォルトのプリンシパルは、TGT を所有するユーザ (この場合はユーザ自身) のログインです。

有効な開始／期限／サービス・プリンシパル 以降の出力は、既存のチケットのリストです。これは要求した最初のチケットであるため、1 つのチケットのみがリストに含まれています。サービス・プリンシパル (krbtgt/YOUR.REALM@YOUR.REALM) は、このチケットが TGT であることを示しています。このチケットは、約 8 時間有効であることに注意してください。

ODBC ドライバ・マネージャのトレースなしのロギング

Adaptive Server ODBC ドライバでは、ODBC ドライバ・マネージャのトレースを使用せずに ODBC API の呼び出しをロギングできます。これは、ドライバ・マネージャが使用されないか、トレースをサポートしていないプラットフォームでドライバ・マネージャを実行している場合に便利です。

この機能を Microsoft Windows で有効にするには、LOGCONFIGFILE 環境変数または Microsoft Windows レジストリを使用します。Linux で有効にするには、LOGCONFIGFILE を使用します。

LOGCONFIGFILE を使用するときは、環境変数を ODBC ログの設定ファイルのフル・パスに設定します。LOGCONFIGFILE は既存のレジストレ・エントリをすべて上書きします。

Microsoft Windows レジストリを使用する場合は、LogConfigFile というエントリを *HKEY_CURRENT_USER ¥Software ¥Sybase ¥ODBC* または *HKEY_LOCAL_MACHINE ¥Software ¥Sybase ¥ODBC* に作成し、その値を ODBC ログの設定ファイルのフル・パスに設定します。次に例を示します。

```
Windows レジストリ・エディタ・バージョン 5.00

[HKEY_CURRENT_USER¥Software¥Sybase¥ODBC]
"LogConfigFile"="c:¥¥temp¥¥odbclog.properties"
```

ロギングを無効にするには、*LogConfigFile* の値を削除するか、名前を変更します。

注意 *HKEY_CURRENT_USER* に指定されている値は、*HKEY_LOCAL_MACHINE* に設定されている値を上書きします。

ログ設定ファイル

設定ファイルは ODBC ログ・ファイルのフォーマットと場所を制御します。この例では、太字の行がログ・ファイルの保存先を指定します。

```
log4cplus.rootLogger=OFF, NULL

log4cplus.logger.com.sybase.dataaccess.odbc.api=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.parameter=TRACE, ODBCTRACE
```

```
log4cplus.additivity.com.sybase.dataaccess.odbc.api.parameter=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.returncode=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.returncode=false

log4cplus.appender.NULL=log4cplus::NullAppender

log4cplus.appender.ODBCTRACE=log4cplus::FileAppender
log4cplus.appender.ODBCTRACE.File=c:¥temp¥odbc.log
log4cplus.appender.ODBCTRACE.layout=log4cplus::PatternLayout
log4cplus.appender.ODBCTRACE.ImmediateFlush=true
log4cplus.appender.ODBCTRACE.layout.ConversionPattern=%d{%H:%M:%S.%q} %t %p
   %-25.25c{2} %m%n
```

ODBC ドライバ・マネージャのトレースなしの動的ロギング・サポート

Adaptive Server Enterprise ODBC ドライバ 15.7 ESD #4 より、SQL_OPT_TRACE 環境の属性を設定することでアプリケーションの実行時にアプリケーションのロギングを動的に有効化または無効化できます。有効値は、無効化の 0 (デフォルト) と有効化の 1 です。

```
// enable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)1,
  SQLINTEGER);
// disable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)0,
  SQLINTEGER);
```

- 動的ロギングは、グローバルに有効化および無効化できます。また、開始されたタイミングや、SQL_OPT_TRACE の設定に使用する環境ハンドルに含まれているどうかに関係なく、すべての接続に影響します。
- デフォルトで、ログは、現在のディレクトリの *sybodbc.log* ファイルに書き込まれます。別のファイル / パスを設定するには、SQL_OPT_TRACEFILE 環境属性を使用します。
- LOGCONFIGFILE 環境変数またはレジストリ値の設定により、アプリケーションの実行の存続期間にわたってロギングが可能であり、SQL_OPT_TRACE を上書きします。
- ODBC ドライバ・マネージャが使用中の場合、SQL_OPT_TRACE の設定によってドライバ・マネージャのトレースが有効になり、ドライバのトレースには影響しません。

- クライアント・アプリケーションでは、ドライバに直接リンクするときは null ハンドルを使用でき、ドライバ・マネージャを使用するときは割り付け済みハンドルを使用できます。
- log4cplus 設定ファイルは、SQL_OPT_TRACE とともに使用することはできません。

TDS プロトコルの取得

ProtocolCapture 接続文字列は、ODBC アプリケーションとサーバ間を送受信される Tabular Data Stream™ (TDS) パケットをデバッグの目的で取得するために使用します。このプロパティは、取得ファイル・プレフィクスを指定することで有効になります。

ProtocolCapture の設定はすぐに反映されるので、接続の確立中に交換された TDS パケットは指定されたファイル・プレフィクスを使用して生成された一意のファイル名に書き込まれます。TDS パケットは、接続している期間中、ファイルに書き込まれます。TDS 取得ファイルを解釈するには、Ribo および他のプロトコル変換ツールを使用できます。

たとえば、*tds_capture* を TDS 取得ファイル・プレフィクスとして指定するには、次のように入力します。

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;ProtocolCapture=tds_capture;
```

最初の接続は *tds_capture0.tds* を生成し、2 番目の接続は *tds_capture1.tds* を生成します (以下同様)。

注意　ファイルに保存された取得済み TDS プロトコル・データは機密のユーザ認証情報を含んでおり、さらに企業や顧客の機密データを含んでいる場合があります。この機密データの無許可または不慮の開示を防止するために、取得済みデータを含むファイルは、ファイルのパーミッションや暗号化によって適切に保護する必要があります。

TDS プロトコルの取得の動的な制御

Adaptive Server Enterprise ODBC ドライバの SQL_ATTR_TDS_CAPTURE 接続属性を使用すれば、TDS プロトコルの取得の一時停止 (SQL_CAPTURE_PAUSE) および再開 (SQL_CAPTURE_RESUME) が可能です。

```
// pause protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_PAUSE, SQLINTEGER);

