FUJITSU

FUJITSU Software
Symfoware Server V12.5.0

アプリケーション開発ガイド (ODBCドライバ編)

Windows/Solaris/Linux

J2UL-1758-10Z0(00)
2021年6月

まえがき

本書の目的

本書は、ODBCドライバであるODOSを利用して、Symfoware/RDBのデータベースにアクセスする方法について説明しています。

本書の読者

本書は、ODBCドライバであるODOSを利用して、Symfoware/RDBのデータベースにアクセスする方々に読んでいただくマニュアルです。
本書を読むためには、以下の知識が必要になります。

- Symfoware/RDBについての基礎知識
- Windows(R)についての一般的な知識
- ODBC対応のアプリケーション開発ツール(Excel、Visual Basicほか)についての基礎知識

参照

グローバルサーバ上のデータベースを利用する場合には、“アプリケーション開発ガイド(ODBCドライバ編)グローバルサーバ連携”を参照してください。

本書の構成

本書の構成と内容は以下のとおりです。

第1章 アプリケーション開発の概要

ODOSを利用したアプリケーション開発の概要について説明しています。

第2章 アプリケーションの設計

ODOSを利用したアプリケーションの設計時に考慮すべき点について説明しています。

第3章 環境のセットアップ

ODOSを利用するための環境のセットアップについて説明しています。

第4章 アプリケーションの作成および実行

ODOSを利用したアプリケーションの作成および実行方法について説明しています。

付録A Web連携のサンプルプログラムの使用方法

Web連携のサンプルプログラムの使用方法について説明しています。

付録B Windows(R)アプリケーションのサンプル

Windows(R)アプリケーションのサンプルコードおよび留意事項について説明しています。

付録C Webアプリケーションのサンプル

Webアプリケーションのサンプルコードおよび留意事項について説明しています。

付録D Symfoware/RDBのデータベースの資源と命名規約

ODOS固有の命名規約について説明しています。

付録E セットアップAPI

ODOSのODBCデータソースの登録をコマンドから行う方法を説明しています。

付録F パスワード変更機能

パスワード変更機能について説明しています。

付録G 使用可能SQL文一覧

ODOSを使用したアプリケーションで指定可能なSQL文について説明しています。

付録H APIリファレンス

ODOSで提供するAPIについて説明しています。

付録I DPCライブラリからの移行について

DPCライブラリからODBCドライバにアプリケーションインターフェースを移行する場合の注意事項を説明しています。

注意事項

サンプルプログラムについて

本書で説明しているサンプルプログラムについての注意事項です。

- サンプルプログラムを使用したことによるいかなる損害についても、作者、関係者および関係会社は、一切の責任を負いません。
  ユーザの自己責任において使用してください。
- サンプルプログラムを使用するためにはVisual Basicに関する基礎知識が必要です。
- サンプルプログラム中で使用されているVisual Basicのコードに関する質問には対応できません。詳細は、各自Visual Basicのマニュアルなどで確認してください。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

出版年月および版数

2021年 6月	第10版
2020年 10月	第9版
2019年 11月	第8版
2019年 2月	第7版
2016年 9月	第6版
2015年 10月	第5版
2014年 10月	第4版
2014年 8月	第3版
2013年 11月	第2版
2013年 9月	初版

著作権

目　次

第1章 アプリケーション開発の概要

本章では、ODBCドライバであるODOSを利用したアプリケーション開発の概要について説明します。

1.1 ODBCの概要

ODBC(Open Database Connectivity)は、米国Microsoft社が提唱したWindows(R)対応のアプリケーションからデータベースに接続するための標準的なインタフェースです。

Symfoware Serverでは、ODBC 3.5をサポートしています。

ODBCは、以下の構成により、標準的なインタフェースを呼び出すだけでデータベースに依存しないODBC対応のアプリケーションを作成することができます。

ODBC対応のアプリケーション

利用者の操作によってSQL文の発行と結果の取得を行うために、ODBC関数を呼び出します。

ODBCドライバマネージャ

ODBC対応のアプリケーションから呼び出されたODBC関数に対して、指定されたODBCデータソースの特定のデータベースに接続するなどの制御を行います。

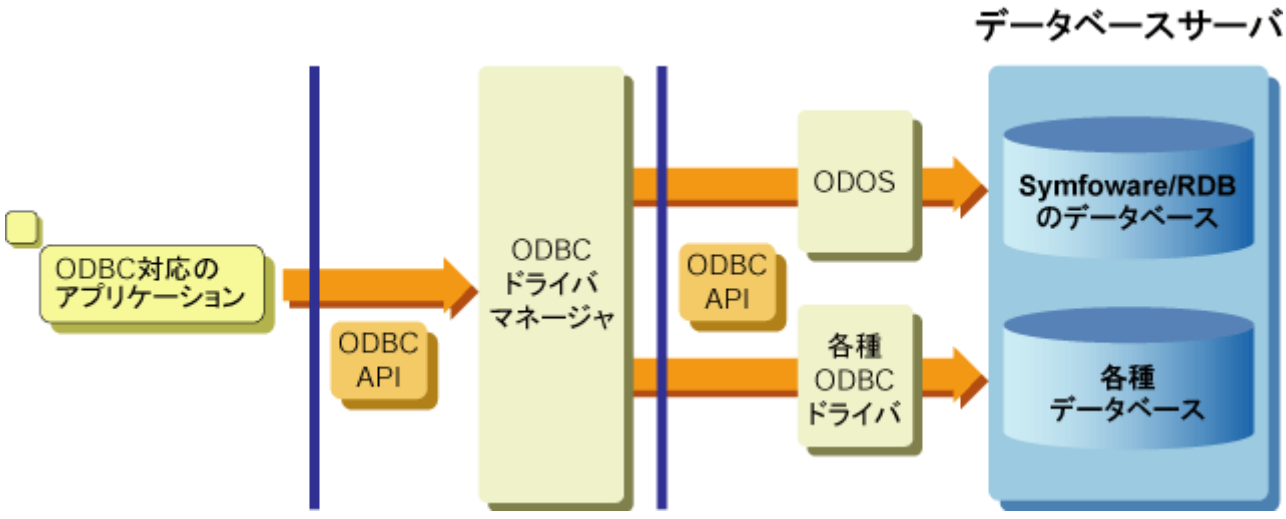
ODBCドライバ

各種のデータベースに接続するための、それぞれのデータベースに固有のドライバです。

Symfoware/RDBのODBCドライバには、ODOSがあります。

ODOS

ODOSは、ODBCインタフェースを利用してSymfoware/RDBのデータベースにアクセスできます。



1.2 運用形態

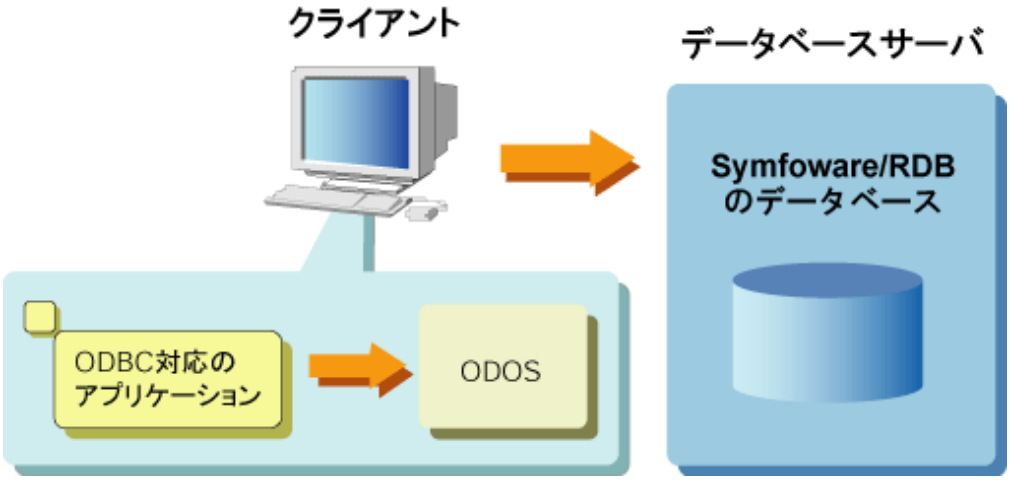
ODOSを利用する上で、Symfoware/RDBのデータベースに接続可能な運用形態には、以下の種類があります。

- 基本の運用形態
- Webサーバからの運用形態

1.2.1 基本の運用形態

ODOSは、Access、ExcelまたはVisual Basicなど、ODBCインタフェースに対応したアプリケーションで利用できます。これらの使い慣れたODBC対応のアプリケーションから、ODOSを経由してSymfoware/RDBのデータベースにアクセスすることができます。

以下に、ODBC対応のアプリケーションからのODOSの運用形態を示します。



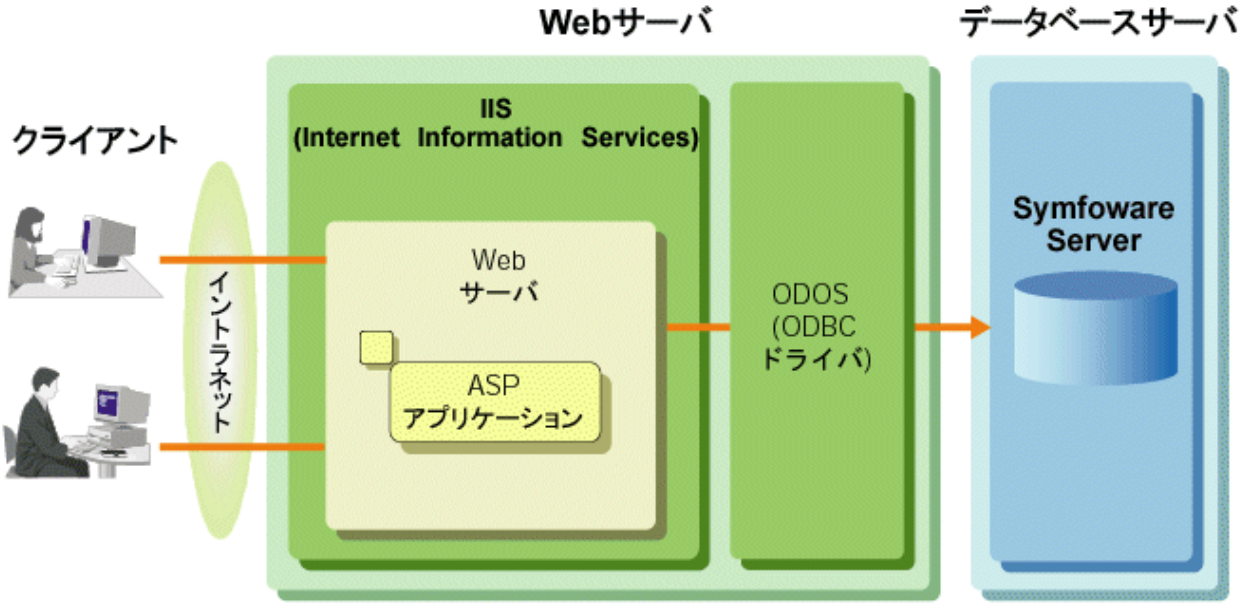
1.2.2 Webサーバからの運用形態

IIS(Internet Information Services) を利用することにより、Webページから実行されたASP(Active Server Pages)のアプリケーションは、ODOSを経由して、Symfoware/RDBのデータベースにアクセスすることができます。

これにより、Webサーバ上でSymfoware/RDBのデータベースの情報が提供可能になります。

以下に、WebサーバからのODOSの運用形態を示します。

なお、データベースサーバとWebサーバは、同一マシン上に構築することもできます。



1.3 ODBCが利用できる各種アプリケーションと機能概要

ODBCを利用した一般的なアプリケーションには、以下のものがあります。

Access

データベースのGUI操作やVBA(Visual Basic for Applications)を利用したプログラミングができるアプリケーションです。

GUIを利用してSymfoware/RDBのデータベースを操作することができます。

Excel

表計算アプリケーションです。Symfoware/RDBのデータベースから情報を取得し、帳票などが作成できます。

Visual Basic

Windows(R)対応のアプリケーションが開発できるツールです。

Symfoware/RDBのデータベースのアクセス方法として、DAO(データアクセスオブジェクト)、RDO(リモートデータオブジェクト)およびADO(ActiveXデータオブジェクト)が提供されています。

利用者が意図したSQL文を実行できるためRDOを推奨します。

Visual Basic.NET

WebアプリケーションおよびWindows(R)対応のアプリケーションが開発できるツールです。

Symfoware/RDBのデータベースのアクセス方法として、ADOおよびADO.NET (ActiveXデータオブジェクト.NET)のアクセス方法が提供されています。

IIS

ODBCとの連携でデータベースにアクセスする機能を利用できるWebサーバです。

ASPのアプリケーションを作成して、Symfoware/RDBのデータベースにアクセスすることができます。

Visual C++

C++言語でMFCライブラリ(Microsoft Foundation Class Library)を利用して、Symfoware/RDBのデータベースにアクセスするアプリケーションが作成できます。

富士通 NetCOBOLシリーズ/PowerCOBOL97シリーズ

COBOLアプリケーションを作成する開発ツールです。

使い慣れたCOBOL言語でODBCを利用したアプリケーションの開発ができます。

Macromedia ColdFusion

ODBCとの連携でデータベースにアクセスできるWebサーバです。

注意

Symfoware ODOS(UniCode)は、以下の環境で利用できます。

- Visual Basic(RDO)
- Visual Basic .NET(ADO.NET)
- Access 2013/2016/2019

Symfoware ODOS(UniCode)については、“3.1 ODBCデータソースの登録”を参照してください。

ポイント

Symfoware/RDBの利用者制御機能を利用する場合

利用者制御機能を利用する場合は、以下の環境で運用することを推奨します。

- Visual Basic（RDO、ADO）
- ASP
- Visual C++ 6.0 MFC
- Visual Basic .NET（ADO、ADO.NET）

上記以外の製品を利用する場合、Symfoware Serverから返却されるインフォメーションメッセージが表示されないことがありますので、以下の点に注意が必要です。

- ACCESS（GUI）、Excel（クエリ）

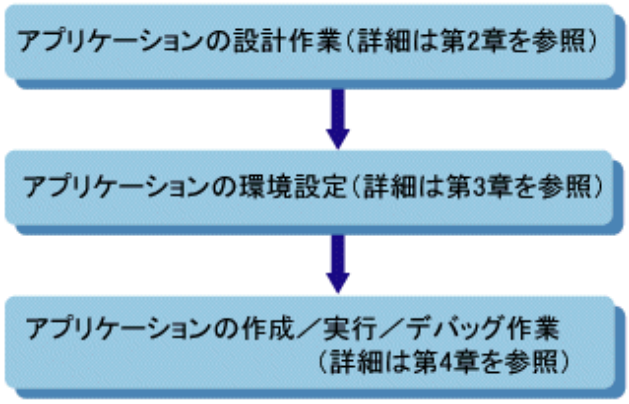
  パスワード期限を指定した場合、パスワードの変更催促期間のメッセージが出力されないため、パスワード期限に対して注意が必要です。

- ACCESS（VBA）、Excel（VBA）、Visual Basic（DAO/ODBCDirect）

  パスワード期限を指定した場合、コネクション接続後にODBC関数でメッセージを取得してください。

1.4 アプリケーション開発作業の流れ

ODBCドライバであるODOSを利用したアプリケーションを使用してシステム構築する場合の作業の流れを以下に示します。



第2章 アプリケーションの設計

本章では、ODOSを利用したアプリケーションの設計時に考慮すべき点を説明します。

2.1 コネクション

コネクションとは、ODOSを利用するアプリケーションが、Symfoware/RDBのデータベースにアクセスするために結んだ接続関係のことです。アプリケーションがデータベースシステムとコネクションをもつことにより、そのデータベースシステムの資源を操作することができます。

ODOSを利用して、アプリケーションからSymfoware/RDBのデータベースにアクセスするためのインタフェースには、以下のAPIなどがあります。

- RDO
- ADO
- ADO.NET

ODOSでは、業務の用件に応じて自由にSQL文が指定できるため、RDOを推奨しています。

2.1.1 コネクションの接続方法

アプリケーションがデータベースとのコネクションを接続・切断する方法について説明します。

コネクションの接続

アプリケーションがデータベースとのコネクションを接続するには、以下のメソッドで行います。

API名	オブジェクト名	メソッド名
RDO	rdoEnvironment	OpenConnection
ADO	Connection	Open
ADO.NET	OdbcConnection	Open

上記のメソッドには、以下に示す接続情報を含む文字列を設定します。

```
DSN=DSN01;UID=USER01;PWD=PASS01;
     (1)        (2)          (3)
```

(1) DSNキーワードには、登録したODOSのデータソース名を指定します。データソース名に、“;”、“-”、“/”、“<”、“>”および“%”は指定できません。

(2) UIDキーワードには、Symfoware/RDBに接続するための認可識別子を指定します。UIDキーワードに、“;”、“=”および“/”は指定できません。

ただし、以下の場合、UIDキーワードの指定は不要です。

- ローカルアクセスの場合
- 接続先ホスト名に自端末のIPアドレス、自端末のホスト名、“localhost”またはループバックアドレスを指定したリモートアクセスの場合

(3) PWDキーワードには、パスワードを指定します。PWDキーワードに、“;”および“=”は指定できません。

ただし、以下の場合、PWDキーワードの指定は不要です。

- ローカルアクセスの場合
- 接続先ホスト名に自端末のIPアドレス、自端末のホスト名、“localhost”またはループバックアドレスを指定したリモートアクセスの場合

コネクションの切断

アプリケーションがデータベースとのコネクションを切断するには、以下のメソッドで行います。

API名	オブジェクト名	メソッド名
RDO	rdoConnection	Close
ADO	Connection	Close
ADO.NET	OdbcConnection	Close

上記メソッドを利用して、データベースとのコネクションを接続・切断するサンプルコードの詳細については、それぞれ以下を参照してください。

- RDOを利用する場合：B.2.1 接続および切断
- ADOを利用する場合：B.3.1 接続および切断
- ADO.NETを利用する場合：B.5.1 接続および切断

2.1.2 ODBCコネクションプーリング機能

ODBCコネクションプーリング機能について説明します。

ODBCコネクションプーリング機能とは

ODBCコネクションプーリング機能 (以降、プーリング機能と呼びます)とは、アプリケーションから切断要求されたコネクションをプール中に保存しておき、次の接続要求があった場合に再利用する機能で、ODBCドライバマネージャが提供しています。