// resume protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_RESUME, SQLINTEGER);
```

デフォルトで、ProtocolCapture 接続プロパティが接続に設定されている場合、TDS プロトコル取得は接続の間中動作します。SQL_ATTR_TDS_CAPTURE を使用すると (ProtocolCapture 接続プロパティを設定)、目的のプログラム実行セグメントに対する TDS プロトコル取得を選択的に一時停止および再開することができます。

SQL_ATTR_TDS_CAPTURE は、接続ハンドルの割り付け後に設定することができます。接続が確立される前に TDS プロトコル取得を一時停止 / 再開すること、または、TDS プロトコル取得を使用していない接続に対して TDS プロトコル取得を一時停止 / 再開することは、エラーではありません。取得ストリームの整合性を確保するために、TDS プロトコル取得の一時停止または再開がドライバで遅延することがあります。これにより、Ribo および他のプロトコル・トランスレータ・ユーティリティによる正確な取得消費のためのフル PDU パケットの書き込みが可能になります。

接続のすべての TDS パケットの取得を必要とするアプリケーションには、SQL_ATTR_TDS_CAPTURE を設定しないでください。

バインド・パラメータ配列を使用しない ODBC データのバッチ処理

同じ SQL 文が異なるパラメータ値に対して実行される場合、クライアント・アプリケーションは通常パラメータ配列をバインドし、SQLExecute、SQLExecuteDirect、SQLBulkOperations を使用してパラメータの各セットを実行します。配列を SQL パラメータにバインドする際は、配列のメモリが割り付けられ、データがすべて配列にコピーされてから、SQL 文が実行されます。これにより、大量のトランザクションを処理する場合にメモリとリソースの使用効率が低下することがあります。この動作は、Adaptive Server ODBC ドライバのバージョン 15.7 よりも以前のバージョンで発生します。

Adaptive Server ODBC ドライバ 15.7 以降ではクライアント・アプリケーションは SQLExecute を使用して、パラメータを配列としてバインドせずに、パラメータを Adaptive Server にバッチで送信します。SQLExecute は、最後のパラメータのバッチが送信および処理されるまで、SQL_BATCH_EXECUTING を返します。最後のパラメータのバッチが処理されると、実行ステータスが返されます。

SQLRowCount の呼び出しは、最後の SQLExecute 文が完了した後でのみ有効です。

データ・バッチの管理

Sybase 固有の接続属性である SQL_ATTR_BATCH_PARAMS は、Adaptive Server に送信されるパラメータのバッチを管理するために使用します。SQL_ATTR_BATCH_PARAMS は SQLSetConnectAttr を使用して設定します。

値：

- SQL_BATCH_ENABLED － パラメータをバッチ処理するように Adaptive Server ODBC ドライバに伝えます。この状態では、現在処理中の文、つまり SQL_ATTR_BATCH_PARAMS を SQL_BATCH_ENABLED に設定した後に SQLExecute によって実行される最初の文以外の文が接続で実行されると、エラーが送信されます。
- SQL_BATCH_LAST_DATA － 次のパラメータのバッチが最後のバッチであり、パラメータにデータが含まれていることを指定します。

- SQL_BATCH_LAST_NO_DATA － 次のパラメータのバッチが最後のバッチであり、これらのパラメータを無視するように指定します。
- SQL_BATCH_CANCEL － バッチをキャンセルし、トランザクションをロールバックするように Adaptive Server ODBC ドライバに伝えます。

  ロールバックできるのは、コミットされていないトランザクションのみです。
- SQL_BATCH_DISABLED － (デフォルト値) Adaptive Server ODBC ドライバは、最後のパラメータのバッチを処理した後に、この状態に戻ります。SQL_ATTR_BATCH_PARAMS をこの値に手動で設定することはできません。

例

例 1 パラメータ配列をバインドせずにパラメータのバッチをサーバに送信します。

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to start
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
   (SQLPOINTER)SQL_BATCH_ENABLED, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Bind the parameters.This can be done once for the entire batch
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
   SQL_C_LONG, SQL_INTEGER, l1, 0, &c1, l1, &l1);
sr = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_LONGVARCHAR, l2, 0, buffer, l2, &l2);
}

// Run a batch of 10 for (int i = 0; i < 10; i++)
{
   c1 = i;
   memset(buffer, 'a'+i, l2);
   sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
   printError(sr, SQL_HANDLE_STMT, stmt);
}
```

例 2 バッチを終了して閉じます。

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to end
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
  (SQLPOINTER)SQL_BATCH_LAST_NO_DATA, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Call SQLExecDirect one more time to close the batch
// - Due to SQL_BATCH_LAST_NO_DATA, this will not
// process the parameters
sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
```

注意すべき点

- パラメータのバッチ化は、HomogeneousBatch 接続パラメータが 0 以外の値に設定された場合にのみ有効になります。HomogeneousBatch が 2 に設定され、EnableBulkLoad が 0 以外に設定された場合、単純な挿入文が ASE バルク挿入プロトコルを使用して実行されます。非挿入文を実行する場合や、複雑な挿入文を実行する場合には、ASE バッチ・プロトコルを使用します。
- この機能は、結果セットを返さないか、出力パラメータを持たない文およびストアド・プロシージャのみをサポートしています。
- 非同期モードはサポートされていません。バッチ・モード中に、現在バッチ処理している文以外の文をアプリケーションが同じ接続で実行することはできません。
- SQL_DATA_AT_EXEC はサポートされていません。LOB パラメータは通常のパラメータとしてバインドします。
- パラメータ配列をバインドせずにデータをバッチ処理する場合に、SQL_ATTR_PARAM_STATUS_PTR が設定されているとき、Adaptive Server ODBC ドライバでは SQL_ATTR_PARAMSET_SIZE ではなく、SQLSetStmtAttr の StringLength パラメータから複数の配列要素を取得します。

ODBC データのバッチ処理用バルク挿入のサポート

15.7 ESD #4 より、ODBC データのバッチ処理では、パラメータ配列のバインド機能なしで、バルク挿入プロトコルを使用したバッチの挿入がサポートされます。EnableBulkLoad 接続プロパティを目的のバルク・レベル (1、2、または 3) に設定し、HomogeneousBatch 接続プロパティを 2 に設定します。

たとえば、`;enablebulkload=3;homogeneousbatch=2` を接続文字列に付加すると、バッチ処理で実行される単純な挿入文が高速ログによるバルク挿入文に変換されます。

あるいは、SQL_ATTR_HOMOGENEOUS_BATCH 接続属性と SQL_ATTR_ENABLE_BULK_LOAD 接続属性を使用し、次のように接続プロパティをプログラム的に設定すると、同じ結果が得られます。

```
sr = SQLSetConnectAttr(hdbc,
SQL_ATTR_HOMOGENEOUS_BATCH, (SQLPOINTER)2,
SQL_IS_INTEGER);
sr = SQLSetConnectAttr(hdbc,
SQL_ATTR_ENABLE_BULK_LOAD, (SQLPOINTER)3,
SQL_IS_INTEGER);
```

ODBC 遅延配列バインド

Adaptive Server Enterprise ODBC ドライバで、拡張された SQLBindColumnDA() と SQLBindParameterDA() API が提供されるようになりました。これにより、一度の API 呼び出しで、すべてのカラムまたはパラメータをバインドすることができます。これらの API を使用するときに、カラム・バッファまたはパラメータ・バッファを指すポインタがそれぞれの SQLExecute() 呼び出しまたは SQLExecDirect() 呼び出しごとに再評価されます。したがって、アプリケーションでは別の SQLBindCol() 呼び出しまたは SQLBindParameter() 呼び出しなしでバッファの変更が可能になります。新しいポインタをバインドする呼び出しはリソースを多く消費する場合があり、同じ文を何度も実行する必要がある場合に、新しい拡張 API を使用すればアプリケーションのパフォーマンスを改善できます。また、アプリケーションでは特定のメモリ・コピーの処理を省略できます。これは、データが存在する場所から読み取って要求されている場所にコピーするクエリを実行する前に、バッファ・ポインタを変更することで可能です。

SQLBindColumnDA()

説明

バッファを一連のカラム・マーカにバインドします。

構文

```
SQLRETURN SQLBindColumnDA(
  SQLHSTMT StatementHandle,
  SQLSMALLINT* TargetTypes,
  SQLSMALLINT* Precisions,
  SQLSMALLINT* Scales,
  SQLPOINTER* TargetValuePtrs,
  SQLLEN* BufferLengths,
  SQLLEN** StrLens_or_Inds,
  SQLUSMALLINT Columns)
```

パラメータ

StatementHandle

[入力] 文ハンドル。

TargetTypes

[入力] *TargetValuePtrs* の C 言語型。配列のコピーが作成されます。カラムの C 言語型を更新する唯一の方法は、この関数を再度呼び出すことです。

Precisions

[遅延入力] このカラム・バッファに使用する精度。

Scales

[遅延入力] このカラム・バッファに使用する位取り。

TargetValuePtrs

[遅延入力 / 出力] カラムにバインドするデータ・バッファへのポインタ。配列の要素を NULL にすることはできません。

BufferLengths

[遅延入力] バイト単位の *TargetValuePtrs* バッファの長さ。

StrLens_or_Inds

[遅延入力 / 出力] カラムにバインドする長さ / インジケータ・バッファへのポインタ。

Columns

[入力] バインドされたカラムの数。

SQLBindParameterDA()

説明　バッファを一連のパラメータ・マーカにバインドします。

構文

```
SQLRETURN SQLBindParameterDA(
  SQLHSTMT StatementHandle,
  SQLSMALLINT* InputOutputTypes,
  SQLSMALLINT* ValueTypes,
  SQLSMALLINT* ParameterTypes,
  SQLULEN* ColumnSizes,
  SQLSMALLINT* DecimalDigits,
  SQLPOINTER* ParameterValuePtrs,
  SQLLEN* BufferLength,
  SQLLEN** StrLens_or_IndPtrs,
  SQLUSMALLINT Parameters)
```

パラメータ

StatementHandle

[入力] 文ハンドル。

InputOutputTypes

[入力] パラメータの型。配列のコピーが作成されます。パラメータの *InputOutputType* を更新する唯一の方法は、この関数を再度呼び出すことです。

ValueTypes

[入力] パラメータの C データ型。配列のコピーが作成されます。パラメータの *ValueType* を更新する唯一の方法は、この関数を再度呼び出すことです。

ParameterTypes

[入力] パラメータの SQL データ型。配列のコピーが作成されます。パラメータの *ParameterType* を更新する唯一の方法は、この関数を再度呼び出すことです。

ColumnSizes

[入力] 対応するパラメータ・マーカーのカラムまたは式のサイズ。配列のコピーが作成されます。パラメータの *ColumnSize* を更新する唯一の方法は、この関数を再度呼び出すことです。

DecimalDigits

[入力] 対応するパラメータ・マーカーのカラムまたは式の小数行数。配列のコピーが作成されます。パラメータの *DecimalDigits* を更新する唯一の方法は、この関数を再度呼び出すことです。

ParameterValuePtrs

[遅延入力 / 出力] パラメータ・データ用バッファへのポインタの配列。配列の要素を NULL にすることはできません。

BufferLength

[遅延入力] バッファの長さの配列。

StrLens_or_IndPtrs

[遅延入力] パラメータの長さ用バッファへのポインタの配列。

Parameters

[入力] バインドされたパラメータの数。

使用法

- 標準のバインドと遅延配列バインドを混在して使用することはできません。
- 標準バインドと遅延配列バインド間で切り替える場合は、すべてのバインドを解除する必要があります。
- 遅延配列バインドを使用するときは、単一の API 呼び出しですべてのカラムまたはパラメータを省略することなく適切にバインドする必要があります。これは以降の SQLBindColumnDA() または SQLBindParameterDA() への呼び出しで、前の呼び出しの値が置き換えられてしまうためです。
- SQLExecute() および SQLExecDirect() では、SQLBindParameterDA() がパラメータのバインドに使用されている場合に、*BufferLength* に関連する SQLBindParameter() のエラーを返せるようになりました。
- SQLFetch() では、SQLBindColumnDA() がパラメータのバインドに使用されている場合に、*BufferLength* に関連する SQLBindCol() のエラーを返せるようになりました。
- SQLBindColumnDA() および SQLBindParameterDA() は、ODBC ドライバ・マネージャとともに使用することができません。これは、この機能で非標準 API 呼び出しを使用しているためです。
- SQLSetDescField() の使用方法の制限として、遅延配列バインドの使用時 *FieldIdentifier* の一部の値を使用できないことが挙げられます。たとえば、アプリケーションは使用しているバッファを変更するために *ValuePtr* を変更する必要があるため、SQL_DESC_DATA_PTR がエラーを返します。SQLBindCol() または SQLBindParameter() フィールドを更新する *FieldIdentifier* は、遅延配列バインドが使用されたときエラーを返します。

サンプル・プログラム

dabinding サンプル・プログラムはこの機能の実例を示しています。

追加のロー・フォーマットの抑制に関する情報

SuppressRowFormat2 接続文字列プロパティは、Adaptive Server が TDS_ROWFMT2 バイト・シーケンスではなく、可能な場合に TDS_ROWFMT バイト・シーケンスを使用してデータを送信するように強制するために使用します。TDS_ROWFMT は、カタログ、スキーマ、テーブル、カラム情報を含む TDS_ROWFMT2 よりも少ないデータを含んでおり、多くの小さな select オペレーションでより優れたパフォーマンスを実現します。SuppressRowFormat2 が 1 に設定されている場合、サーバは縮小された結果セット・メタデータを送信するので、一部の情報がクライアント・プログラムで使用できなくなります。欠如しているメタデータにアプリケーションが依存している場合、このプロパティは有効にしないでください。

値：

- 0 ― デフォルト値。TDS_ROWFMT2 は抑制されません。
- 1 ― サーバがデータをできる限り TDS_ROWFMT で送信するように強制します。

注意　SuppressRowFormat2 接続文字列プロパティは、SQLBulkOperations API を使用する ODBC プログラムに使用しないでください。SuppressRowFormat2 を有効にすると、SQL バルク処理に必要な情報が抑制され、エラーの原因になります。

注意　Adaptive Server 15.7 ESD #1 以降に接続する場合、SuppressRowFormat2 プロパティは古いものと考えてください。パフォーマンスが高く、制限が少ない SuppressRowFormat 接続プロパティを使用してください。

ロー・フォーマット・メタデータの抑制

繰り返し実行されるクエリのパフォーマンスは、Adaptive Server ODBC ドライバによって向上できます。これには、セッションで繰り返し実行されるクエリのロー・フォーマット・メタデータ (TDS_ROWFMT または TDS_ROWFMT2) を抑制するように Adaptive Server で指定します。Adaptive Server 15.7 ESD#1 以降では、ロー・フォーマット・メタデータの抑制がサポートされています。

ロー・フォーマット・メタデータを抑制するには、SuppressRowFormat 接続文字列プロパティを使用します。

有効な SuppressRowFormat 接続文字列プロパティの値は次のとおりです。

- 0 ーロー・フォーマット・メタデータは抑制されません。
- 1 ー デフォルト値。Adaptive Server は可能な場合にはロー・フォーマット・メタデータを送信しません。

注意 ロー・フォーマット・メタデータの抑制は、接続している Adaptive Server でこの機能をサポートしている場合にのみ実行できます。SuppressRowFormat パラメータが 1 に設定され、接続している Adaptive Server でロー・フォーマット・メタデータの抑制がサポートされていない場合、Adaptive Server はこのパラメータを無視します。

例

この ODBC 接続文字列によって、ロー・フォーマット・メタデータが抑制されます。

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressRowFormat=1;
```