インターネット環境では、数多くのパソコン上のWebブラウザからアクセス要求が発生し、Symfoware/RDBのデータベースへの接続および切断が繰り返されます。この状況下でプーリング機能を利用すると、コネクションが再利用されてパフォーマンスが向上します。

ODBCコネクションプーリング機能とリソースプーリング機能

ADOやADO.NETを利用した場合には、ODBCドライバマネージャが提供するプーリング機能の他に、OLE DBコアコンポーネントが提供するリソースプーリング機能を利用することができます。プーリング機能とリソースプーリング機能の同時使用はできません。

ODBCドライバマネージャが提供するプーリング機能では、プール中の使用されていないコネクションを自動的に解放するまでの時間(プーリング時間)を指定することができます。一方、OLE DBコアコンポーネントが提供するリソースプーリング機能では、プーリング時間を指定することができず、OS任せになります(長時間接続されたままになります)。Webサーバへの電文を遮断してSymfoware/RDBのデータベースの保守を行うケースを想定して、プーリング時間を指定できるODBCドライバマネージャのプーリング機能を利用することを推奨します。

以降では、ODBCドライバマネージャが提供するプーリング機能の利用方法について説明しています。

プーリング機能を利用するための事前準備

プーリング機能を有効にするには、ODBCデータソースアドミニストレータを設定する必要があります。

設定する情報は以下のとおりです。

CPTimeout値

CPTimeout値が示す時間が経過すると、保持していたコネクションが自動的に解放されます。初期値は60秒です。0を指定すると、プーリング機能は無効になります。

Retry Wait値

Retry Wait値は、サーバが応答していないと判断された時に接続プールがブロックされる時間の長さ(秒単位)です。プールの中に不良な接続がある場合、ODBCドライバマネージャは、接続が最初に再使用される前に接続を試行します。なお、接続が再度失敗した場合、ODBCドライバマネージャはエラーを返し、接続を時間でマークします。その時点からRetry

Wait値が期限切れになるまで、ODBCドライバマネージャは、接続を試行することなく、失敗を返します。初期値は120秒です。

ODBCデータソースアドミニストレータの設定方法

ODBCデータソースアドミニストレータは、以下の手順で設定します。

なお、設定方法については、“ODBCデータソースアドミニストレータ 3.5”を例にとって説明します。

1. Windows(R)のコントロールパネルから、ODBCを実行します。
2. [接続プール]タブを選択します。
3. プーリング機能の利用状況が確認できるようにする場合は、[パフォーマンスモニタ]セクションで[有効にする]を選択します。
4. Retry Wait値を変更する場合は、[再試行までの待ち時間]セクションに秒単位で指定します。
5. [接続プールのタイムアウト]セクションで、Symfoware ODOSまたはSymfoware ODOS(UniCode)をダブルクリックします。
6. [このドライバに接続をプールする]を選択します。
7. CPTimeout値を変更する場合は、[未使用の接続がプール内に保持される時間]に秒単位で指定します。
8. [OK]をクリックします。

アプリケーションの記述

プーリング機能は、Visual Basicの初期値では無効に設定されています。

したがって、プーリング機能を有効にするためにアプリケーションは、コネクションを接続する前にSQLSetEnvAttr関数を発行する必要があります。また、SQLDriverConnect関数を使用して接続を行う場合は、SQL_DRIVER_NOPROMPTパラメタを指定する必要があります。プールされたコネクションは、同一のドライバの間でのみ再利用することができます。

注意

IIS、Visual Basic .NETの初期値では、プーリング機能は有効に設定されています。したがって、コネクションを接続する前にSQLSetEnvAttr関数を発行する必要はありません。

プーリング機能の確認方法

以下の方法でプーリング機能の状態を確認することができます。

- パフォーマンスモニタを有効にした場合は、パフォーマンスモニタで、[ODBC Connection Pooling]オブジェクトをグラフに追加します。

以下のようなカウンタが[ODBC Connection Pooling]オブジェクトの下に表示されます。

- Connections Currently Active(現在アクティブな接続):

  アプリケーションが現在使用している接続の数

- Connections Currently Free(現在未使用の接続):

  接続要求に対して現在使用可能な接続の数

- Connections Sec/Hard(毎秒の接続数/ハード):

  1秒ごとにODBCデータソースに対して行われる接続の数

- Connections Sec/Soft(毎秒の接続数/ソフト):

  1秒ごとのプールからの接続数

- Disconnections Sec/Hard(毎秒の切断数/ハード):

  1秒ごとにODBCデータソースに対して行われる切断の数

- Disconnections Sec/Soft(毎秒の切断数/ソフト):

  1秒ごとのプールからの切断数

プーリング機能の利用上の注意事項

以下にプーリング機能を利用する場合の注意点を示します。

- コネクションで利用できる一時表を使用する。

  一時表はアプリケーションのコネクションで利用できる表とトランザクション内で利用できる表があります。

  コネクションで利用できる一時表を使用した場合、一時表に格納したデータは、コネクションを終了したら削除する必要があります。しかし、プーリング機能を利用した場合、実際のコネクションが切断されないため、一時表に格納したデータが残ってしまいます。したがって、プーリング機能を利用する場合、一時表に格納したデータをSQLのDELETE文を使用してデータを削除させる処理をアプリケーションで行ってください。

2.2 トランザクションと排他制御

ODOSを利用したアプリケーションは、通常SQL文ごとにトランザクションが制御されます。しかし、トランザクション制御は、明示的に行うこともできます。

トランザクションモードの設定および変更は、APIのメソッドまたはSET TRANSACTION文で行います。

明示的なトランザクション制御の詳細は、以下を参照してください。

- RDOを利用する場合:B.2.9 トランザクション制御
- ADOを利用する場合:B.3.9 トランザクション制御
- ADO.NETを利用する場合:B.5.9 トランザクション制御

2.3 通信データを暗号化する場合の設定

通信データの暗号化機能を使用してリモートアクセスを行う場合は、以下の設定を行ってください。

通信内容を暗号化してサーバに接続する設定

通信の暗号化は、システム用の動作環境ファイルのSSL_USEパラメタにONが指定されている場合にのみ使用できます。クライアント側では、ODBCデータソース登録時のSymfoware ODOSセットアップ画面で、接続プロトコルとしてRDB2_TCPSを指定します。

サーバ認証を行う場合の設定

サーバのなりすましから守るために、サーバ認証を行う場合は、データベース管理者が配布するCA証明書ファイルを格納し、格納先のパスをODOSのオプションのCLI_SSL_CLI_CA_CERT_FILEパラメタに指定します。

注意

- 通信内容を暗号化してサーバに接続する設定を行わずに、CLI_SSL_CLI_CA_CERT_FILEパラメタを指定した場合は、エラーになります。
- サーバとクライアントで暗号化の指定が矛盾している場合は、エラーになります。通信内容を暗号化してサーバに接続する設定を行った場合は、システム用の動作環境ファイルのSSL_USEパラメタにONを指定してください。
- データベース簡単運用では、通信データの暗号化はできません。

2.4 アプリケーション設計時の留意事項

アプリケーション設計時の留意事項を説明します。

- SQL_LONGVARBINARYの操作について、以下の操作はできません。

  ー 同一コネクション配下の文操作は実行できません。

- ー SQL_LONGVARBINARY以降の列に対し、遅延パラメタ操作でNULLデータを設定することはできません。
- ー OBJECT構造の表に対して、BLOB32K以上の列名の後に他の列名をSQL文上に記述した場合はデータ操作することができません。BLOB列は一番最後に記述してSQL文を実行してください。

- Visual Basicを使用する場合、EXECUTEメソッドでパススルー指定を行うと更新処理が行われません。
- 一時表に対してインデックス定義は実行できません。
- 更新可能性句に列名を指定しても、全レコードが対象となります。
- スカラ関数は使用できません。
- ROW_ID(行識別子)は使用できません。

第3章 環境のセットアップ

本章では、ODOSを利用するための環境のセットアップについて説明します。

3.1 ODBCデータソースの登録

ODOSを利用してSymfoware/RDBのデータベースにアクセスするためには、ODBCデータソースの登録を事前に行います。

ODBCデータソースとは、Symfoware/RDBのデータベースを表すパソコン上の仮想的な資源です。利用者はODBCデータソースをとおして、Symfoware/RDBのデータベースにアクセスすることが可能になります。

使用するODOSの種類について

ODOSには、以下の2種類のドライバがあります。

- Symfoware ODOS
  (f3cwodbc.dll)

  シフトJISコードの文字コードの範囲で使用できます。

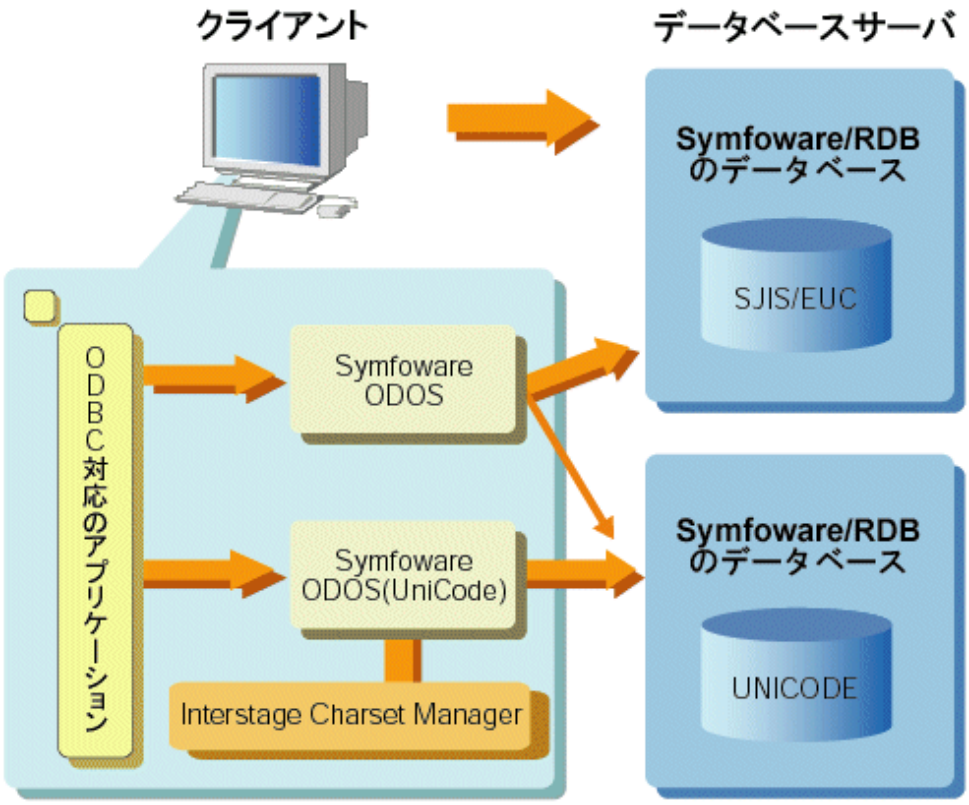
  Symfoware/RDBのデータベースの文字コード系にシフトJISコードまたはEUCコードを使用する場合に利用できます。

- Symfoware ODOS(UniCode)
  (f3cwodbu.dll)

  UNICODEの文字コードの範囲で使用できます。

  Symfoware/RDBのデータベースの文字コード系にUNICODEを使用する場合に利用できます。

  外字を使用する場合は、別途、コード変換製品(Interstage Charset Manager)が必要です。



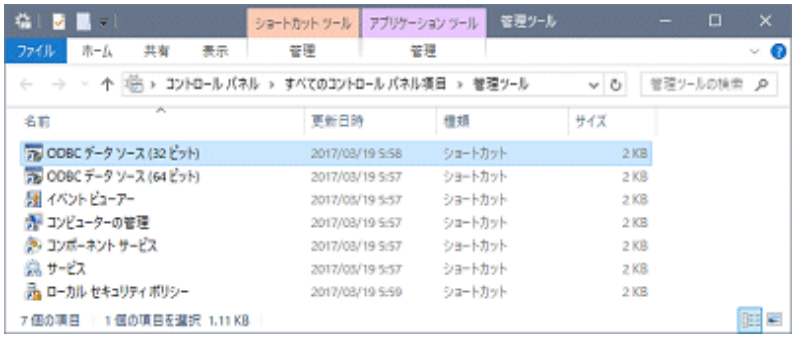
注意

Symfoware ODOSはSymfoware/RDBのデータベースの文字コード系がUNICODEのUTF-8コードの場合にも、シフトJISコードの文字コードの範囲で利用できますが、Symfoware/RDBのデータベースに格納されている文字を問題なく利用するには、Symfoware ODOS(UniCode)をお勧めします。

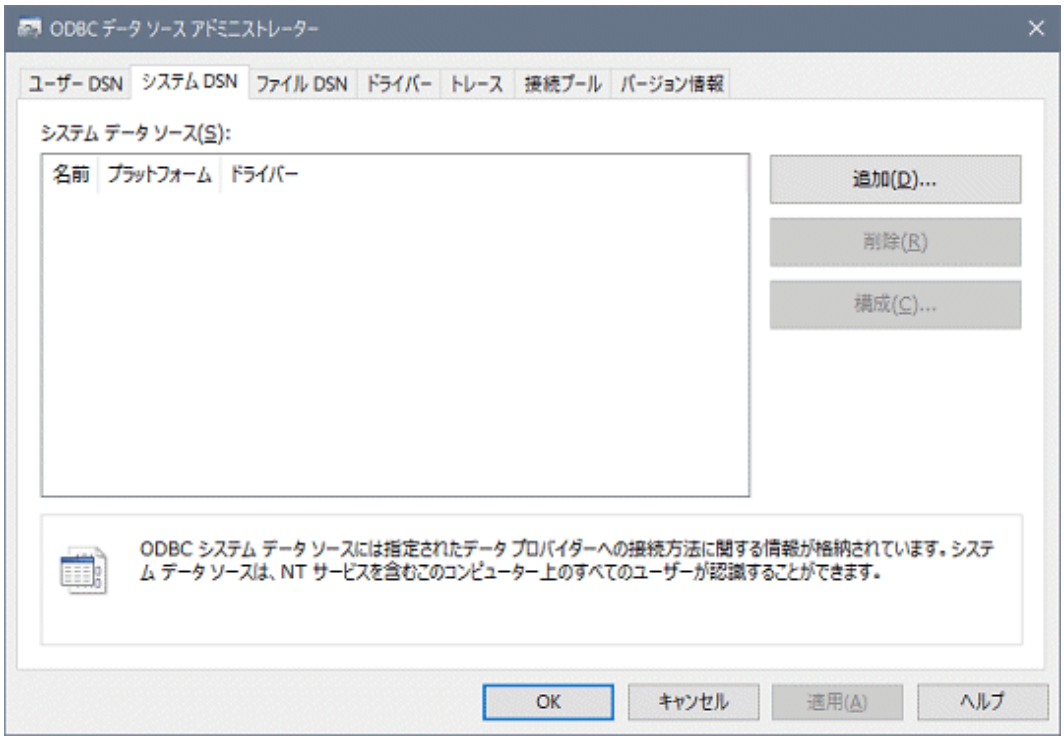
ODBCデータソースの登録手順

ODBCデータソースの登録は、以下の手順で行います。

1. [ODBC データソースアドミニストレーター]を起動します。
   [ODBC データソースアドミニストレーター]はコントロールパネルの[管理ツール]にあります。

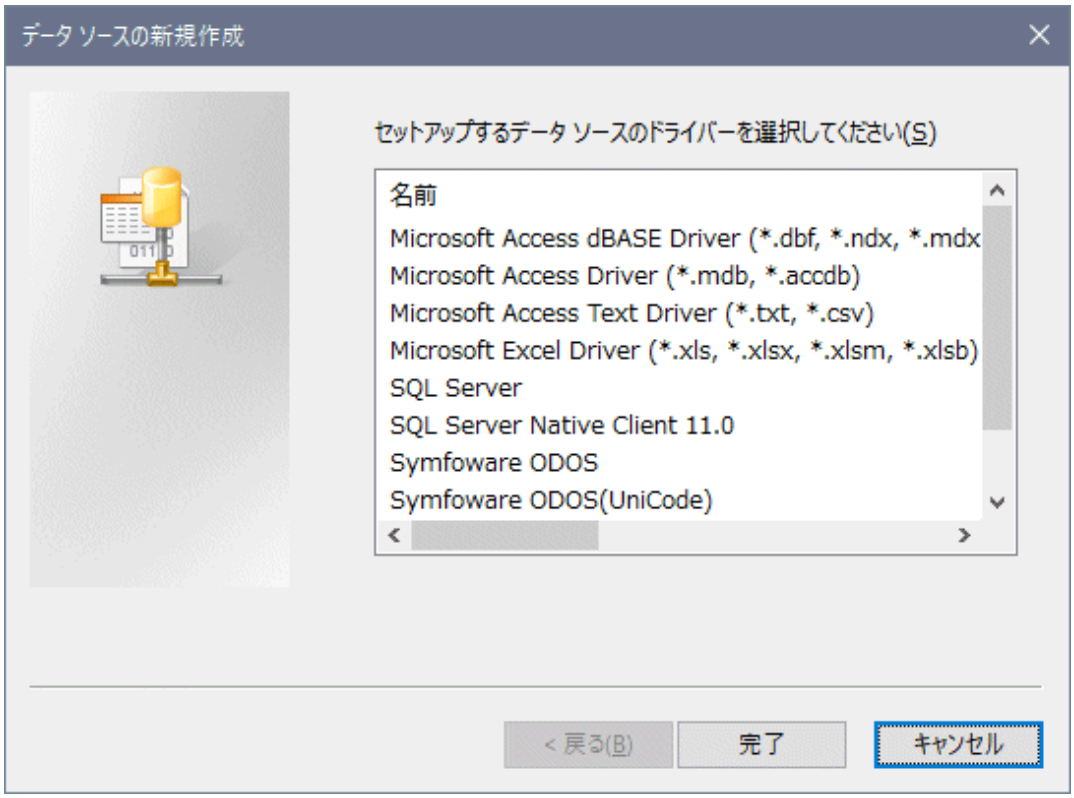


2. ODBCデータソースを現在のユーザのみが使用する場合は、[ユーザーDSN]タブを選択してください。同一のパソコンを利用するすべてのユーザが利用する場合は、[システムDSN]タブを選択してください。



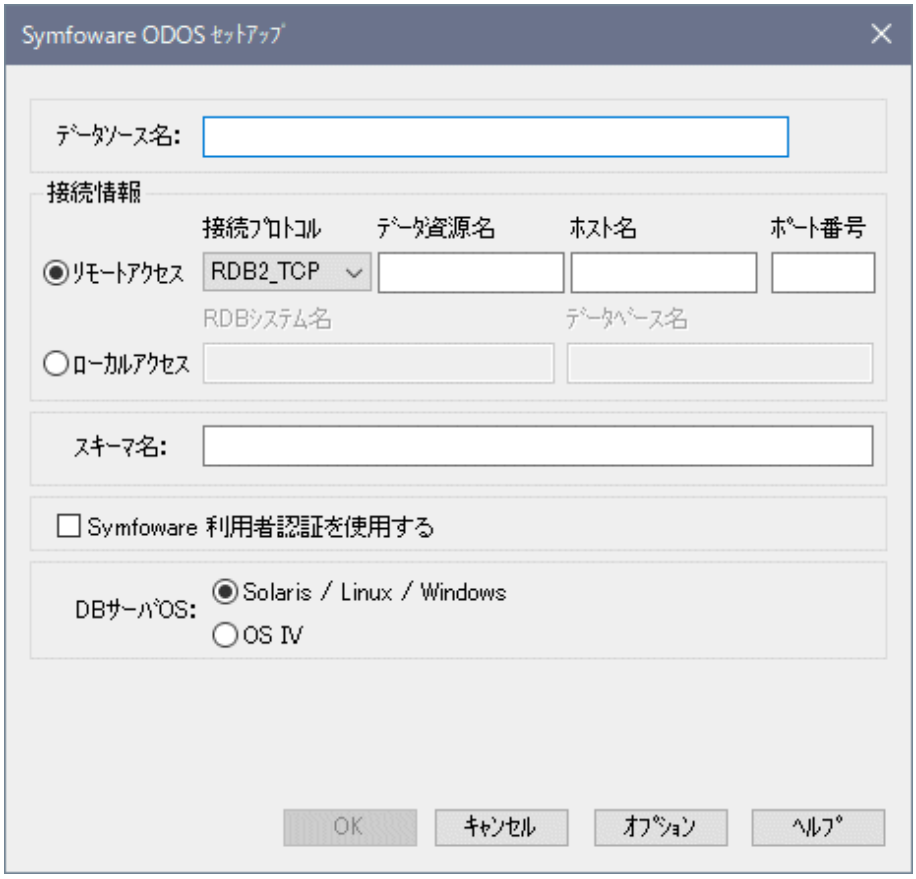
3. [追加]ボタンをクリックします。

4. [データソースの新規作成]画面で、使用可能なODBCドライバの一覧の中から“Symfoware ODOS”または“Symfoware ODOS(UniCode)”を選択して[完了]ボタンをクリックします。



5. [Symfoware ODOS セットアップ]画面が表示されますので、必要な項目を入力または選択します。また、必要な項目をすべて入力または選択した後、[OK]ボタンをクリックします。

Symfoware ODOSの場合



Symfoware ODOS(UniCode)の場合

データソース名

ODBCドライバマネージャに登録するデータソース名を指定します。アプリケーションはここで指定したデータソース名を選択してSymfoware/RDBのデータベースに接続します。本パラメタは省略できません。

32バイト以内の以下の文字が指定できます。

- 各国語文字
- 英数字
- “_”、“<”、“>”、“+”、“`”、“|”、“~”、“'”、“&”、“””、“#”、“$”、“%”、“-”、“^”、“:”、“/”、“.”

接続情報

リモートアクセスの場合

アプリケーションから別マシンのSymfoware/RDBのデータベースに接続する場合の接続方法です。

接続プロトコル

データベースへの接続形態を、以下から選択します。

・RDB2_TCP: 通信データを暗号化しない場合に選択します。

・RDB2_TCPS: 通信データを暗号化する場合に選択します。

データベース簡単運用では、通信データの暗号化はできません。

データ資源名

アクセスするSymfoware/RDBのデータベース名を指定します。
本パラメタは省略できません。

ホスト名

接続したいSymfoware/RDBのデータベースが存在するデータベースサーバのホスト名を指定します。
ホスト名はパソコン上のホスト名管理ファイルに登録してある18バイト以内のホスト名です。

本パラメタは省略できません。
ホスト名管理ファイル(hostsファイル)の設定方法は以下のとおりです。

Windows(R)インストールフォルダ配下のsystem32¥drivers¥etc¥hostsファイルを変更します。なお、hostsファイルが存在しない場合は、hosts.samファイルをhostsファイルとして同じフォルダに複写して作成してください。

hostsファイルの最後に、以下の指定例の形式でデータベースサーバの情報を追加してください。

hostsファイルの指定例(IPv4の場合)

```
123.123.123.123     SYMFO-SV
      (1)          (2)   (3)
```

hostsファイルの指定例(IPv6の場合)

```
2001:db8::20c:29ff:fe71:6ddf     SYMFO-SV
            (1)                  (2)   (3)
```

(1) データベースサーバのIPアドレス
(2) 空白
(3) データベースサーバ名

ポート番号

リモートアクセスで使用するポート番号を指定します。
本パラメタは省略できません。

Symfoware Serverのバージョンレベルや対象のプラットフォームによって、デフォルトのポート番号が異なります。以下に示すように、サーバ側で定義されているポート番号を確認の上、指定してください。

Solaris/Linuxの場合
サーバ側の/etc/servicesファイルに定義したリモートアクセスで使用するポート番号を指定します。

Windows(R)の場合
Symfoware/RDBのRDBシステムのセットアップで指定したリモートアクセスで使用するポート番号を指定します。

ローカルアクセスの場合

アプリケーションから同一マシンのSymfoware/RDBのデータベースに接続する場合の接続方法です。

RDBシステム名

マルチRDB運用をしている場合のみ、接続するRDBシステム名を指定します。

データベース名

アクセスするデータベース名を指定します。
本パラメタは省略できません。

スキーマ名

アプリケーションで実行するSQL文のスキーマ名を省略した場合のデフォルトのスキーマ名を指定します。

本パラメタを省略した場合には、接続時のユーザ名がスキーマ名の初期値として利用されます。

Symfoware利用者認証を利用する

Symfoware/RDBの利用者制御を利用する運用をしている場合にチェックしてください。

詳細については“付録F パスワード変更機能”を参照してください。

DBサーバOS

リモートアクセス時に、接続するデータベースサーバの種類を選択します。

“Solaris/Linux/Windows”を選択してください。“OS IV”は、グローバルサーバおよびPRIMEFORCEに接続する場合に選択します。

DBサーバOSは、“接続情報”の“リモートアクセス” で、“接続プロトコル”に“RDB2_TCP”を選択した時に指定可能です。

[OK]ボタン

指定した内容およびオプション値を有効にしてセットアップを終了します。

データソース名に値が指定されていない場合は使用できません。

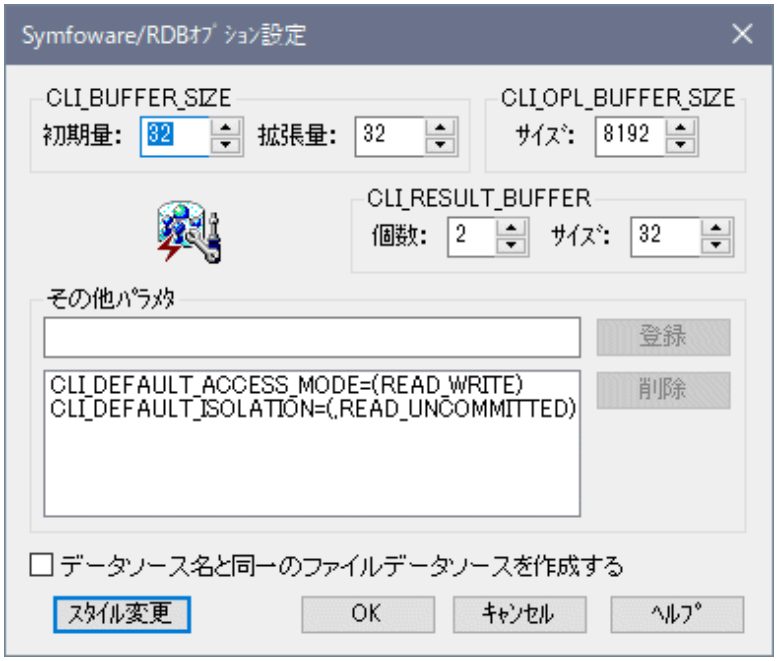
[キャンセル]ボタン

指定した内容を破棄し、セットアップを終了します。

[オプション]ボタン

以下の[Symfoware/RDBオプション設定]画面を表示します。

オプションの詳細については、[ヘルプ]ボタンをクリックして、オプション値の説明を参照してください。



パラメタの詳細については、“4.5 アプリケーションのチューニング”を参照してください。

ポイント

ODBCデータソースの登録は、上記の手順で登録することを推奨します。複数のパソコン上の環境にODBCデータソースの登録が必要な場合は、セットアップAPIを使用してください。

詳細については、“付録E セットアップAPI”を参照してください。

第4章 アプリケーションの作成および実行

本章では、ODOSを利用したアプリケーションの作成および実行方法を説明します。

4.1 Officeとの連携

OfficeのAccessまたはExcelからSymfoware/RDBのデータベースにアクセスする方法を説明します。

4.1.1 Accessの利用方法

AccessからSymfoware/RDBのデータベースにアクセスする方法を説明します。

なお、本手順は、Access 2019を使用しています。

参照

詳細な利用方法およびAccessの機能については、Accessのマニュアルを参照してください。

4.1.1.1 接続までの手順

Accessを利用してSymfoware/RDBに接続する手順について説明します。

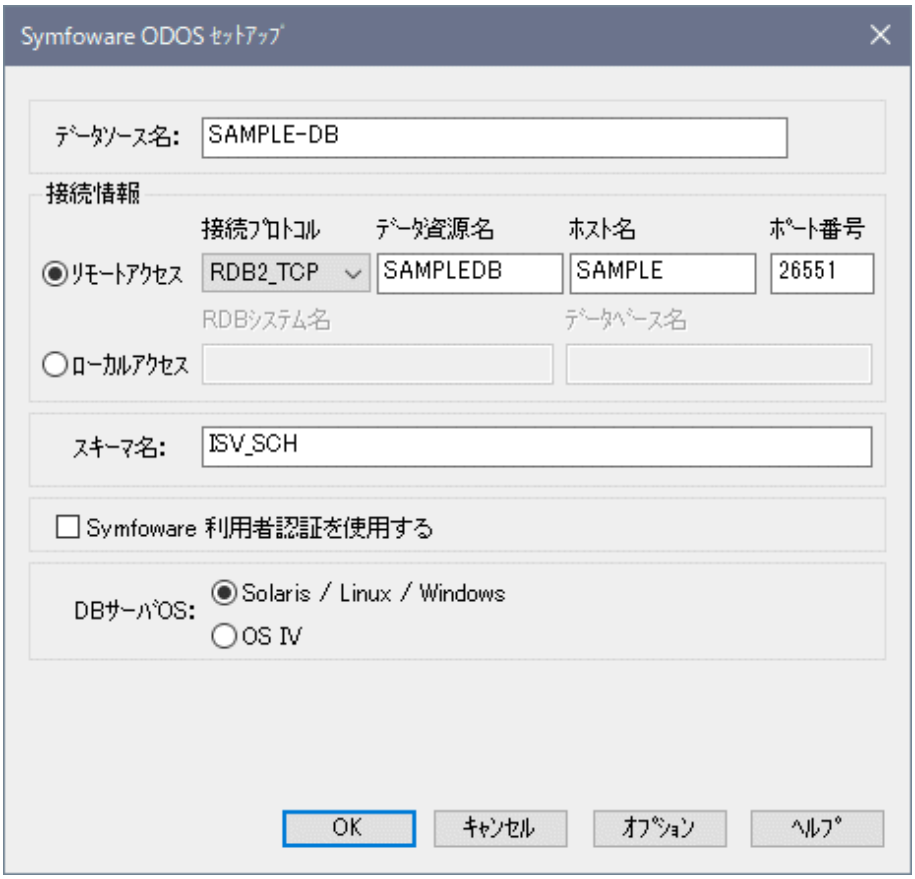
1. ODBCデータソースの登録
2. Accessのaccdb作成
3. Symfoware/RDBへの接続

1) ODBCデータソースの登録

Symfoware/RDBへ接続するためのODBCデータソースの登録を行います。

詳細については、“3.1 ODBCデータソースの登録”を参照してください。

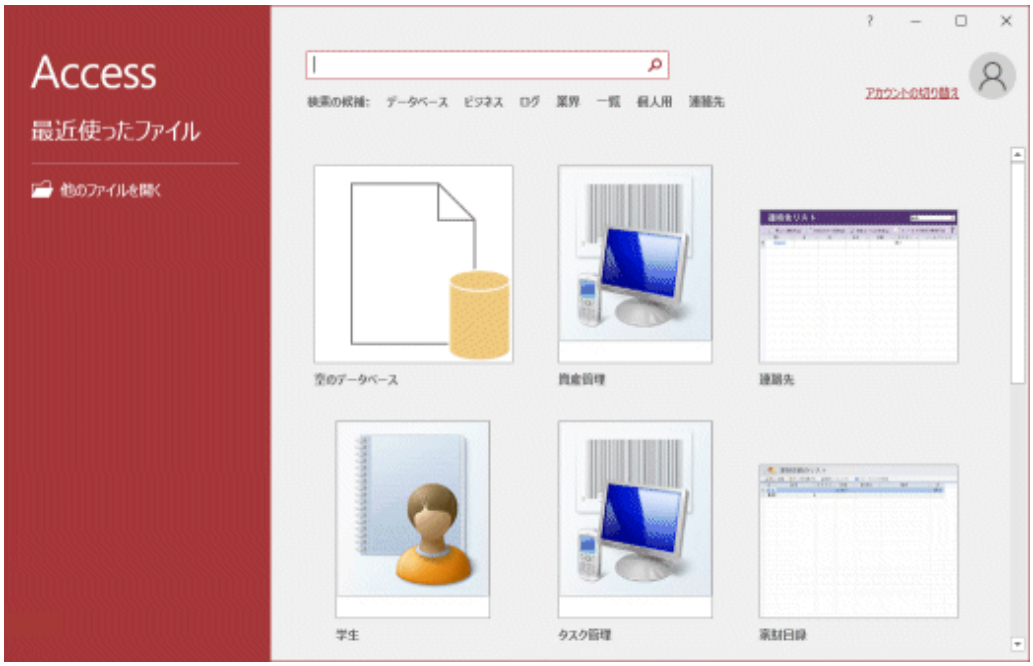
ここでは、ODBCデータソースの登録例を以下に記載します。



2) Accessのaccdb作成

接続情報などを管理するためのaccdbをAccessに作成します。

1. [Access]画面で、[空のデータベース]を選び[OK]ボタンをクリックします。

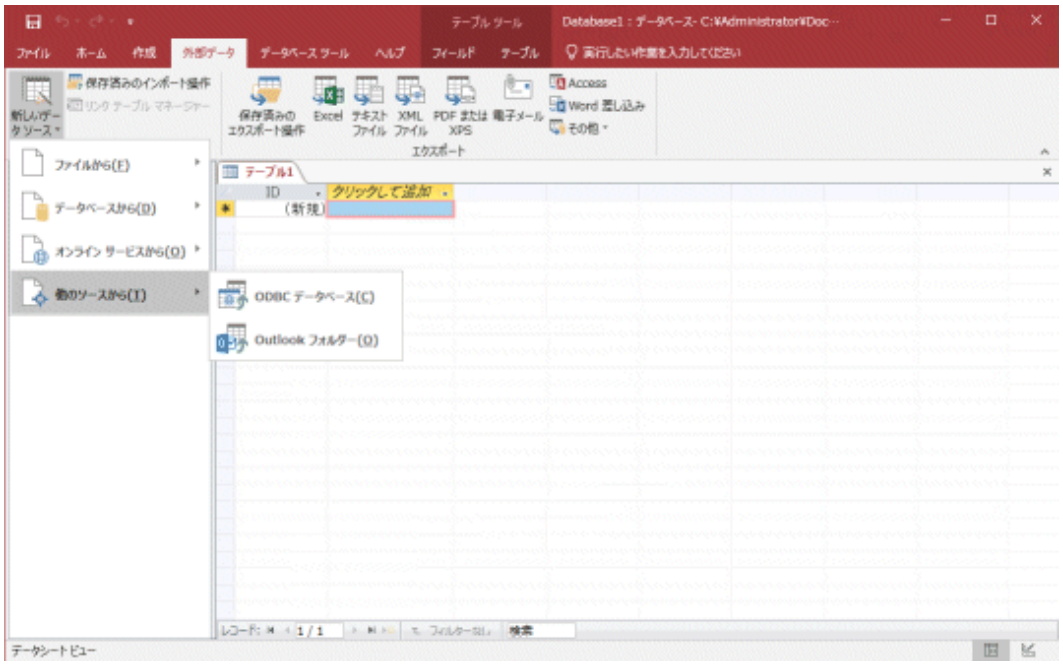


2. [空のデータベース]画面でファイル名を入力し[作成]ボタンをクリックします。

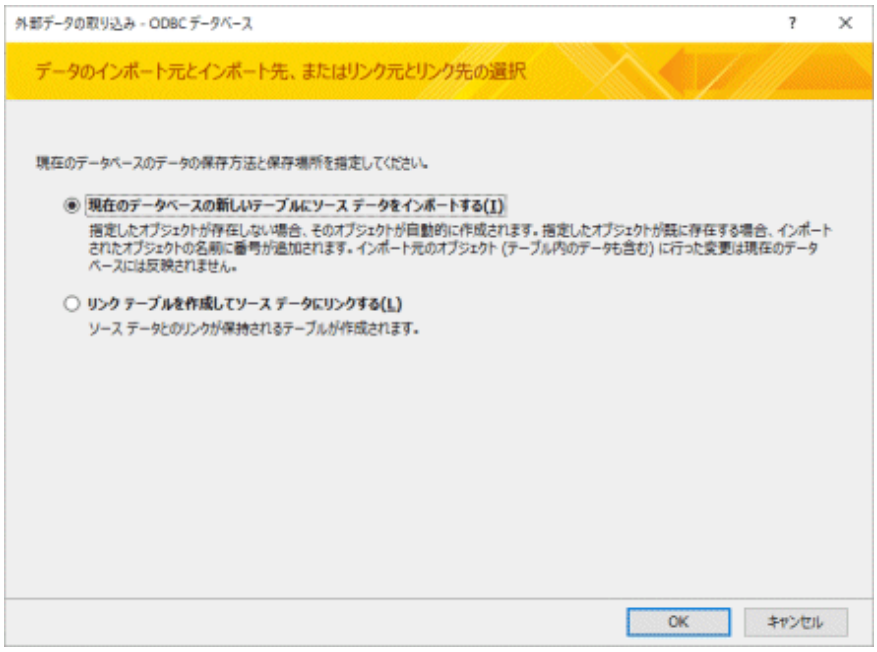
3) Symfoware/RDBへの接続

Symfoware/RDBに以下の手順で接続します。

1. Accessのメニューから[外部データ]-[新しいデータソース]-[他のソースから]-[ODBCデータベース]をクリックします。

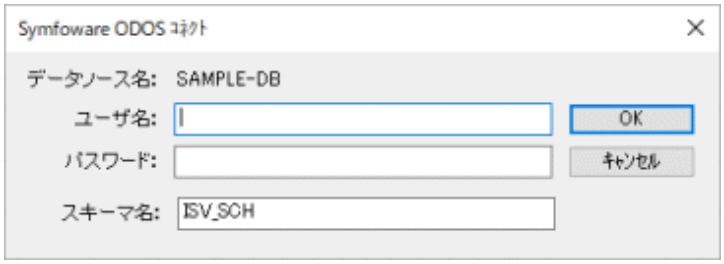


2. [外部データの取り込み-ODBCデータベース]の[データのインポート元とインポート先、またはリンク元とリンク先の選択]画面で[現在のデータベースの新しいテーブルにソースデータをインポートする]を選択し[OK]ボタンをクリックします。