パラメータ・フォーマット・メタデータの抑制

準備文のパフォーマンスは ODBC ドライバによって向上できます。これには、準備文の再実行時にパラメータ・フォーマット・メタデータを抑制します。Adaptive Server 15.7 ESD#1 以降では、パラメータ・フォーマット・メタデータの抑制がサポートされています。

パラメータ・フォーマット・メタデータを抑制するには、DynamicPrepare 接続プロパティを 1 に設定し、次に SuppressParamFormat 接続文字列プロパティを使用します。

SuppressParamFormat の有効な接続文字列プロパティの値は次のとおりです。

- 0 ー パラメータ・フォーマット・メタデータは、準備文では抑制されません。
- 1 ー デフォルト値。パラメータ・フォーマット・メタデータは、可能な場合抑制されます。

注意　準備文でのパラメータ・フォーマット・メタデータの抑制は、接続している Adaptive Server でこの機能がサポートされている場合にのみ実行できます。DynamicPrepare および SuppressParamFormat パラメータの両方が 1 に設定され、接続している Adaptive Server でパラメータ・フォーマット・メタデータの抑制がサポートされていない場合、Adaptive Server はパラメータ設定を無視します。

例

次の ODBC 接続文字列によって、パラメータ・フォーマット・メタデータは、準備文で抑制されます。

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressParamFormat=1;
```

カーソル・クローズ時のロックの解放

Adaptive Server では、release_locks_on_close オプションを含めるように declare cursor 構文が拡張されています。このオプションは、カーソルのクローズ時に独立性レベル 2 および 3 でカーソルの共有ロックを解放します。Adaptive Server ODBC ドライバは、release-lock-on-close セマンティックをサポートしています。

この機能を Adaptive Server ODBC Driver 接続で作成されたすべての読み取り専用カーソルに適用するには、ReleaseLocksOnCursorClose 接続プロパティを 1 に設定します。デフォルトの ReleaseLocksOnCursorClose 値は 0 です。

ReleaseLocksOnCursorClose 接続プロパティによって適用された設定は静的で、接続の確立後に変更することはできません。この設定は、release_locks_on_close をサポートしているサーバに接続されている場合にのみ有効です。

release_locks_on_close の詳細については、『ASE リファレンス・マニュアル：コマンド』を参照してください。

select for update のサポート

Adaptive Server は、同じトランザクション内の後続の更新用にローをロックできる select for update と、更新可能なカーソル用の排他ロックをサポートしています。『ASE Transact-SQL ユーザーズ・ガイド』の「第 2 章 クエリ：テーブルからのデータの選択」を参照してください。

この機能は、for update 句が select 文に追加されたときと、クライアント内で開いている更新可能なカーソルに追加されたときにクライアントで自動的に使用可能になります。

データオンリーロック・テーブルの可変長ロー

16K 論理ページ・サイズを使用するように設定されている Adaptive Server 15.7 より前のバージョンでは、可変長カラムが行の先頭から 8191 バイトを超えた位置から始まっている場合、可変長ローを含むデータオンリー・ロック (DOL) テーブルは作成できませんでした。この制限は、Adaptive Server 15.7 以降では取り除かれています。『ASE パフォーマンス&チューニング・シリーズ：物理データベースのチューニング』の「第 2 章 データの格納」を参照してください。

ODBC クライアントがこの機能を使用するにあたって特別な設定を行う必要はありません。長い DOL ローを受信するように設定されている Adaptive Server バージョン 15.7 に接続すると、これらのクライアントは長いオフセットを使用して自動的にレコードを挿入します。クライアントが長い DOL ローを以前のバージョンの Adaptive Server に接続しようとするか、長い DOL ロー・オプションが無効になっている 15.7 Adaptive Server に接続しようとすると、エラー・メッセージが送信されます。

非実体化カラム

Adaptive Server ODBC ドライバ・バージョン 15.7 以降のバルク挿入ルーチンは、Adaptive Server 15.7 の非実体化カラムを処理できます。以前のバージョンの Adaptive Server ODBC ドライバでは、テーブル定義に非実体化カラムが含まれている場合、Adaptive Server へのデータのバルク挿入は実行できません。以前のバージョンの Adaptive Server ODBC ドライバで非実体化カラムにバルク挿入を実行しようとすると、エラーが発生します。

ラージ・オブジェクト (LOB) サポート

Adaptive Server ODBC ドライバは、ラージ・オブジェクト (LOB) データ型の text、unitext、および image の使用を次のようにサポートしています。

- ロー内の記憶領域を使用する LOB カラム

  Adaptive Server では、ロー内に格納するようにマークされている LOB カラムは、ロー全体を格納するのに十分なメモリがある場合、ロー内に格納されます。ロー内のカラムが更新されたために、ローのサイズが定義されている制限を超えた場合、ロー内に格納されている LOB カラムは制限を満たすためにロー外に移動されます。『ASE Transact-SQL ユーザーズ・ガイド』の「第 21 章 ロー内／ロー外の LOB」を参照してください。

  Adaptive Server ODBC ドライバのバルク挿入ルーチンは、Adaptive Server の text、image、unitext の LOB カラムのロー内およびロー外の記憶領域をサポートしています。以前のクライアント・バージョンからのバルク挿入ルーチンでは、LOB カラムは常にロー外に格納されます。

- ストアド・プロシージャのパラメータとしての LOB オブジェクト

  Adaptive Server ODBC ドライバは、ストアド・プロシージャでの入力パラメータおよびパラメータ・マーカーのデータ型として text、unitext、および image の使用をサポートしています。

ラージ・オブジェクト (LOB) のロケータのサポート

Adaptive Server ODBC ドライバは、ラージ・オブジェクト (LOB) ロケータをサポートしています。LOB ロケータには、データ自体ではなく、LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。

Adaptive Server ODBC ドライバ・クライアントは、LOB ロケータをサポートしている Adaptive Server に接続していない限り、LOB ロケータを使用できません。Adaptive Server には、LOB ロケータに対するサーバ・サポートが導入されています。

注意 LOB ロケータを使用している場合、各ローに LOB データを含む大きな結果セットを取得すると、アプリケーションのパフォーマンスに影響が及ぶ場合があります。Adaptive Server では LOB ロケータを結果セットの一部として返します。LOB データを取得するには、Adaptive Server ODBC ドライバが残りの結果セットをキャッシュに格納する必要があります。結果セットは小さいサイズを保持するか、カーソルのサポートを有効にしてキャッシュに格納するデータのサイズを制限することをおすすめします。

LOB ロケータのサポート

LOB ロケータのサポートを Adaptive Server ODBC ドライバで有効にするには、EnableLOBLocator 接続プロパティを 1 に設定して Adaptive Server への接続を確立します。When EnableLOBLocator がデフォルト値である 0 に設定されている場合、Adaptive Server ODBC ドライバは LOB カラムのロケータを取得できません。LOB ロケータを有効にする場合は、接続を autocommit off に設定してください。

sybasesqltypes.h ファイルをプログラムに含める必要もあります。*sybasesqltypes.h* ファイルは ODBC インストール・ディレクトリの下の *include* ディレクトリに含まれています。