3. [データソースの選択]の画面が表示されますので、[コンピューターデータソース]タブを選択し、“ODBCデータソースの登録”で登録したODBCデータソースを選択し、[OK]ボタンをクリックします。

4. [Symfoware ODOSコネクト]画面が表示されます。



リモートアクセスの場合

Symfoware/RDBで利用者の認証情報を管理している場合は、Symfoware/RDBに登録してあるユーザ名とパスワードを指定してください。

Symfoware/RDBで利用者の認証情報を管理していない場合は、データベースサーバに登録してあるユーザ名とパスワードを指定してください。

ただし、接続先ホスト名に自端末のIPアドレス、自端末のホスト名、“localhost”またはループバックアドレスを指定した場合、ユーザ名およびパスワードの指定は不要です。

省略した場合は、現在Windows(R)にログインしているユーザ名の権限で動作します。

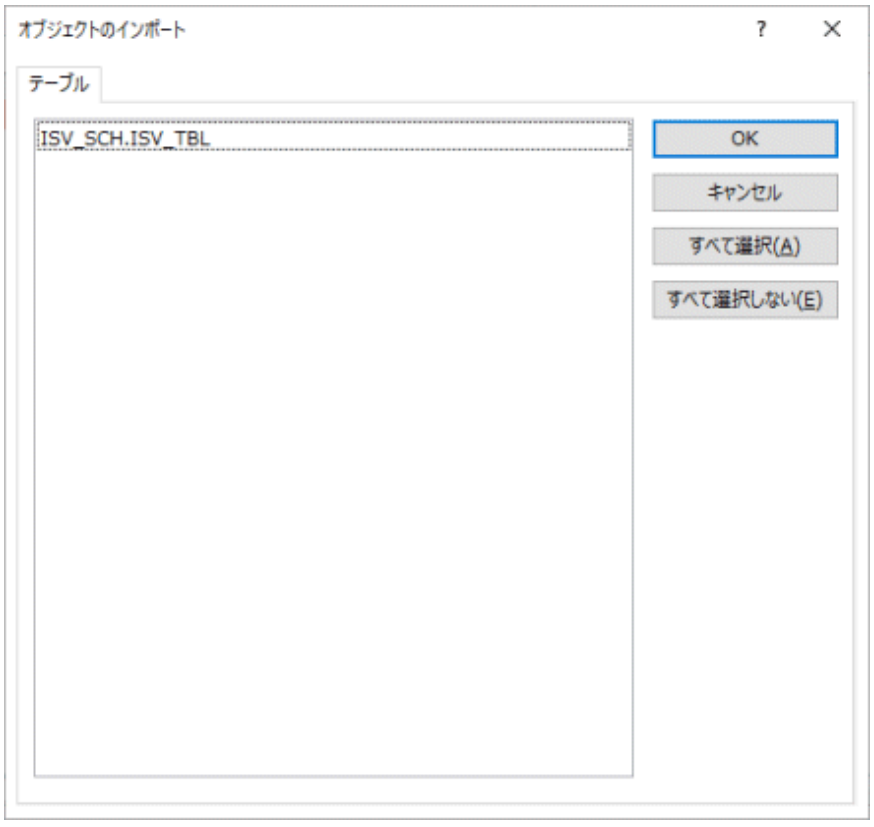
ローカルアクセスの場合

Symfoware/RDBで利用者の認証情報を管理している場合は、Symfoware/RDBに登録してあるユーザ名とパスワードを指定してください。

Symfoware/RDBで利用者の認証情報を管理していない場合は、同一のパソコン上に登録してあるユーザ名とパスワードを指定してください。または、ユーザ名とパスワードを省略してください。

省略した場合は、現在Windows(R)にログインしているユーザ名の権限で動作します。

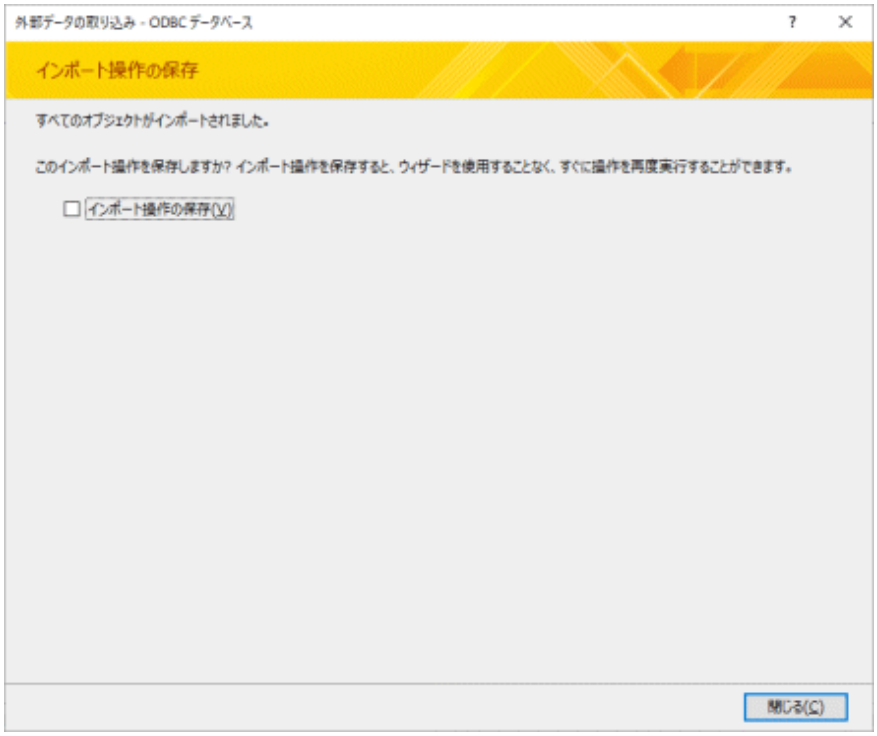
5. [オブジェクトのインポート]画面が表示されますので、該当のテーブルを選択し、[OK]ボタンをクリックします。



6. [外部データの取り込み-ODBCデータベース]の[インポート操作の保存]画面で[閉じる]ボタンをクリックします。

ポイント

インポート操作を保存する場合は、[インポート操作の保存]を選択してから、[閉じる]ボタンをクリックしてください。



以降はAccessの機能を利用し、データ操作を行ってください。

4.1.1.2 利用時の注意事項

Accessを利用してSymfoware/RDBのデータベースへアクセスする場合の注意事項を以下に示します。

Accessのクエリ機能による日時型の条件指定の書き方について

SQL文にTIMESTAMP型の定数を指定する場合には、TIMESTAMP '2007-04-01 12:53:15' と記述しますが、変数でTIMESTAMPのデータを与える場合には、時刻印を表す文字列データ'2007-04-01 12:53:15'を文字型の変数に格納して与えます。

Accessでもクエリの検索条件に指定する入力フィールドは文字列データとして扱うようになっています。

そのため、検索条件には文字列の表現で指定してください。

例

TIMESTAMP '2007-04-01 12:53:15'ではなく2007-04-01 12:53:15と入力してください。

なお、DATE型またはTIME型も同様に文字列として表現してください。

標準セキュリティ運用の場合の注意

標準セキュリティ運用時に、テーブルのリンクを作成する場合は、RDBシステムを作成したユーザを使用してください。

フィールドサイズの十進型を指定する場合の注意

数値データ型のフィールドサイズを十進型に指定する場合には、精度を18桁以内に設定してください。

BigInt型

BigInt型は未サポートのデータ型です。

参照

Symfoware Serverのデータ型については、”SQLリファレンス”を参照してください。

ODBC接続再試行ロジック

ODBC接続が失われた際の接続再試行機能は未サポートです。Accessデータベースを再起動してください。

4.1.2 Excelの利用方法

ExcelからSymfoware/RDBのデータベースにアクセスする方法を説明します。

なお、本手順は、Excel 2019を使用しています。

参照

詳細な利用方法およびExcelの機能については、Excelのマニュアルを参照してください。

4.1.2.1 接続までの手順

Excelを利用して、Symfoware/RDBへ接続する手順について説明します。

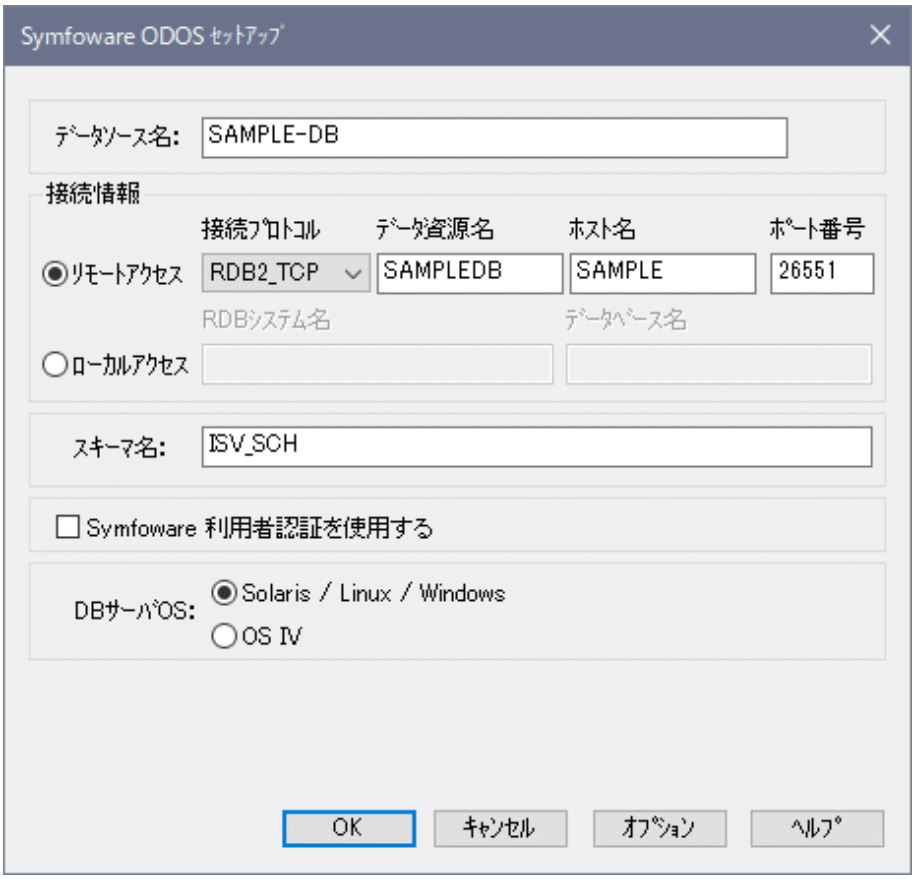
1. ODBCデータソースの登録
2. Symfoware/RDBへの接続

1) ODBCデータソースの登録

Symfoware/RDBへ接続するためのODBCデータソースの登録を行います。

詳細については、“3.1 ODBCデータソースの登録”を参照してください。

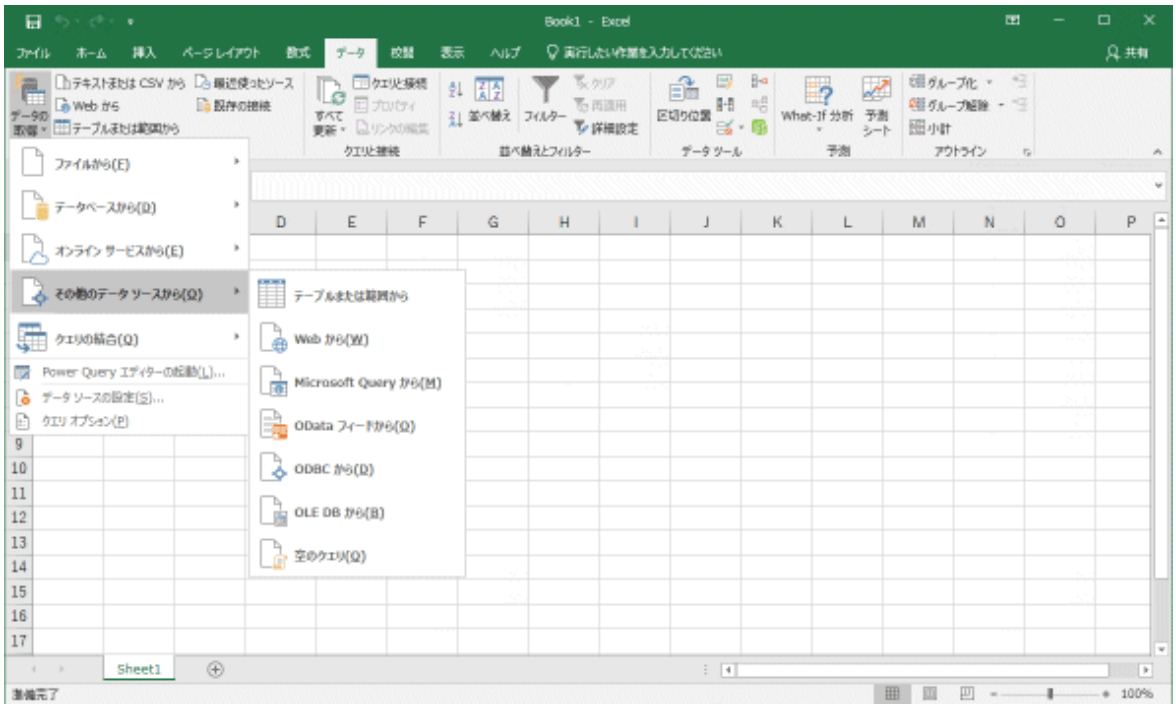
ここでは、ODBCデータソースの登録例を以下に記載します。



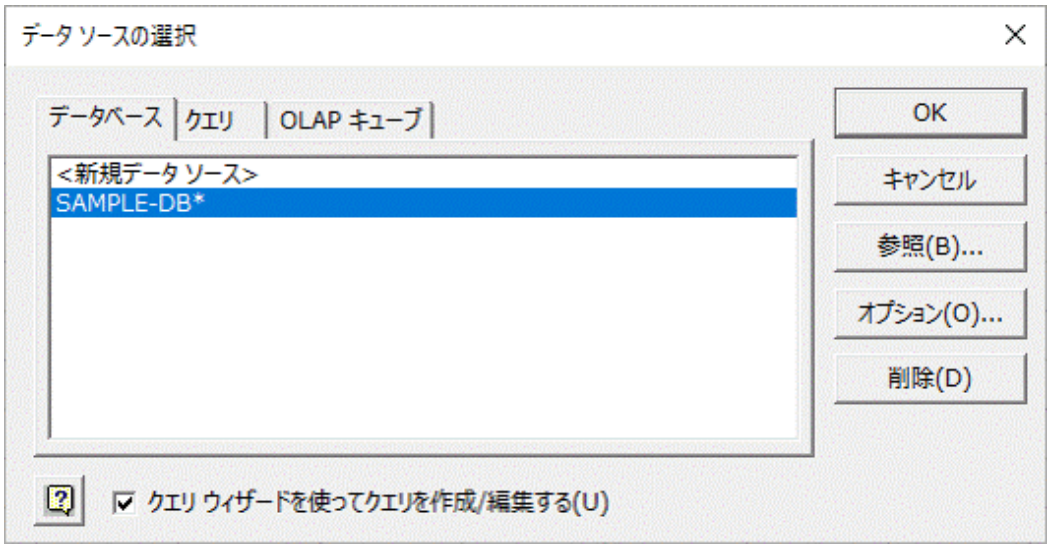
2) Symfoware/RDBへの接続

Symfoware/RDBに以下の手順で接続します。

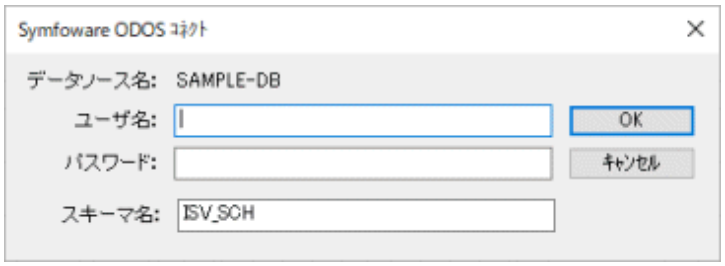
1. Excelのメニューから[データ]-[データの取得]-[その他のデータソースから]-[Microsoft Queryから]を選択し実行します。



2. [データソースの選択]画面が表示されますので、“ODBCデータソースの登録”で登録したODBCデータソースを選択し、[OK]ボタンをクリックします。



3. [Symfoware ODOSコネクト]画面が表示されます。



リモートアクセスの場合

Symfoware/RDBで利用者の認証情報を管理している場合は、Symfoware/RDBに登録してあるユーザ名とパスワードを指定してください。

Symfoware/RDBで利用者の認証情報を管理していない場合は、データベースサーバに登録してあるユーザ名とパスワードを指定してください。

ただし、接続先ホスト名に自端末のIPアドレス、自端末のホスト名、“localhost”またはループバックアドレスを指定した場合、ユーザ名およびパスワードの指定は不要です。

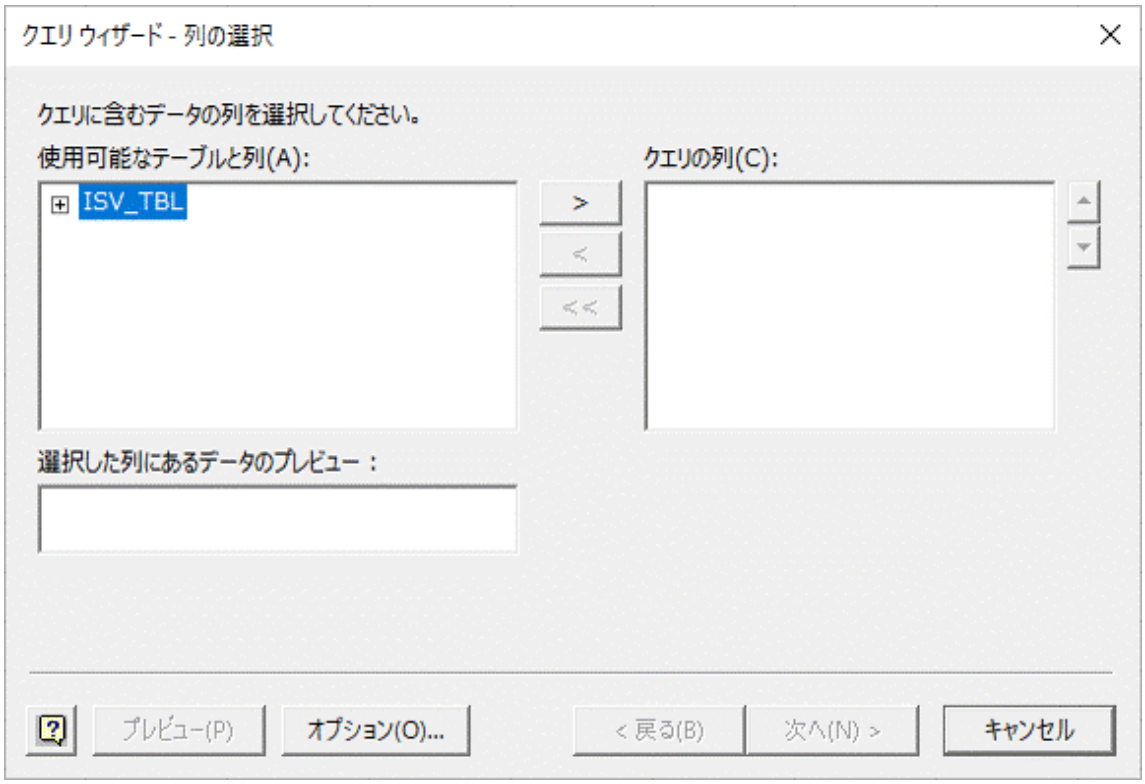
省略した場合は、現在Windows(R)にログインしているユーザ名の権限で動作します。

ローカルアクセスの場合

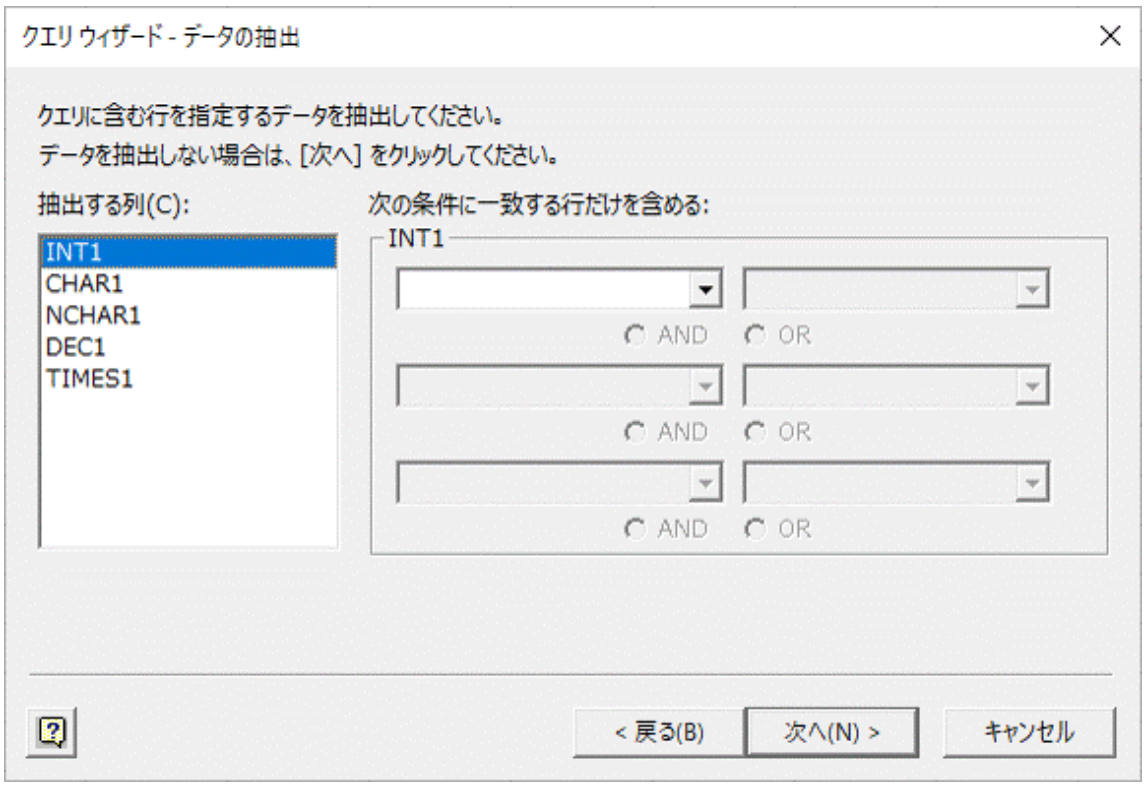
Symfoware/RDBで利用者の認証情報を管理している場合は、Symfoware/RDBに登録してあるユーザ名とパスワードを指定してください。

Symfoware/RDBで利用者の認証情報を管理していない場合は、同一のパソコン上に登録してあるユーザ名とパスワードを指定してください。または、ユーザ名とパスワードを省略してください。省略した場合は、現在Windows(R)にログインしているユーザ名の権限で動作します。

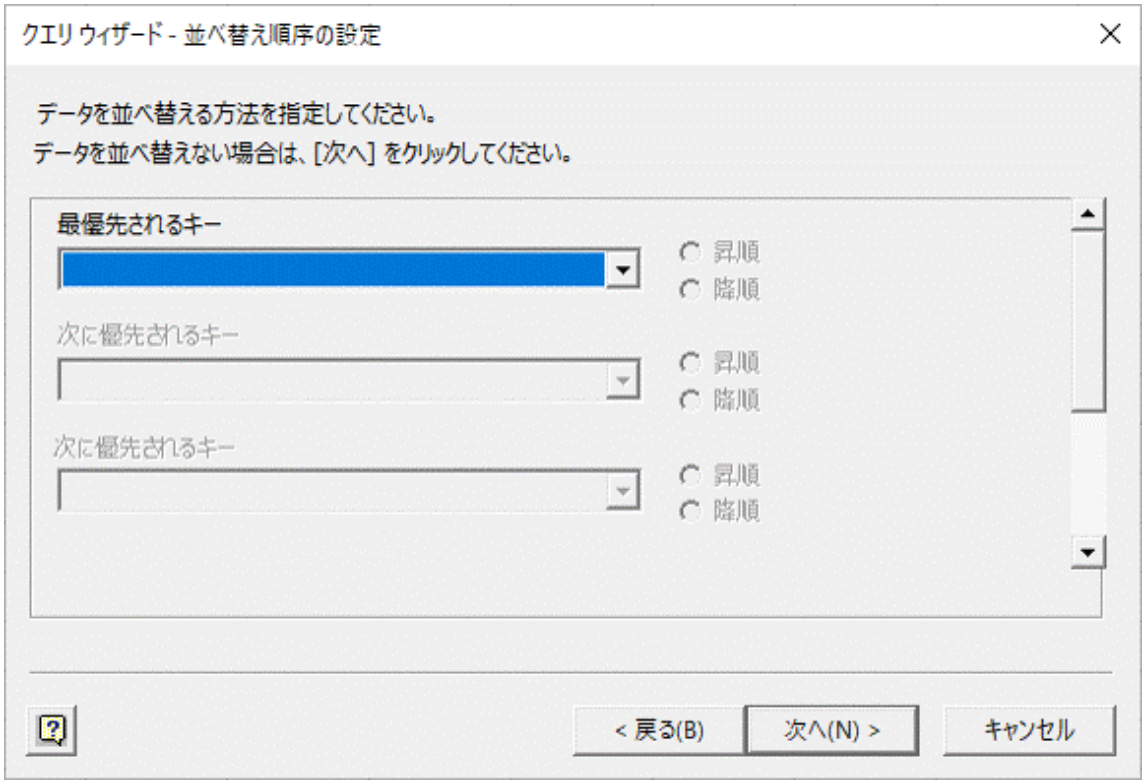
4. [クエリウィザード ― 列の選択]画面が表示されますので、該当のテーブルを選択し、[＞]ボタンをクリックします。そして[次へ]ボタンをクリックします。



5. [クエリ ウィザード ― データの抽出]画面が表示されますので、抽出条件を設定し、[次へ]ボタンをクリックします。

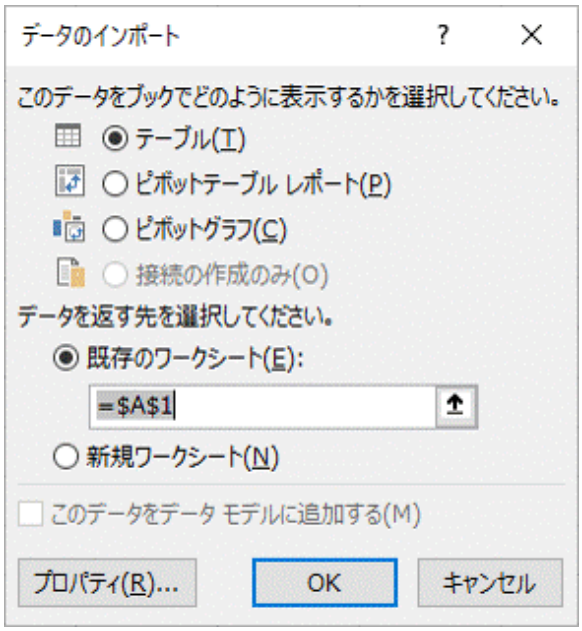


6. [クエリウィザード ー 並べ替え順序の設定]画面が表示されますので、並べ替え順序を設定し、[次へ]ボタンをクリックします。

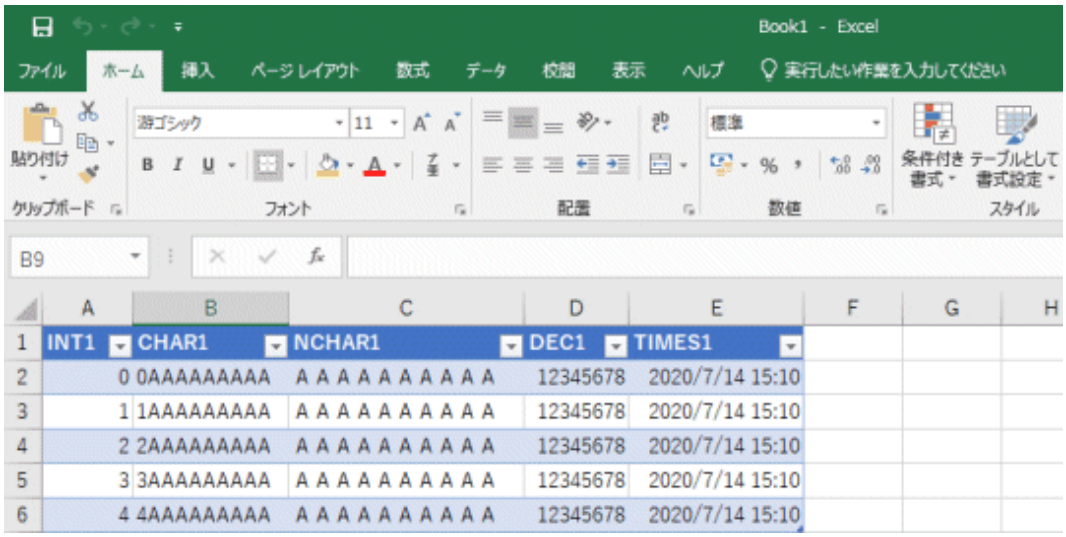


7. [クエリウィザード ー 完了]画面が表示されますので、[Microsoft Excelにデータを返す]を選択し、[完了]ボタンをクリックします。

8. [データのインポート]画面が表示されますので、データの表示方法として[テーブル]を選択し、データの返却先を設定して、[OK]ボタンをクリックします。



9. ExcelにSymfoware/RDBのデータが表示されます。



4.1.2.2 利用時の注意事項

Excelを利用してSymfoware/RDBのデータベースへアクセスする場合の注意事項を以下に示します。

日時型の条件指定の書き方について

SQL文にTIMESTAMP型の定数を指定する場合には、TIMESTAMP '2007-04-01 12:53:15'と記述しますが、変数でTIMESTAMPのデータを与える場合には、時刻印を表す文字列データ'2007-04-01 12:53:15'を文字型の変数に格納して与えます。

Excelでもクエリの検索条件に指定する入力フィールドは文字列データとして扱うようになっています。

そのため、検索条件には文字列の表現で指定してください。

例

TIMESTAMP '2007-04-01 12:53:15'ではなく2007-04-01 12:53:15と入力してください。

なお、DATE型またはTIME型も同様に文字列として表現してください。

Symfoware/RDBへの接続方法

データソースを利用しSymfoware/RDBへ接続する方法のうち、[データ]-[データの取得]-[その他のデータソースから]-[ODBCから]を利用し、接続することはできません。

4.2 Visual Basicとの連携

本章では、Visual BasicからSymfoware/RDBのデータベースにアクセスする方法を説明します。

4.2.1 概要

Visual Basicは、Windows(R)上で実行するアプリケーションを作成するためのツールです。Visual Basicを使用して、Symfoware/RDBのデータベースにアクセスするアプリケーションを開発することができます。

Visual BasicからSymfoware/RDBのデータベースにアクセスするAPIとして、以下の接続方法があります。

- RDO
- ADO
- ADO.NET
- DAO

4.2.2 必須製品

ADO.NETを利用する場合は、以下の製品が必要です。

なお、以下のプログラムは、Microsoft社のホームページより入手してください。

- ODBC.NET Data Provider
- .NET Framework SDK

4.2.3 アプリケーション作成の準備

アプリケーションを作成するまでの前作業の手順について、以下に示します。なお、本手順は、Visual Studio 2017を使用しています。

注意

データソースの作成はODBCアドミニストレータを使用してください。RegisterDatabaseおよびrdoRegisterDataSourceでの設定はできません。