注意 アプリケーションがドライバに直接リンクされているときには、LOB ロケータを確実に使用できます。ドライバ・マネージャと LOB ロケータが使用されいているとき、一部のドライバ・マネージャでベンダ定義の C および SQL 型が制限され、アプリケーションで " サポートされていない型 " のエラーが発生する可能性があります。

ロケータをサポートするための ODBC データ型のマッピング

Adaptive Server ロケータのデータ型に対する ODBC データ型のマッピングは次のとおりです。

ASE データ型	ODBC SQL 型	ODBC C 型
text_locator	SQL_TEXT_LOCATOR	SQL_C_TEXT_LOCATOR
image_locator	SQL_IMAGE_LOCATOR	SQL_C_ IMAGE_LOCATOR
unitext_locator	SQL_UNITEXT_LOCATOR	SQL_C_ UNITEXT_LOCATOR

サポートされている変換

Adaptive Server ロケータのデータ型に対してサポートされている変換は次のとおりです。

	SQL_C_TEXT_ LOCATOR	SQL_C_IMAGE_ LOCATOR	SQL_C_UNITEXT_ LOCATOR
SQL_TEXT_LOCATOR	X		
SQL_IMAGE_LOCATOR		X	
SQL_UNITEXT_LOCATOR			X
SQL_LONGVARCHAR			
SQL_WLONGVARCHAR			
SQL_LONGVARBINARY			

凡例：X = サポートされている変換

LOB ロケータをサポートしている ODBC API メソッド

- SQLBindCol － *TargetType* には ODBC C ロケータの任意のデータ型を指定できます。
- SQLBindParameter － *ValueType* には ODBC C ロケータの任意のデータ型を指定できます。*ParameterType* には ODBC SQL ロケータの任意のデータ型を指定できます。
- SQLGetData － *TargetType* には ODBC C ロケータの任意のデータ型を指定できます。
- SQLColAttribute － SQL_DESC_TYPE および SQL_DESC_CONCISE_TYPE 記述子は、ODBC SQL ロケータの任意のデータ型を返すことができます。
- SQLDescribeCol － データ型のポインタには、ODBC SQL ロケータの任意のデータ型を指定できます。

『Microsoft ODBC API Reference』を参照してください。

プリフェッチされた LOB データの暗黙的変換

Adaptive Server が LOB ロケータを返した場合、SQLGetData および SQLBindCol を使用して、text ロケータ用の SQL_C_CHAR または SQL_C_WCHAR、あるいは image ロケータ用の SQL_C_BINARY にカラムをバインドすることで、基本となるプリフェッチされた LOB データを取得します。

接続内のロケータを有効または無効にするには、SQL_ATTR_LOBLOCATOR 属性を設定します。EnableLOBLocator が接続文字列内で指定されている場合、SQL_ATTR_LOBLOCATOR は EnableLOBLocator の値で初期化されます。それ以外の場合はデフォルト値である SQL_LOBLOCATOR_OFF に設定されます。ロケータを有効にするには、属性を SQL_LOBLOCATOR_ON に設定します。属性の値を設定するには SQLSetConnectAttr を使用し、属性の値を取得するには SQLGetConnectAttr を使用します。

SQLSetStatementAttr を使用して SQL_ATTR_LOBLOCATOR_FETCHSIZE を設定し、取得する LOB データのサイズを指定します。バイナリ・データのサイズはバイト数で指定し、文字データのサイズは文字数で指定します。デフォルト値の 0 は、プリフェッチされたデータが要求されないことを示し、-1 の値は LOB データ全体が取得されることを示します。

注意　取得するカラムの基本となる LOB データのサイズが、設定されているプリフェッチされたデータのサイズを超えた場合は、ODBC クライアントがデータを直接取得しようとしたときにネイティブ・エラー 3202 が発生します。これが発生した場合、クライアントは SQLGetData を呼び出すことで完全なデータを取得し、基本となるロケータを取得して、ロケータで利用できるオペレーションをすべて実行できます。

例 1 プリフェッチされたデータが完全な LOB 値を表しているときに、SQLGetData を使用して image ロケータを取得します。

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR, (SQLPOINTER)SQL_LOCATOR_ON,
   0);

// Set size of prefetched LOB data
```

```
sr = SQLSetStatementAttr(stmt, SQL_ATTR_LOBLOCATOR_FETCHSIZE,
(SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);

printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);

sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

//Retrieve the binary data  (Complete Data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);
printError(sr, SQL_HANDLE_STMT, stmt);


//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR, (SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

例 2 プリフェッチされたデータがトランケートされた LOB 値を表しているときに、SQLGetData を使用して image ロケータを取得します。

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
```

```
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
   (SQLPOINTER)SQL_LOCATOR_ON, 0);

// Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt,
   SQL_ATTR_LOBLOCATOR_FETCHSIZE, (SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);
printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

// Retrieve the binary data(Truncated data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);

if(sr == SQL_SUCCESS_WITH_INFO)
{
   SQLTCHAR errormsg[ERR_MSG_LEN];
   SQLTCHAR sqlstate[SQL_SQLSTATE_SIZE+1];
   SQLINTEGER nativeerror = 0;
   SQLSMALLINT errormsglen = 0;

   retcode = SQLGetDiagRec(handleType, handle, 1, sqlstate, &nativeerror,
      errormsg, ERR_MSG_LEN, &errormsglen);

   printf("SqlState:%s Error Message:%s¥n", sqlstate, errormsg);

   //Handle truncation of LOB data; if data was truncated call SQLGetData to
```

```
   // retrieve the locator.

   /* Warning returns truncated LOB data */
   if (NativeError == 32028) //Error code may change
   {
      BYTE ImageLocator[SQL_LOCATOR_SIZE];
      sr = SQLGetData(stmt, 1, SQL_C_IMAGE_LOCATOR, &ImageLocator,
         sizeof(ImageLocator), &Len);