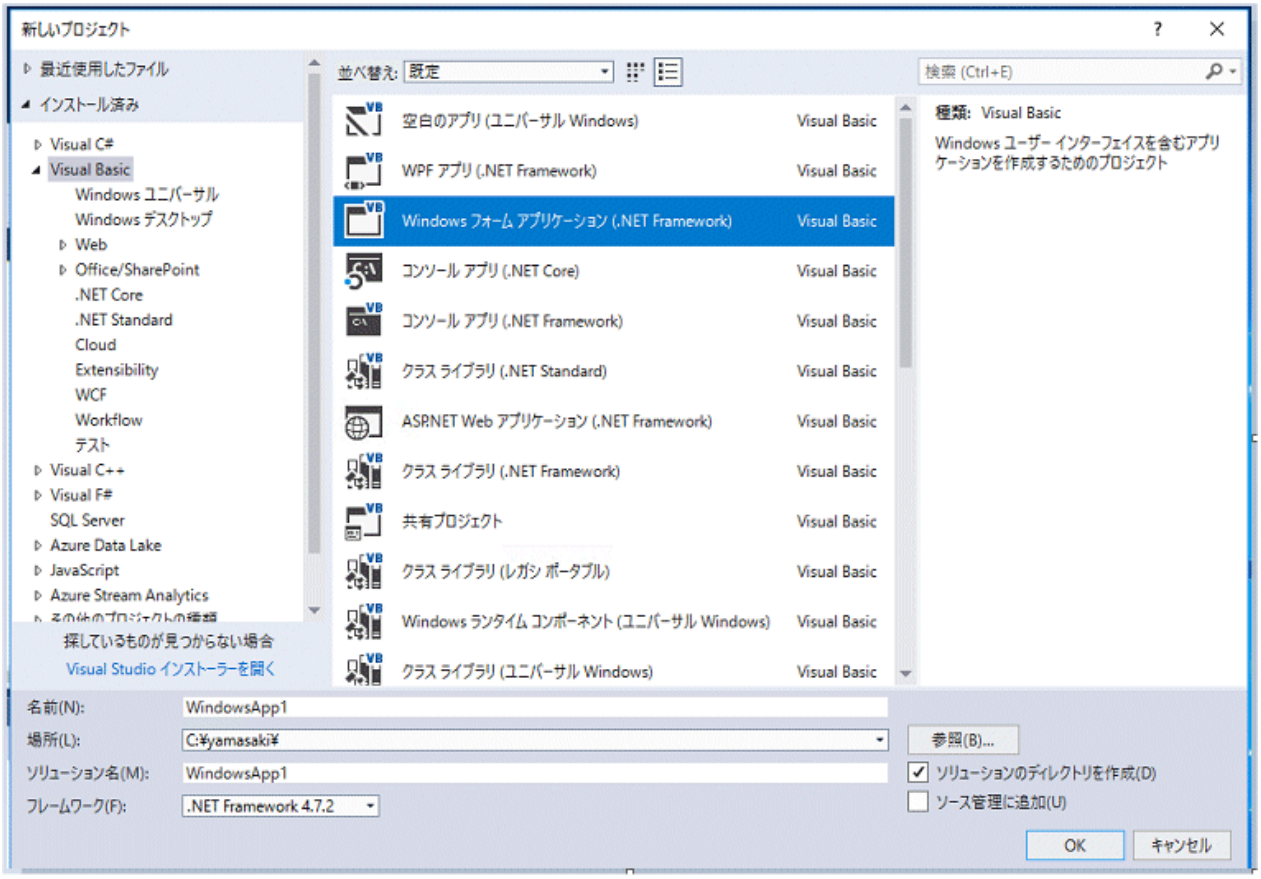
1. ODBCデータソースの作成

   Symfoware/RDBへ接続するためのODBCデータソースの作成を行います。

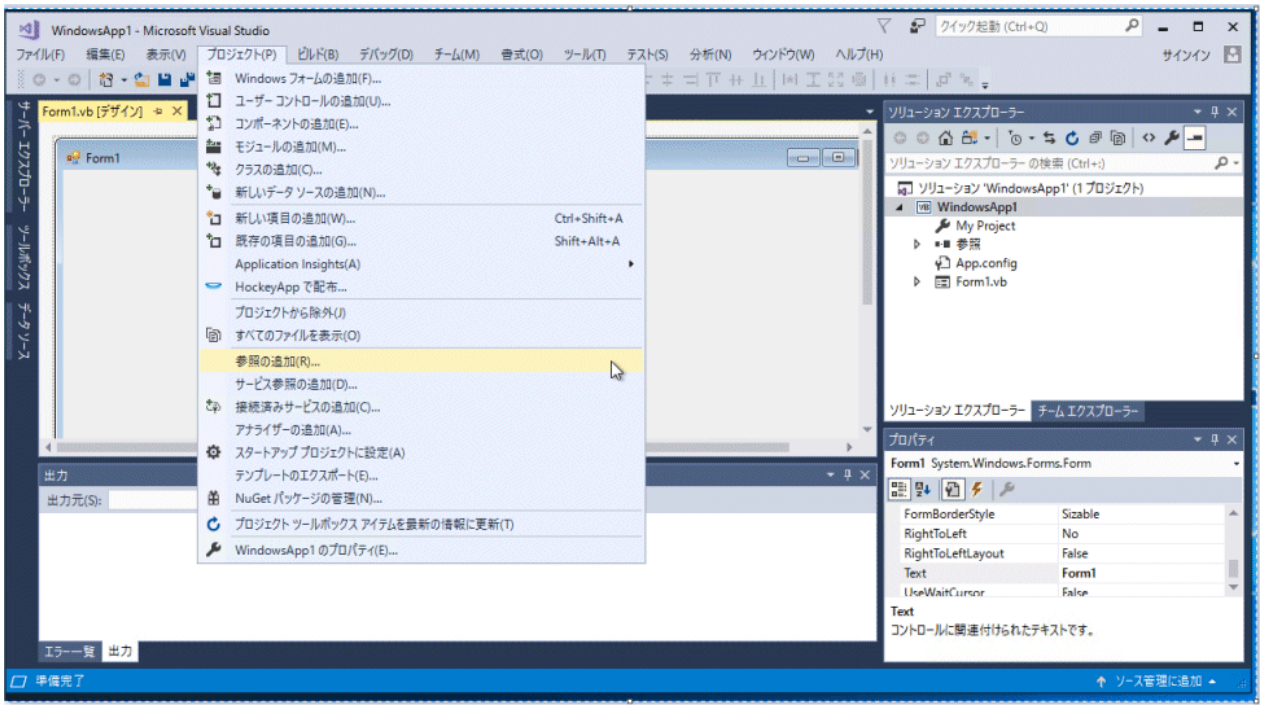
   詳細については、“3.1 ODBCデータソースの登録”を参照してください。

2. Visual Studioを起動します。
3. [ファイル(F)]メニューから、[新規作成(N)]で[プロジェクト(P)]を選択します。

4. [新しいプロジェクト]画面が表示されたら、[Visual Basic]を選択し、[Windows フォーム アプリケーション(.NET Framework)]を選択して、[OK]ボタンをクリックします。

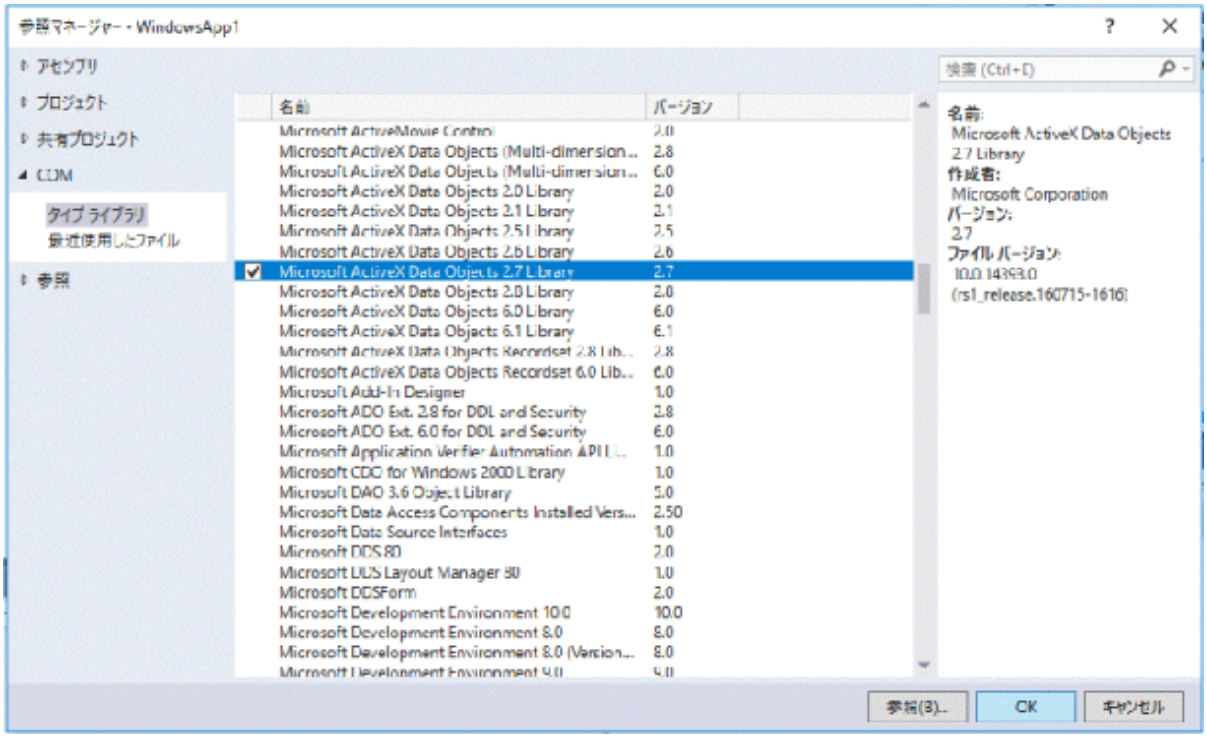


5. Visual Studioのメニューから[プロジェクト]-[参照の追加]を選択します。



6. [参照マネージャー]画面が表示されたら、以下の接続形態に合わせて必要なライブラリを選択して、[OK]ボタンをクリックします。

   ー Visual Basic.NETのADOの場合、[COM]タブを選択し[Microsoft ActiveX Data Objects 2.7 Library]を選択します。



注意

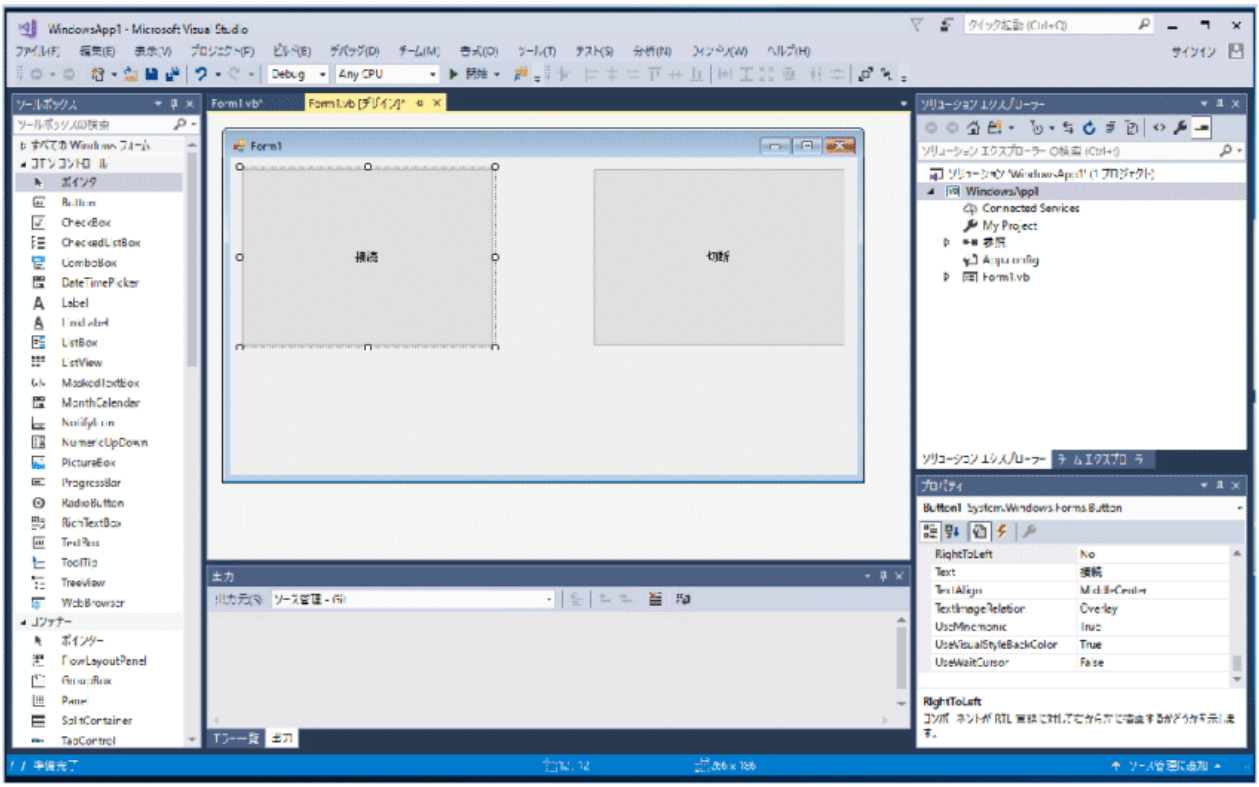
Visual Basic .NETのADO.NETの場合、[アセンブリ]タブを選択し[System.Data]を選択します。

4.2.4 アプリケーションの作成および実行

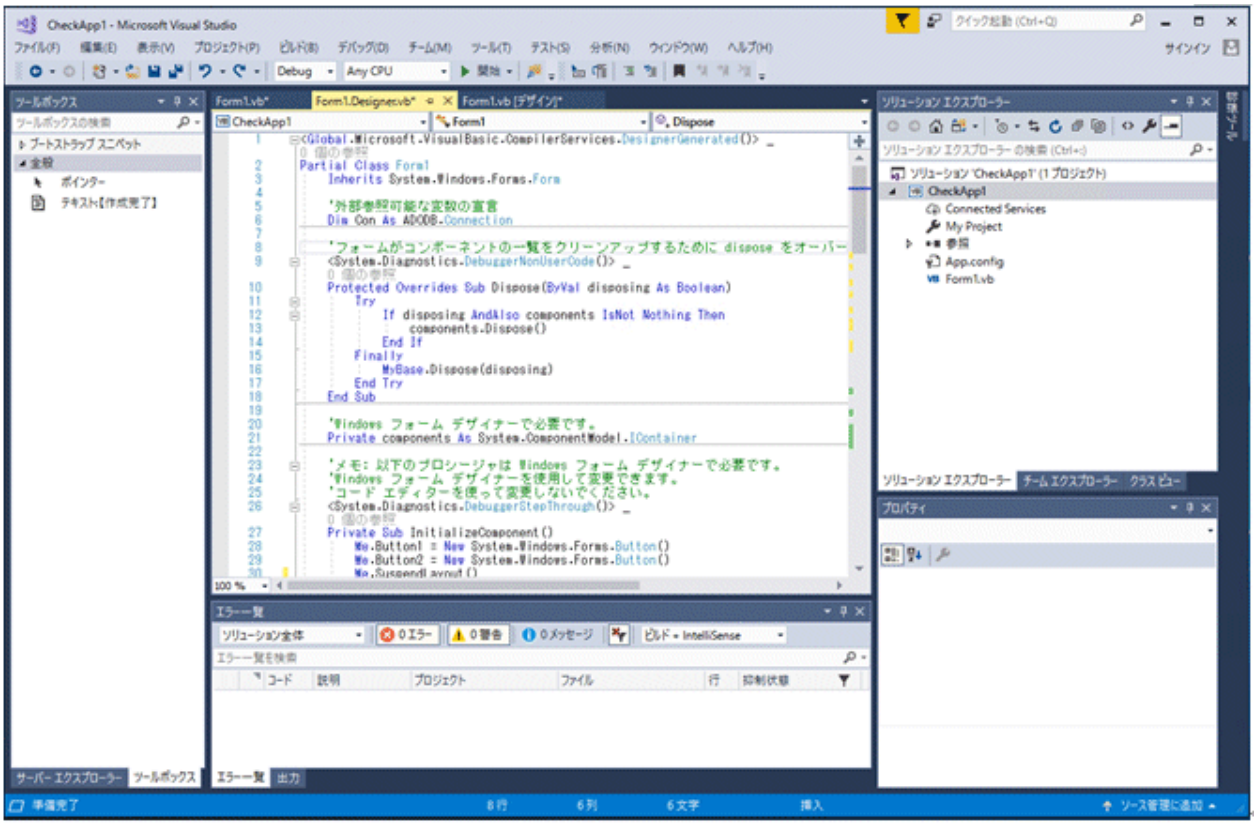
Symfoware/RDBのデータベースに接続または切断するアプリケーションを例にして、Visual Basic.Netによるアプリケーションの作成方法を説明します。なお、以下の手順は、Visual Studio 2017を使用しています。

1. 2つのコマンドボタンをフォームに貼り付けます。

2. 貼り付けたコマンドボタンの各[Text]プロパティに“接続”、“切断”と設定します。



3. フォームをダブルクリックし、「Form1.Designer.vb*」タブを開きます。



4.「Form1.Designer.vb*」に、以下のサンプルコードを入力します。

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Form1
    Inherits System.Windows.Forms.Form

    '外部参照可能な変数の宣言
    Dim Con As ADODB.Connection

    'フォームがコンポーネントの一覧をクリーンアップするために dispose をオーバーライドします。
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    …省略

    'Windows フォーム デザイナーで必要です。
    Private components As System.ComponentModel.IContainer

    'メモ: 以下のプロシージャは Windows フォーム デザイナーで必要です。
    'Windows フォーム デザイナーを使用して変更できます。
    'コード エディターを使って変更しないでください。
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        …省略

        '1. ADODB.Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=checkRDB;UID=ADMINISTRATOR;PWD=symfo000@admin;"

    End Sub
```

5. フォーム上の[接続]ボタンをダブルクリックして、カーソルが移動した場所に以下のサンプルコードを入力します。

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

        On Error GoTo ErrorProc

        Con.Open()
        MsgBox("接続できました", MsgBoxStyle.OkOnly, "接続完了")
        Button2.Enabled = True
        Button1.Enabled = False
        Exit Sub
ErrorProc:
        'Error時の処理を記述
        MsgBox("", MsgBoxStyle.OkOnly, "Error")

    End Sub
```

6. フォーム上の[切断]ボタンをダブルクリックして、カーソルが移動した場所に以下のサンプルコードを入力します。

```
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        Con.Close()
        Con = Nothing
        MsgBox("切断しました", MsgBoxStyle.OkOnly, "切断完了")
        Button2.Enabled = False
        Button1.Enabled = True

        Hide()
        Close()

    End Sub
```

以上でサンプルコードの作成は終了です。

7. メニューの[デバッグ]-[デバッグ開始]を選択するとアプリケーションを実行できます。

以下は上記で入力した全ソースです。

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _

Partial Class Form1
    Inherits System.Windows.Forms.Form

    '外部参照可能な変数の宣言
    Dim Con As ADODB.Connection

    'フォームがコンポーネントの一覧をクリーンアップするために dispose をオーバーライドします。
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Windows フォーム デザイナーで必要です。
    Private components As System.ComponentModel.IContainer

    'メモ: 以下のプロシージャは Windows フォーム デザイナーで必要です。
    'Windows フォーム デザイナーを使用して変更できます。
      'コード エディターを使って変更しないでください。
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.Button2 = New System.Windows.Forms.Button()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(12, 127)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(237, 164)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "接続"
        Me.Button1.UseVisualStyleBackColor = True
        '
        'Button2
        '
        Me.Button2.Location = New System.Drawing.Point(332, 127)
        Me.Button2.Name = "Button2"
        Me.Button2.Size = New System.Drawing.Size(237, 164)
        Me.Button2.TabIndex = 1
        Me.Button2.Text = "切断"
        Me.Button2.UseVisualStyleBackColor = True
        '
        'Form1
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 12.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(581, 382)
        Me.Controls.Add(Me.Button2)
        Me.Controls.Add(Me.Button1)
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)
```

```
        '1. ADODB.Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=checkRDB;UID=ADMINISTRATOR;PWD=symfo000@admin;"

    End Sub

    Friend WithEvents Button1 As Button

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Button2.Enabled = False
    End Sub

    Friend WithEvents Button2 As Button

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

        On Error GoTo ErrorProc

        Con.Open()
        MsgBox("接続できました", MsgBoxStyle.OkOnly, "接続完了")
        Button2.Enabled = True
        Button1.Enabled = False
        Exit Sub
ErrorProc:
        'Error時の処理を記述
        MsgBox("", MsgBoxStyle.OkOnly, "Error")

    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        Con.Close()
        Con = Nothing
        MsgBox("切断しました", MsgBoxStyle.OkOnly, "切断完了")
        Button2.Enabled = False
        Button1.Enabled = True

        Hide()
        Close()

    End Sub
End Class
```

4.2.5 アプリケーションのデバッグ

Visual Basicとの連携時に発生するエラーと対処方法を説明します。

作成したアプリケーションは、Visual Basicのデバッグ機能で十分にテストを行ってください。

参照

Visual Basicのデバッグ方法はVisual Basicのマニュアルを参照してください。

Symfoware/RDBおよびODOSのエラー(JYPxxxx)が発生した場合

Symfoware/RDBおよびODOSのエラーを取得する処理を必ず入れてください。

エラーの取得方法は、“付録B Windows(R)アプリケーションのサンプル”を参照してください。

参照

エラーメッセージについては、“メッセージ集”を参照して、対処をしてください。

ODOSのトレースの利用方法

Visual BasicではRDOやADOなどのAPIを利用してSymfoware/RDBのデータベースにアクセスしているため、Symfoware/RDBに発行しているSQL文がわかりません。

このような場合、“トラブルシューティング集”を参照し、ODOSのトレースを採取すれば、ODBCのAPIの流れや、どのようなSQL文をSymfoware/RDBに発行しているか確認できます。

参照

ODOSのトレースの採取方法については、“トラブルシューティング集”を参照してください。

また、データベース簡単運用の場合には、“トラブルシューティング集(データベース簡単運用編)”を参照してください。

4.2.6 利用時の注意事項

Visual Basicを利用してSymfoware/RDBのデータベースにアクセスする場合の注意事項を以下に示します。

VBアプリケーションで“カーソル位置が不当”というエラーが発生した場合

トランザクションを終了(コミットまたはロールバック)すると、カーソルはクローズされる仕組みとなっています。そのため、トランザクションの終了を実施した場合、必ずOpenRecordSetから実施してください。

“JYP9605E 無効な文字データが指定されました”というエラーが発生した場合

Parameterオブジェクトで数値型を指定したが、数値以外の文字を設定した場合に発生します。

以下のようなソースを記述した時に発生します。

例

```
Set OBJdbParameter = OBJdbCommand.CreateParameter
OBJdbParameter.Name = "PARA"
OBJdbParameter.Type = adInteger
OBJdbParameter.Value = "400a"
```

4.3 IISとの連携

本章では、IIS上で、ASPまたはASP.NETを利用してSymfoware/RDBのデータベースにアクセスするWebアプリケーションを作成する方法を説明します。

4.3.1 概要

IISは、Windows(R)上で動作するWebサーバです。

IISでは、ASPまたはASP.NETを利用してSymfoware/RDBのデータベースにアクセスするWebアプリケーションを作成できます。

4.3.2 必須製品

ASP.NETを利用する場合は、以下の製品が必要です。

なお、以下のプログラムは、Microsoft社のホームページより入手してください。

- ODBC.NET Data Provider

・ .NET Framework SDK

4.3.3 環境設定

ここでは、アプリケーションを実行するまでの前作業の手順について示します。

ASPアプリケーションを実行するために必要な準備について、以下の順序で説明します。

1. 格納フォルダの作成
2. IISの設定
3. ODBCデータソースの登録

1) 格納フォルダの作成

例:

C:¥ASPSample

2) IISの設定

IIS 5.0およびIIS 6.0の場合

1. インターネットサービスマネージャを起動します。
2. [既定のWebサイト]を選択し、[操作]-[新規作成]-[仮想ディレクトリ]を選択します。
3. [仮想ディレクトリの作成ウィザード]画面が表示されます。
4. ウィザードに従い、仮想ディレクトリのエイリアスを入力します。
   例:ASPSamp
5. ASPファイルを格納してあるディレクトリのパスを入力します。
   例:C:¥ASPSample
6. アクセス許可を設定します。以下の項目をチェックしてください。
   - 読み取り
   - ASPなどのスクリプトを実行する
7. インターネットサービスマネージャを終了します。

IIS 7.0およびIIS 7.5の場合

1. インターネットインフォメーションサービスマネージャを起動します。
2. Default Web Siteを右クリックし、[仮想ディレクトリの追加]を選択します。
3. [仮想ディレクトリの追加]画面が表示されます。
4. ウィザードに従い、仮想ディレクトリのエイリアスと物理パスを入力し、OKボタンをクリックすると仮想ディレクトリが作成されます。
5. 作成された仮想ディレクトリを右クリックし、[アクセス許可の編集]を選択します。
6. プロパティが表示されますので、セキュリティタブを選択し、アクセス許可を設定します。
7. インターネットインフォメーションサービスマネージャを終了します。

3) ODBCデータソースの登録

詳細については、“3.1 ODBCデータソースの登録”を参照してください。

4.3.4 アプリケーションの作成

Symfoware/RDBのデータベースに接続し、データを参照する以下のアプリケーションの作成方法について説明します。

・ ASPアプリケーション

・ ASP.NETアプリケーション

4.3.4.1 ASPアプリケーションの作成方法

ASPは、既存のHTMLファイルの拡張子の“.htm”または“.html”を“.asp”に変えるだけで簡単に作成できます。ファイルの拡張子が“.asp”のファイルはテキストファイルであり、テキストエディタを使って、以下の要素を自由に記述することができます。

・ テキスト

・ HTMLタグ

・ スクリプトコマンド

ASPのスクリプトコマンドは、区切り記号によってテキストやHTMLタグと区別されます。ASPでは、スクリプトコマンドを囲む区切り記号として“<%”と“%>”を使います。

ASPのスクリプトコマンド、またはADOのAPIの詳細については、米国Microsoft社のMSDNライブラリのホームページを参照してください。

以下に、ASPアプリケーションの例を示します。

例

以下のサンプルコードを“Samp.asp”というファイル名で作成し、“環境設定”で定義したWebサーバのフォルダに格納します。

サンプルコード中のconstr変数に設定する接続文字列と、sqlstr変数に設定するSQL文は実行環境に合わせて変更してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>データの参照</title>
</head>
<body bgcolor="White">
<b>データの参照</b>
<hr>

<%
consrc = "DSN=DSN01;UID=USER01;PWD=PASS01"
sqlsrc = "SELECT KEY_C,DEC_C FROM SAMPLE1.TESTTBL"

' エラーを取得するためエラー発生時に処理を止めないようにする
On Error Resume Next

'0. Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = consrc
Response.Write "１. コネクション接続<br>"
OBJdbConnection.Open

'エラー処理
If OBJdbConnection.Errors.Count > 0 Then
     Response.Write "Description = " & _
                    OBJdbConnection.Errors(0).Description & "<br>"
     Response.Write "NativeError = " & _
                    OBJdbConnection.Errors(0).NativeError & "<br>"
     Response.Write "Number = " & _
                    OBJdbConnection.Errors(0).Number & "<br>"
     Response.Write "Source = " & _
                    OBJdbConnection.Errors(0).Source & "<br>"
     Response.Write "SQLState = " & _
                    OBJdbConnection.Errors(0).SQLState & "<br>"
End If
```

```
Response.Write "２．Recordsetオブジェクトの生成<br>"
Set RecordSetObj = Server.CreateObject("ADODB.Recordset")

Response.Write "３．Recordsetオブジェクトを開く<br>"
RecordSetObj.Open sqlsrc,OBJdbConnection,adOpenForwardOnly, _
                  adLockReadOnly,adCmdText

'エラー処理
If OBJdbConnection.Errors.Count > 0 Then
     Response.Write "Description = " & _
                    OBJdbConnection.Errors(0).Description & "<br>"
     Response.Write "NativeError = " & _
                    OBJdbConnection.Errors(0).NativeError & "<br>"
     Response.Write "Number = " & _
                    OBJdbConnection.Errors(0).Number & "<br>"
     Response.Write "Source = " & _
                    OBJdbConnection.Errors(0).Source & "<br>"
     Response.Write "SQLState = " & _
                    OBJdbConnection.Errors(0).SQLState & "<br>"
Else

Response.Write "４．データの取得（最終行まで繰り返し）<br>"
Response.Write "KEY_C,DEC_C<br>"
Do While Not RecordSetObj.EOF
    For i = 0 To RecordSetObj.Fields.Count - 1
        'データの取得
        Response.Write CStr(RecordSetObj.Fields(i)) + ","
    Next
    Response.Write "<br>"
    RecordSetObj.MoveNext
Loop
End If

Response.Write "５．Recordsetオブジェクトを閉じる<br>"
RecordSetObj.Close()

Response.Write "６．コネクション切断<br>"
OBJdbConnection.Close

%>

<hr>
</body>
</html>
```

ASPアプリケーションの実行方法

ブラウザのアドレスにASPファイル(拡張子“.asp”のファイル)を格納したURLを指定すると実行できます。

```
http://Webサーバ名/ASPSamp/Samp.asp
```

実行結果は以下のようになります。

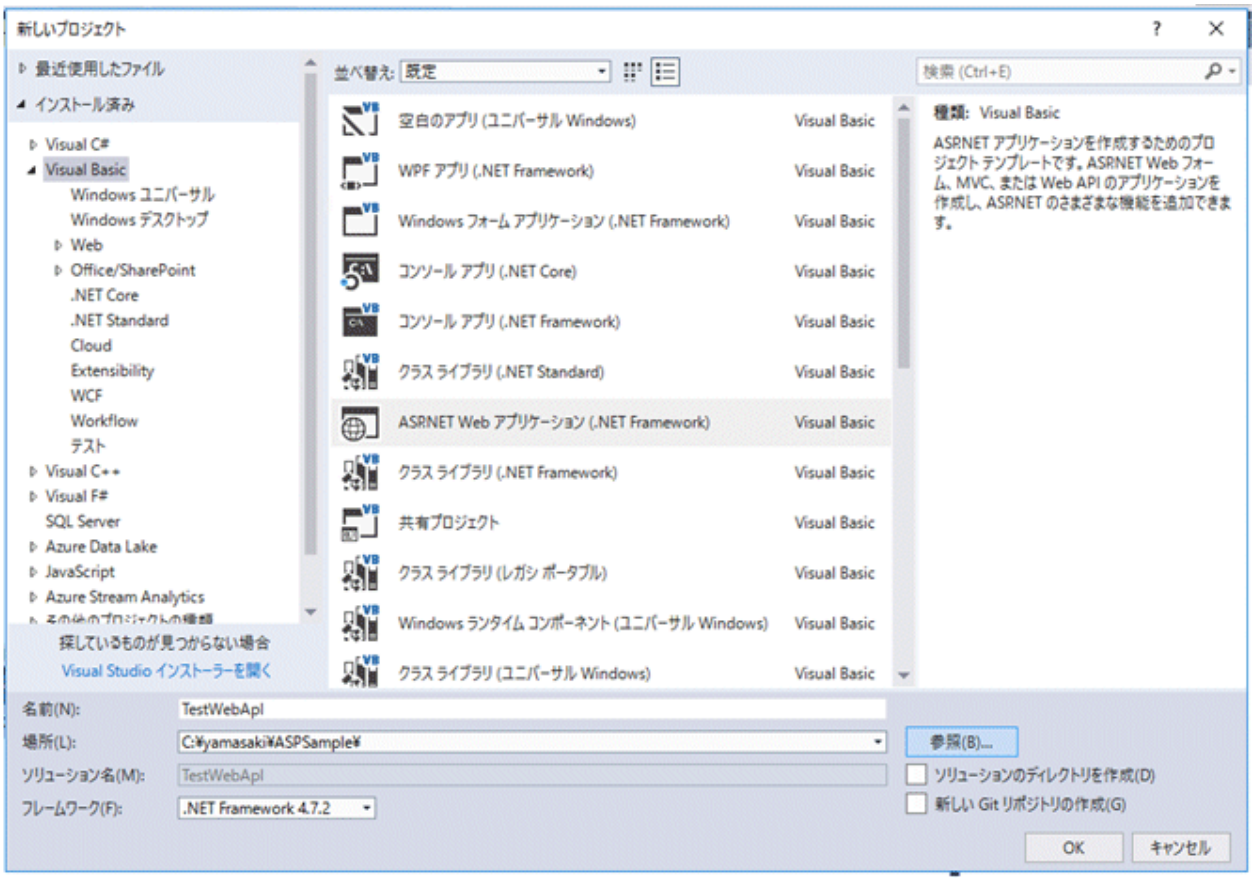
```
データの参照
-------------------------------------------------------------
1．コネクション接続
2．Recordsetオブジェクトの生成
3．Recordsetオブジェクトを開く
4．データの取得（最終行まで繰り返し）
KEY_C,DEC_C
500,2000.222,
100,1000.025,
5．Recordsetオブジェクトを閉じる
6．コネクション切断
-------------------------------------------------------------
```

4.3.4.2 ASP.NETアプリケーションの作成方法

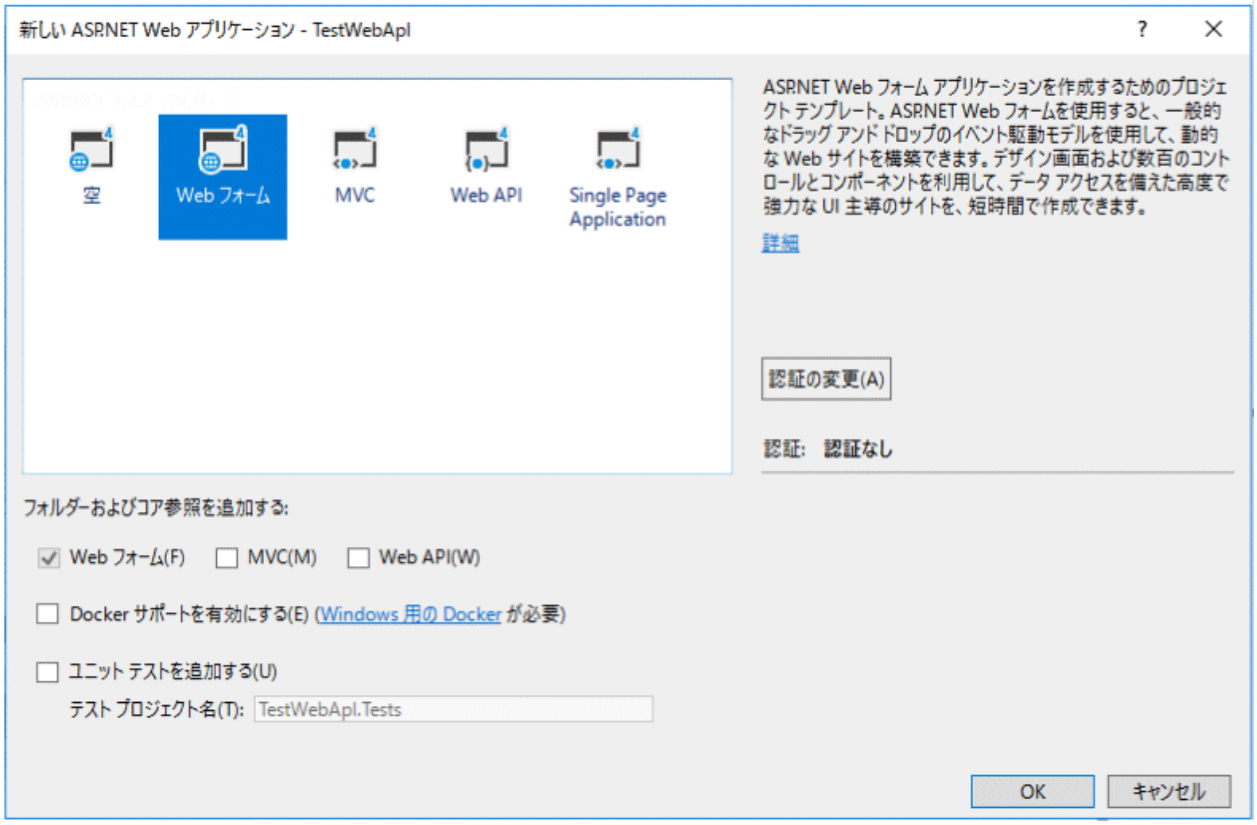
Symfoware/RDBのデータベースに接続するWebアプリケーションを例にして、Visual Basic .NETによるWebアプリケーションの作成方法を説明します。

なお、以下の手順はVisual Studio 2017を使用しています。

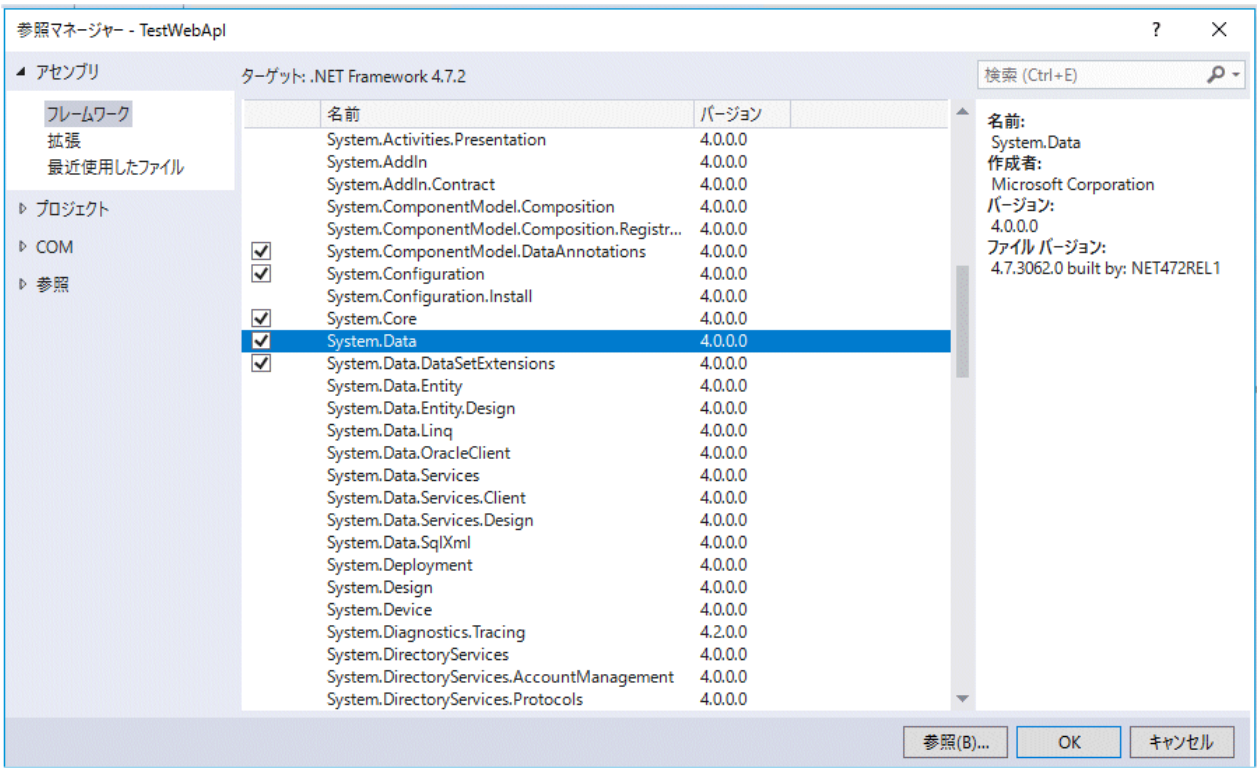
1. Visual Studioを起動します。
2. 新しいプロジェクトを作成します。[新しいプロジェクト]画面で、[Visual Basic]を選択し、テンプレートから[ASP.NET Webアプリケーション]を選択します。[名前]に、プロジェクト名(任意 例:TestWebApl)、[場所]に、“環境設定”で作成したエイリアス(例:ASPSamp)がある物理パスを指定し、[OK]ボタンをクリックします。