      printError(sr, SQL_HANDLE_STMT, stmt);

      /*
         Perform locator specific calls using image Locator on a separate


         statement handle if needed
      */
   }
}

//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR, (SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

ロケータを使用した LOB のアクセスと操作

ODBC API は LOB ロケータを直接サポートしていません。ODBC クライアント・アプリケーションでは Transact-SQLR 関数を使用して、ロケータに対するオペレーションを行い、LOB 値を操作する必要があります。Adaptive Server ODBC ドライバには、必要な Transact-SQL 関数の使用を助長するためのストアド・プロシージャがいくつか導入されています。

この項では、LOB ロケータに対してさまざまなオペレーションを実行する方法について説明します。パラメータの入出力値には、Adaptive Server がストアド・プロシージャ定義に暗黙的に変換できる任意の型を指定できます。

ここに示す Transact-SQL コマンドおよび関数の詳細については、『ASE リファレンス・マニュアル：ビルディング・ブロック』の「Transact-SQL 関数」を参照してください。

text ロケータの初期化

sp_drv_create_text_locator を使用して text_locator を作成し、オプションでその値を初期化します。sp_drv_create_text_locator は、Transact-SQL 関数 create_locator にアクセスします。

構文

sp_drv_create_text_locator [*init_value*]

入力パラメータ

init_value ― 新しいロケータの初期化に使用される varchar または text の値。

出力パラメータ

なし

結果セット

text_locator 型のカラム。ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

unitext ロケータの初期化

sp_drv_create_unitext_locator を使用して unitext_locator を作成し、オプションでその値を初期化します。sp_drv_create_unitext_locator は、Transact-SQL 関数 create_locator にアクセスします。

構文

sp_drv_create_unitext_locator [*init_value*]

入力パラメータ

init_value ― 新しいロケータの初期化に使用される univarchar または unitext。

出力パラメータ

なし

結果セット

unitext_locator 型のカラム。ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

image ロケータの初期化

sp_drv_create_image_locator を使用して image_locator を作成し、オプションでその値を初期化します。sp_drv_create_image_locator は、Transact-SQL 関数 create_locator にアクセスします。

構文

sp_drv_create_image_locator [*init_value*]

入力パラメータ

init_value ― 新しいロケータの初期化に使用される varbinary または image。

出力パラメータ

なし

結果セット

image_locator 型のカラム。ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

text ロケータからの完全な *text* 値の取得

Transact-SQL 関数 return_lob にアクセスする sp_drv_locator_to_text を使用します。

構文　sp_drv_locator_to_text *locator*

入力パラメータ　*locator* ― 値の取得対象となる text_locator。

出力パラメータ　なし

結果セット　*locator* によって参照される text 値を含むカラム。

unitext ロケータからの完全な *unitext* 値の取得

Transact-SQL 関数 return_lob にアクセスする sp_drv_locator_to_unitext を使用します。

構文　sp_drv_locator_to_unitext *locator*

入力パラメータ　*locator* ― 値の取得対象となる unitext_locator。

出力パラメータ　なし

結果セット　*locator* によって参照される unitext 値を含むカラム。

image ロケータからの完全な *image* 値の取得

Transact-SQL 関数 return_lob にアクセスする sp_drv_locator_to_image を使用します。

構文　sp_drv_locator_to_image *locator*

入力パラメータ　*locator* ― 値の取得対象となる image_locator。

出力パラメータ　なし

結果セット　*locator* によって参照される image 値を含むカラム。

text ロケータからの部分文字列の取得

Transact-SQL 関数 substring にアクセスする sp_drv_text_substring を使用します。

構文　sp_drv_text_substring *locator*, *start_position*, *length*

入力パラメータ

- *locator* ― 操作するデータを参照する text_locator。
- *start_position* ― 読み込んで取得する最初の文字の位置を指定する integer。
- *length* ― 読み込む文字数を指定する integer。

出力パラメータ　なし

結果セット　取得された部分文字列を含む text 型のカラム。

unitext ロケータからの部分文字列の取得

Transact-SQL 関数 substring にアクセスする sp_drv_unitext_substring を使用します。

構文　sp_drv_unitext_substring *locator*, *start_position*, *length*

入力パラメータ
- *locator* ― 操作するデータを参照する unitext_locator。
- *start_position* ― 読み込んで取得する最初の文字の位置を指定する integer。
- *length* ― 読み込む文字数を指定する integer。

出力パラメータ　なし

結果セット　取得された部分文字列を含む unitext 型のカラム。

image ロケータからの部分文字列の取得

Transact-SQL 関数 substring にアクセスする sp_drv_image_substring を使用します。

構文　sp_drv_image_substring *locator*, *start_position*, *length*

入力パラメータ
- *locator* ― 操作するデータを参照する image_locator。
- *start_position* ― 読み込んで取得する最初のバイトの位置を指定する integer。
- *length* ― 読み込むバイト数を指定する integer。

出力パラメータ　なし

結果セット　取得された部分文字列を含む image 型のカラム。

指定した位置への *text* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_text_setdata を使用します。

構文　sp_drv_text_setdata *locator*, *offset*, *new_data*, *data_length*

入力パラメータ
- *locator* ― 挿入先の text カラムを参照する text_locator。
- *offset* ― 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data* ― 挿入する varchar または text データ。

出力パラメータ　*data_length* ― 書き込まれた文字数を含む integer。

結果セット　なし

指定した位置への *unitext* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_unitext_setdata を使用します。

構文

```
sp_drv_unitext_setdata locator, offset, new_data, data_length
```

入力パラメータ

- *locator* ― 挿入先の unitext カラムを参照する unitext_locator。
- *offset* ― 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data* ― 挿入する univarchar または unitext データ。

出力パラメータ　*data_length* ― 書き込まれた文字数を含む integer。

結果セット　なし

指定した位置への *image* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_image_setdata を使用します。

構文

```
sp_drv_image_setdata locator, offset, new_data, datalength
```

入力パラメータ

- *locator* ― 挿入先の image カラムを参照する image_locator。
- *offset* ― 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data* ― 挿入する varbinary または image データ。

出力パラメータ　*data_length* ― 書き込まれたバイト数を含む integer。

結果セット　なし

ロケータが参照する *text* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_text_locator_setdata を使用します。

構文

```
sp_drv_text_locator_setdata locator, offset, new_data_locator,
data_length
```

入力パラメータ

- *locator* ― 挿入先の text カラムを参照する text_locator。
- *offset* ― 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data_locator* ― 挿入先の text データを参照する text_locator。

出力パラメータ　*data_length* ― 書き込まれた文字数を含む integer。

結果セット　なし

ロケータが参照する *unitext* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_unitext_locator_setdata を使用します。

構文　sp_drv_unitext_locator_setdata *locator*, *offset*, *new_data_locator*, *data_length*

入力パラメータ
- *locator* ― 挿入先の unitext カラムを参照する unitext_locator。
- *offset* ― 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data_locator* ― 挿入先の unitext データを参照する unitext_locator。

出力パラメータ　*data_length* ― 書き込まれた文字数を含む integer。

結果セット　なし

ロケータが参照する *image* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_image_locator_setdata を使用します。

構文　sp_drv_image_locator_setdata *locator*, *offset*, *new_data_locator*, *datalength*

入力パラメータ
- *locator* ― 挿入先の image カラムを参照する image_locator。
- *offset* ― 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data_locator* ― 挿入先の image データを参照する image_locator。

出力パラメータ　*data_length* ― 書き込まれたバイト数を含む integer。

結果セット　なし

基本となる LOB データのトランケート

truncate lob を使用して、LOB ロケータが参照している LOB データをトランケートします。Adaptive Server Enterprise の『クラスタ・ユーザ・ガイド：コマンド』を参照してください。

基本となる *text* データの文字長の確認

sp_drv_text_locator_charlength を使用して、text ロケータが参照している LOB カラムの文字長を確認します。sp_drv_text_locator_charlength では Transact-SQL 関数 char_length にアクセスします。

構文　sp_drv_text_locator_charlength *locator*, *data_length*

入力パラメータ　*locator* － 操作する text カラムを参照する text_locator。

出力パラメータ　*data_length* － *locator* が参照する text カラムの文字長を指定する integer。

結果セット　なし

基本となる *text* データのバイト長の確認

sp_drv_text_locator_bytelength を使用して、text ロケータが参照している LOB カラムのバイト長を確認します。sp_drv_text_locator_bytelength では Transact-SQL 関数 data_length にアクセスします。

構文　sp_drv_image_locator_bytelength *locator*, *data_length*

入力パラメータ　*locator* － 操作する text カラムを参照する text_locator。

出力パラメータ　*data_length* － *locator* が参照する text カラムのバイト長を指定する integer。

結果セット　なし

基本となる *unitext* データの文字長の確認

sp_drv_unitext_locator_charlength を使用して、unitext ロケータが参照している LOB カラムの文字長を確認します。
sp_drv_unitext_locator_charlength では Transact-SQL 関数 char_length にアクセスします。

構文　sp_drv_unitext_locator_charlength *locator*, *data_length*

入力パラメータ　*locator* － 操作する unitext カラムを参照する unitext_locator。

出力パラメータ　*data_length* － *locator* が参照する unitext カラムの文字長を指定する integer。

結果セット　なし

基本となる *unitext* データのバイト長の確認

sp_drv_unitext_locator_bytelength を使用して、unitext ロケータが参照している LOB カラムのバイト長を確認します。
sp_drv_unitext_locator_bytelength では Transact-SQL 関数 data_length にアクセスします。

構文

sp_drv_image_locator_bytelength *locator*, *data_length*

入力パラメータ

locator − 操作する unitext カラムを参照する unitext_locator。

出力パラメータ

data_length − *locator* が参照する unitext カラムのバイト長を指定する integer。

結果セット

なし

基本となる *image* データのバイト長の確認

sp_drv_image_locator_bytelength を使用して、image ロケータが参照している LOB カラムのバイト長を確認します。
sp_drv_image_locator_bytelength では Transact-SQL 関数 data_length にアクセスします。

構文

sp_drv_image_locator_bytelength *locator*, *data_length*

入力パラメータ

locator − 操作する image カラムを参照する image_locator。

出力パラメータ

data_length − *locator* が参照する image カラムのバイト長を指定する integer。

結果セット

なし

ロケータが参照する *text* カラム内の検索文字列位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_varchar_charindex を使用します。

構文

sp_drv_varchar_charindex *search_string*, *locator*, *start*, *position*

入力パラメータ

- *search_string* − 検索対象となる varchar 型のリテラル。
- *locator* − 検索元の text カラムを参照する text_locator。
- *start* − 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ

position − *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する integer。

結果セット

なし

別のロケータが参照している *text* カラム内の *text* ロケータが参照している文字列の位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_textlocator_charindex を使用します。

構文

sp_drv_textlocator_charindex *search_locator*, *locator*, *start*, *position*

入力パラメータ

- *search_locator* ― 検索するリテラルを指す text_locator。
- *locator* ― 検索元の text カラムを参照する text_locator。
- *start* ― 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ

position ― *locator* が参照する LOB カラム内のリテラルの開始位置を指定する integer。

結果セット

なし

ロケータが参照する *unitext* カラム内の検索文字列位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_univarchar_charindex を使用します。

構文

sp_drv_univarchar_charindex *search_string*, *locator*, *start*, *position*

入力パラメータ

- *search_string* ― 検索対象となる univarchar 型のリテラル。
- *locator* ― 検索元の unitext カラムを参照する unitext_locator。
- *start* ― 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ

position ― *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する integer。

結果セット

なし

別のロケータが参照している *unitext* カラム内の *unitext* ロケータが参照している文字列の位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_unitext_locator_charindex を使用します。

構文

sp_drv_charindex_unitextloc_in_locator *search_locator*, *locator*, *start*, *位置*

入力パラメータ

- *search_locator* ― 検索するリテラルを指す unitext_locator。
- *locator* ― 検索元の text カラムを参照する unitext_locator。
- *start* ― 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* ― *locator* が参照する LOB カラム内のリテラルの開始位置を指定する integer。

結果セット　なし

image ロケータが参照するカラム内のバイト・シーケンス位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_varbinary_charindex を使用します。

構文

```
sp_drv_varbinary_charindex byte_sequence, locator, start, position
```

入力パラメータ

- *byte_sequence* ― 検索対象の varbinary 型のバイト・シーケンス。
- *locator* ― 検索元の image カラムを参照する image_locator。
- *start* ― 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* ― *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する integer。

結果セット　なし

別のロケータが参照している *image* カラム内の *image* ロケータが参照しているバイト・シーケンス位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_image_locator_charindex を使用します。

構文

```
sp_drv_image_locator_charindex sequence_locator, locator, start, start_position
```

入力パラメータ

- *sequence_locator* ― 検索対象のバイト・シーケンスを指す image_locator。
- *locator* ― 検索元の image カラムを参照する image_locator。
- *start* ― 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*start_position* ― *locator* が参照する LOB カラム内のバイト・シーケンスの開始位置を指定する integer。

結果セット　なし

text_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする sp_drv_text_locator_valid を使用します。

構文　　sp_drv_text_locator_valid *locator*

入力パラメータ　　*locator* － 検証する text_locator。

出力パラメータ　　次の値のいずれかを表す bit。

- 0 － false。*locator* は無効です。
- 1 － true。*locator* は有効です。

結果セット　　なし

unitext_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする sp_drv_unitext_locator_valid を使用します。

構文　　sp_drv_unitext_locator_valid *locator*

パラメータ　　*locator* － 検証する unitext_locator。

出力パラメータ　　次の値のいずれかを表す bit。

- 0 － false。*locator* は無効です。
- 1 － true。*locator* は有効です。

結果セット　　なし

image_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする sp_drv_image_locator_valid を使用します。

構文　　sp_drv_image_locator_valid *locator*

パラメータ　　*locator* － 検証する image_locator。

出力パラメータ　　次の値のいずれかを表す bit。

- 0 － false。*locator* は無効です。
- 1 － true。*locator* は有効です。

結果セット　　なし

LOB ロケータの解放または割り付け解除

deallocate locator を使用します。Adaptive Server Enterprise の『クラスタ・ユーザ・ガイド：コマンド』を参照してください。

例

例 1 ハンドルを割り付け、接続を確立します。

```
// Assumes that DSN has been named "sampledsn" and
// UseLobLocator has been set to 1.

SQLHENV environmentHandle = SQL_NULL_HANDLE;
SQLHDBC connectionHande = SQL_NULL_HANDLE;
SQLHSTMT statementHandle = SQL_NULL_HANDLE;
SQLRETURN ret;

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &environmentHandle);
SQLSetEnvAttr(environmentHandle, SQL_ATTR_ODBC_VERSION, SQL_ATTR_OV_ODBC3);
SQLAllocHandle(SQL_HANDLE_DBC, environmentHandle, &connectionHandle);
Ret = SQLConnect(connectionHandle, "sampledsn",
   SQL_NTS, "sa", SQL_NTS, "Sybase",SQL_NTS);
```

例 2 カラムを選択し、ロケータを取得します。

```
// Selects and retrieves a locator for bk_desc, where
// bk_desc is a column of type text defined in a table
// named books. bk_desc contains the text "A book".

SQLPrepare(statementHandle, "SELECT bk_desc FROM books
   WHERE bk_id =1", SQL_NTS);

SQLExecute(statementHandle);
BYTE TextLocator[SQL_LOCATOR_SIZE];
SQLLEN Len = 0;
ret = SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);

If(Len == sizeof(TextLocator))
{
   Cout << Locator was created with expected size <<
   Len;
}
```

例 3 データ長を判別します。

```
SQLLEN LocatorLen = sizeof(TextLocator);
ret = SQLBindParameter(statementHandle, 1,
   SQL_PARAM_INPUT, SQL_C_TEXT_LOCATOR,
   SQL_TEXT_LOCATOR, SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), &LocatorLen);

SQLLEN CharLenSize = 0;
SQLINTEGER CharLen = 0;
ret = SQLBindParameter(statementHandle, 2,
SQL_PARAM_OUTPUT, SQL_C_LONG,SQL_INTEGER,0 , 0,
&CharLen, sizeof(CharLen), &CharLenSize);
SQLExecDirect(statementHandle,
   "{CALL sp_drv_text_locator_charlength( ?,?)}" , SQL_NTS);

cout<< "Character Length of Data " << charLen;
```

例 4 テキストを LOB カラムに追加します。

```
SQLINTEGER retVal = 0;
SQLLEN Col1Len = sizeof(retVal);
SQLCHAR appendText[10]=” abcdefghi on C++” ;

SQLBindParameter(statementHandle, 14,
   SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &retVal, 0, Col1Len);

SQLBindParameter(statementHandle, 21, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, &TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 32, SQL_PARAM_INPUT,
   SQL_C_SLONG, SQL_INTEGER, 0, 0, &charLen, 0, SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 43, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 10, 0, append_text,
   sizeof(append_text), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle,
   "{? = CALL sp_drv_setdata_text (?, ?, ?,?)}" , SQL_NTS);

SQLFreeStmt(statementHandle, SQL_CLOSE);
```

例 5 LOB データを LOB ロケータから取得します。

```
SQLCHAR description[512];
SQLLEN descriptionLength = 512;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle, "{CALL sp_drv_locator_to_text(?)}", SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, 1,SQL_C_CHAR, description,
   descriptionLength, &descriptionLength)

Cout << "LOB data referenced by locator:"<< description
   << endl;

Cout << "Expected LOB data:A book on C++" << endl;
```

例 6 クライアント・アプリケーションのデータを LOB ロケータに転送します。

```
description = "A lot of data that will be used for a lot
   of inserts, updates and deletes"; descriptionLength = SQL_NTS;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 512, 0, description,
   sizeof(description), &descriptionLength);

SQLExecDirect(statementHandle,
   "{CALL sp_drv_create_text_locator(?)}", SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);
```

サーバで指定されたパケット・サイズの使用

クライアントとサーバは、相互の通信に使用するパッケージを格納するメモリを予約するための準備をする必要があります。これらのパッケージは、プロトコル・データ・ユニット (PDU: Protocol Data Unit) と呼ばれます。すべての PDU は、現在の PDU のバイト単位の実際のサイズ (ヘッダ自体を含む) を記述する、2 バイトの符号なし整数を含む 8 バイトのヘッダで開始されます。クライアントとサーバは、相手側から送信できる PDU の最大サイズ、つまり「パケット・サイズ」を把握している必要があります。パケット・サイズは、ログイン時にネゴシエートされます。

Adaptive Server 15.0 以降の場合、接続すると、Adaptive Server ODBC ドライバを使用してサーバがパケット・サイズを選択し、パフォーマンスを最適化します。15.0 より前の Adaptive Server サーバの場合、接続すると、Adaptive Server ODBC ドライバは、packetsize プロパティを指定しない限り、パケット・サイズとして 512 を使用します。サーバによってパケット・サイズを決定しない場合は、EnableServerPacketSize を 0 に設定する必要があります。メモリの制限がある場合は、RestrictMaximumPacketSize を、Adaptive Server と Adaptive Server ODBC ドライバが指定したサイズよりも大きいパケット・サイズにネゴシエートしないような数値 (512 の倍数) に設定する必要があります。

用語解説

米国規格協会 (ANSI: American National Standards Institute) コード

米国規格協会 (ANSI: American National Standards Institute) が制定した英数字コードの標準セットで、すべての米国連邦機関に渡ってグラフィック要素の識別を統一化するものです。

Adaptive Server Enterprise (ASE)

Adaptive Server Enterprise (ASE) は、ミッションクリティカルな大量のデータを扱う環境に対応する高性能リレーショナル・データベース管理システムです。幅広いプラットフォームで最高の処理効率とスループットを発揮します。

コンパイラ

コンパイラは、特定の高級言語で記述されたソース・プログラムをマシン・コードに変換するプログラムです。

接続文字列

データ・ソースの情報と、そのデータ・ソースへの接続方法について指定する文字列。

カーソル

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。

データ・ソース名 (DSN)

データ・ソース名 (DSN) は、Open Database Connectivity (ODBC) ドライバが接続する必要がある特定のデータベースに関する情報を含んだデータ構造です。

記述子

記述子は、アプリケーションやドライバに見られる、SQL 文のパラメータまたは結果セットのカラムを記述するメタデータの集まりです。

DSURL (Directory Service URL)

Directory Service URL (DSURL) は、使用する LDAP サーバを指定するプロパティです。

分散トランザクション・プロトコル

グラフィカル・ユーザ・インタフェース (GUI: Graphical User Interface)

コンピュータのグラフィック機能を利用してプログラムを使いやすくするプログラム・インタフェース。

ISO 8859-1

ASCII ベースの標準文字エンコードである ISO/IEC 8859 のパート。略式で Latin-1 と呼ばれます。

Kerberos 認証	クライアントとサーバ間、または、あるサーバと他のサーバ間の認証や相互認証のメカニズム。
Lightweight Directory Access Protocol (LDAP)	インターネット・プロトコル (IP) ネットワークを使用した分散ディレクトリ情報サービスでアクセスおよび管理を行うためのアプリケーション・プロトコル。
Mainframe Connect DirectConnect	メインフレームや LAN ベースのデータ・ソースと完全な統合を可能にする接続ツールを提供します。
Microsoft 分散トランザクション・コーディネータ (MS DTC)	データベース、メッセージ・キュー、ファイル・システムなど、複数のリソース・マネージャに渡って、トランザクションをコーディネートするコンポーネント。
ODBC ドライバ・マネージャ	ユーザ・アプリケーションと ODBC ドライバ間の通信を管理するコンポーネント。
Open Database Connectivity (ODBC)	データベースにアクセスするために使用するオープン・スタンダード・アプリケーション・プログラマ・インタフェース (API)。
PowerBuilder	オブジェクト指向クライアント / サーバ・アプリケーション用の一般的な高速アプリケーション開発 (RAD) ツール。その構成要素は、ネットワーク内での分散配置が可能です。
プロトコル・データ・ユニット (PDU: Protocol Data Unit)	クライアントとサーバ間の通信に使用するパッケージ。
Secure Sockets Layer (SSL)/Transport Layer Security (TLS)	Secure Sockets Layers (SSL)/ Transport Layer Security (TLS) は、インターネットでの通信セキュリティを提供する暗号化プロトコルです。
Sybase EAServer	カスタムの配備をサポートする新しいモジュール・アーキテクチャを利用した、分散型 Web 対応 PowerBuilder アプリケーション用の主要なソリューション。
Sybase iAnywhere	モビリティ (モバイル・コンピューティング)、管理、セキュリティ、およびエンタープライズ・キャリバ・データベース・ソフトウェアに特化したソフトウェア・エンティティ。
Sybase Software Development Kit (SDK)	Sybase Software Developer's Kit (SDK) は、データ・ソース、情報アプリケーション、システム・サービスなどへの透過的なアクセスを可能にするプログラミング・インタフェースを提供しています。
Unicode	一貫性のあるテキストのエンコーディング、表示、および処理を実現するコンピューティング業界標準。

索引

W

え

か

き

け

こ

さ

し

す

せ

て

に

ね

は

ふ

ま

ゆ

り

わ