3. Webフォームを選択し、[OK]ボタンをクリックします。



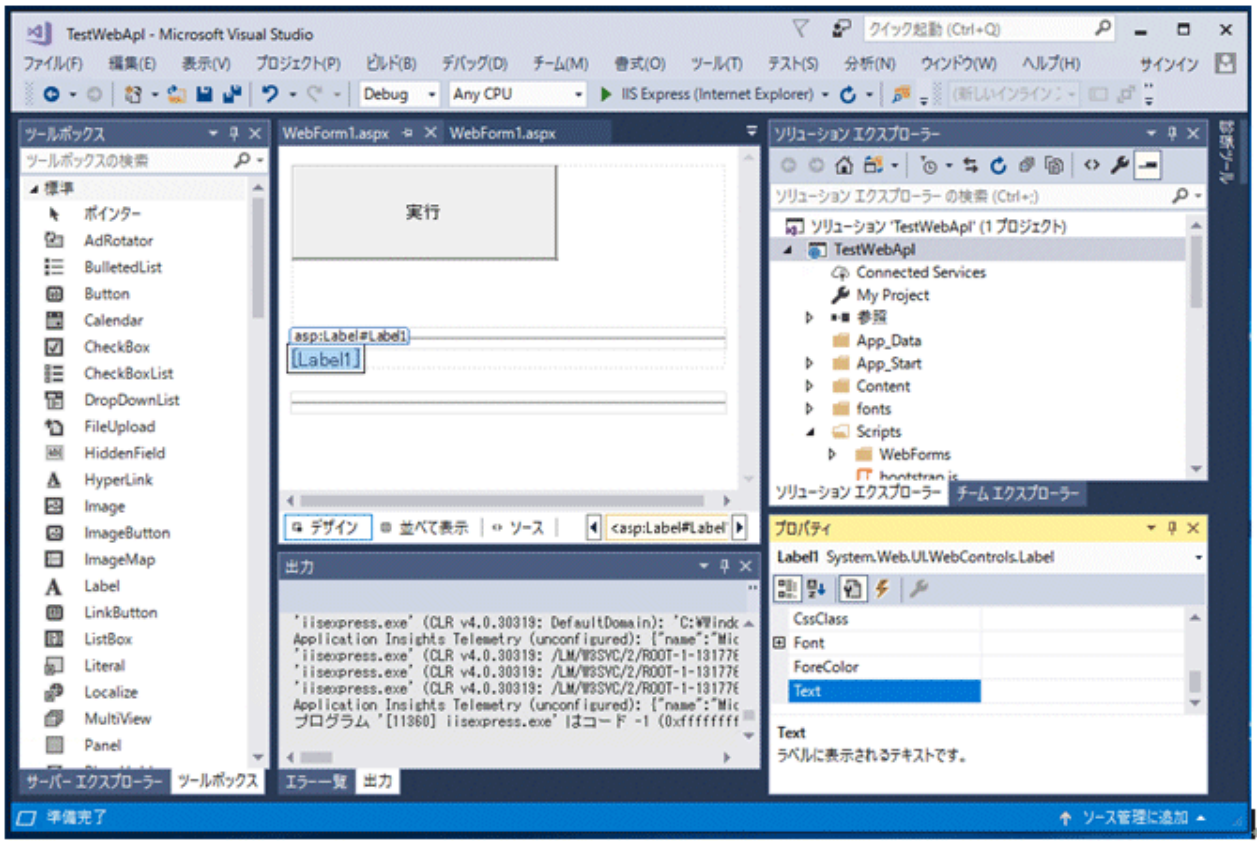
4. Visual Studioのメニューから、[プロジェクト]－[参照の追加]を選択します。[参照マネージャー]画面が表示されたら、[アセンブリ]タブを選択し、フレームワークの一覧から[System.Data]を選択して[OK]ボタンをクリックします。



注意

ADOの場合、[COM]タブを選択し[Microsoft ActiveX Data Objects 2.7 Library]を選択します。

5. Webフォームを編集します。Webフォームにボタンコントロール、ラベルコントロールおよびHorizontal Ruleを貼り付け、以下のようなWebフォームを作成します。ボタンコントロールのTextプロパティを、「実行」と設定します。ラベルコントロールのTextプロパティを空文字列に設定します。



以下に上記デザインのHTMLソースのbodyタグ内を示します。

```
<body>
    <form id="form1" runat="server">
        <p>
            <asp:Button ID="Button1" runat="server" Height="75px" Text="実行" Width="210px"
OnClick="Button1_Click" />
        </p>
        <p>
             </p>
        <hr />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
    <hr />
</body>
```

6. Webフォーム上の[実行]ボタンをダブルクリックしてカーソルが移動した場所に、以下のサンプルコードを入力します。サンプルコード中のconstr変数に設定する接続文字列と、sqlstr変数に設定するSQL文は実行環境に合わせて変更してください。

```
Dim con As Microsoft.Data.Odbc.OdbcConnection
Dim com As Microsoft.Data.Odbc.OdbcCommand
Dim drd As Microsoft.Data.Odbc.OdbcDataReader
```

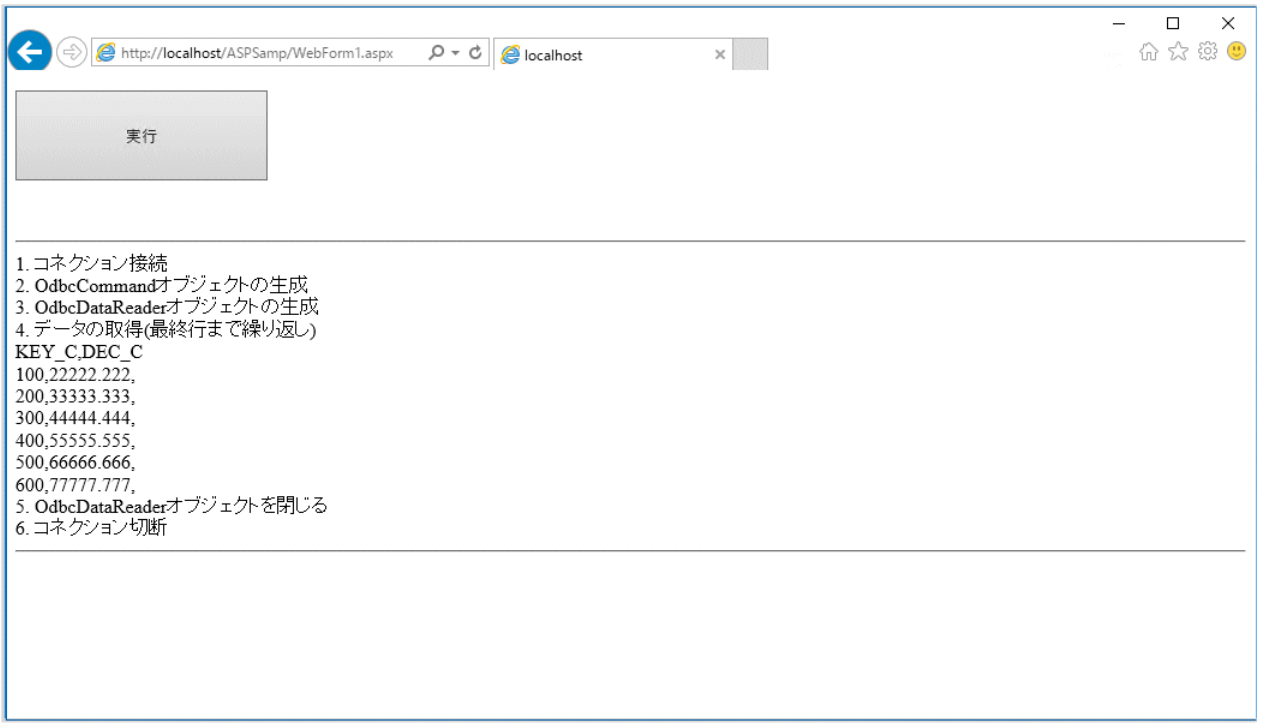
```
Dim i As Integer
Dim msgstr As String
Dim constr As String = "DSN=DSN01;UID=USER01;PWD=PASS01"
Dim sqlstr As String = "SELECT KEY_C,DEC_C FROM SAMPLE1.TESTTBL"

Try
    Label1.Text = "1. コネクション接続<br>"
    con = New Microsoft.Data.Odbc.OdbcConnection(constr)
    con.Open()

    Label1.Text &= "2. OdbcCommandオブジェクトの生成<br>"
    com = New Microsoft.Data.Odbc.OdbcCommand(sqlstr, con)

    Label1.Text &= "3. OdbcDataReaderオブジェクトの生成<br>"
    drd = com.ExecuteReader

    Label1.Text &= "4. データの取得(最終行まで繰り返し)<br>"
    Label1.Text &= "KEY_C,DEC_C<br>"
    While drd.Read()
        For i = 0 To drd.FieldCount - 1
            Label1.Text &= drd.GetValue(i) & ","
        Next
        Label1.Text &= "<br>"
    End While

    Label1.Text &= "5. OdbcDataReaderオブジェクトを閉じる<br>"
    drd.Close()

    Label1.Text &= "6. コネクション切断<br>"
    con.Close()

    con.Dispose()
    com.Dispose()

'Error処理
Catch ex As Microsoft.Data.Odbc.OdbcException
    For i = 0 To ex.Errors.Count - 1
        msgstr &= ex.Errors(i).Message & "<br>"
        msgstr &= "SQLSTATE: " & ex.Errors(i).SQLState & "<br>"
    Next
    Label1.Text = "ODBC Error Message<br>" & msgstr
Catch ex As Exception
    msgstr = ex.Message
    Label1.Text = "Error Message<br>" & msgstr
End Try
```

7. Visual Studioのメニューの、[デバッグ]－[開始]で実行します。

   また、メニューの[ビルド]－[TestWebAplのビルド]でWebアプリケーションをビルドします。プロジェクト内のTestWebApl¥Scripts¥WebForm1.aspx をコピーして、“環境設定”で作成したエイリアス(例:ASPSamp)配下に置きます。ブラウザのアドレスに以下のURLを指定してください。

```
http://Webサーバ名/ASPSamp/TestWebApl/WebForm1.aspx
```

[実行]ボタンをクリックすると、Symfoware/RDBのデータベースに接続し、結果を表示します。



開発用のサーバと実行用のサーバが違う場合には、以下の方法で作成したプロジェクトを実行用のサーバに配置します。

1. Visual Studioのメニューから[プロジェクト]-[プロジェクトのコピー]を選択します。
2. [プロジェクトのコピー]ダイアログボックスの[ターゲット プロジェクトフォルダ]に、実行用のWebサーバ名とプロジェクトフォルダ名(任意 例:TestWebApl)を次の例のように入力します。

   例

   ```
   http:// Webサーバ名/TestWebApl
   ```

3. [コピー]の以下のいずれかを選択します。
   - このアプリケーションの実行に必要なファイルのみ
   - すべてのプロジェクトファイル
   - ソースプロジェクトフォルダのすべてのファイル
4. [OK]ボタンをクリックしてコピーを開始します。

4.3.5 アプリケーションのデバッグ

IISとの連携時に発生するエラーと対処方法を説明します。

エラーが発生した場合のデバッグ情報として利用してください。

Symfoware/RDBおよびODOSのエラー(JYPxxxx)が発生した場合

Symfoware/RDBおよびODOSのエラーを取得する処理を必ず入れてください。

エラー情報の取得方法は、“付録C Webアプリケーションのサンプル”を参照してください。

参照

エラーメッセージについては、“メッセージ集”を参照して、対処をしてください。

ODOSのトレースの利用方法

IISではADOのAPIを利用してSymfoware/RDBのデータベースにアクセスしているため、Symfoware/RDBに発行しているSQL文が分かりません。

そのような場合、“トラブルシューティング集”を参照し、ODOSのトレースを採取すれば、ODBCのAPIの流れや、どのようなSQL文をSymfoware/RDBに発行しているかを確認できます。

参照

ODOSのトレースの採取方法については、“トラブルシューティング集”を参照してください。

また、データベース簡単運用の場合には、“トラブルシューティング集(データベース簡単運用編)”を参照してください。

4.3.6 利用時の注意事項

IISを利用してSymfoware/RDBのデータベースにアクセスする場合の注意事項を以下に示します。

VBアプリケーションで“カーソル位置が不当”というエラーが発生した場合

トランザクションを終了(コミットまたはロールバック)すると、カーソルはクローズされる仕組みとなっています。そのため、トランザクションの終了を実施した場合、必ずOpenRecordSetから実施してください。

“JYP9605E 無効な文字データが指定されました”というエラーが発生した場合

Parameterオブジェクトで数値型を指定したが、数値以外の文字を設定した場合に発生します。

以下のようなソースを記述した時に発生します。

例

```
Set OBJdbParameter = OBJdbCommand.CreateParameter
OBJdbParameter.Name = "PARA"
OBJdbParameter.Type = adInteger
OBJdbParameter.Value = "400a"
```

4.4 プロシジャルーチンを利用する場合

通常のクライアント・サーバ間は、クライアント側にアプリケーション全体を置き、サーバにはデータベースの検索・更新処理だけをさせる形態になっています。この場合、クライアントとサーバ間を検索結果などの大量なデータが行き来して、実際のアプリケーションはクライアント側で制御するようになり、大規模なトランザクション開発に適していません。このような場合は、プロシジャルーチンを利用して、サーバ側でアプリケーションを実行することができます。

本節では、プロシジャルーチンを利用する場合について、以下の項目を説明します。

- プロシジャルーチンを利用するアプリケーションの作成の流れ
- プロシジャルーチンの実行
- プロシジャルーチンの処理結果
- プロシジャルーチン利用時のトランザクション

4.4.1 プロシジャルーチンを利用するアプリケーションの作成の流れ

プロシジャルーチンを利用するアプリケーションは、以下の手順で作成します。

1. プロシジャルーチンの定義
2. プロシジャルーチンの実行

参照

プロシジャルーチンの定義方法については、“アプリケーション開発ガイド(共通編)”を参照してください。

4.4.2 プロシジャルーチンの実行

サーバ上のスキーマに登録済のプロシジャルーチンを、クライアント側から呼び出して実行するには、SQL制御文のCALL文を使用します。プロシジャルーチンに引数を指定することで、外部からの入力情報によって処理制御を切り替えることができます。CALL文の指定方法を以下に示します。

プロシジャルーチン実行の指定方法

[RDOを利用したプロシジャルーチンの実行方法]

```
Set Qry = Con.CreateQuery("", "CALL 在庫管理.営業所別発注処理(?)")
                                    (1)          (2)
Qry.rdoParameters(0).Value = INDATA
                                 (3)
Qry.Execute
```

(1) スキーマ名

(2) ルーチン名

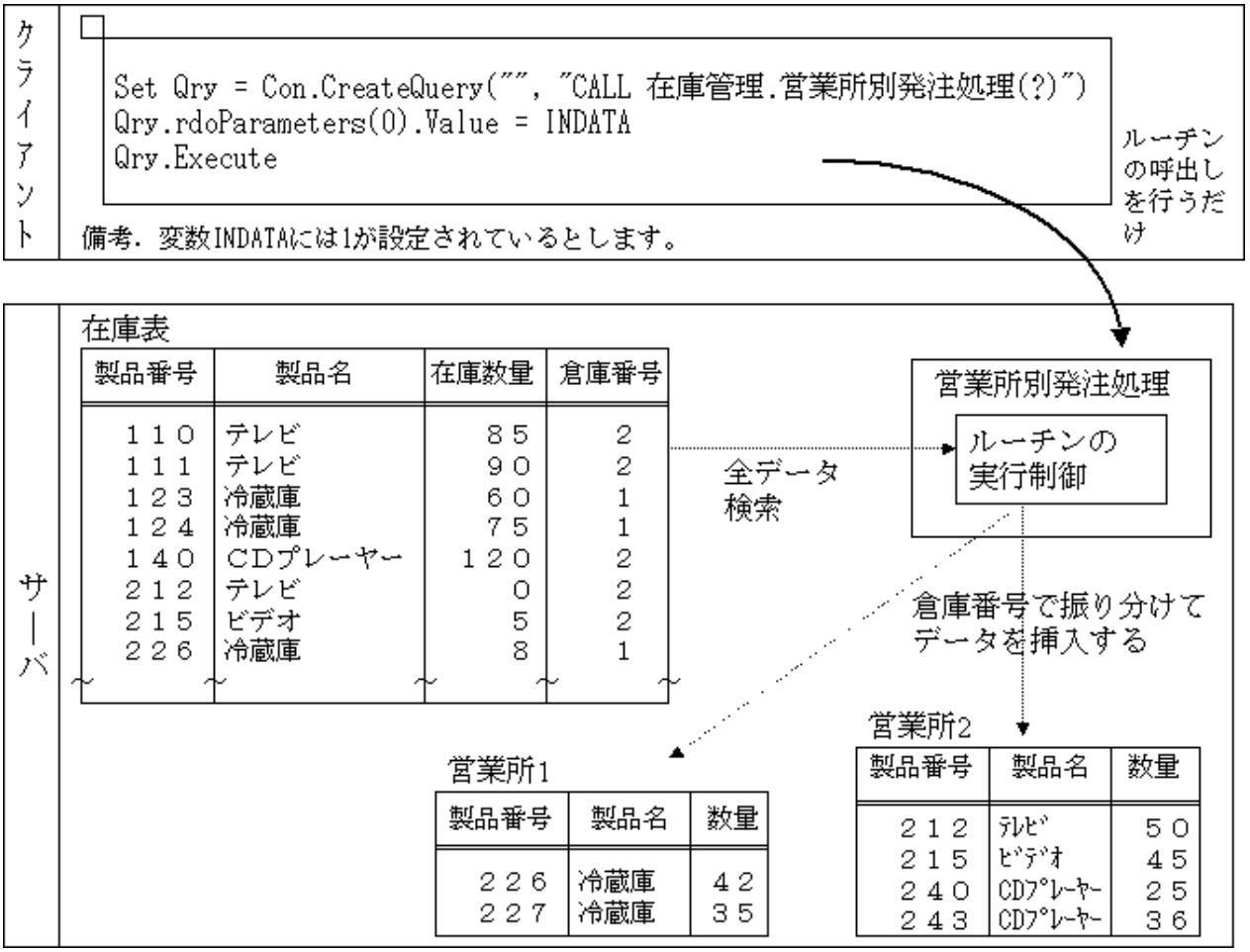
(3) 引数

RDOおよびRDO以外を利用したプロシジャルーチンの実行方法の詳細は、それぞれ以下を参照してください。

- RDO連携時:B.2.8 ストアドプロシジャの実行
- ADO連携時:B.3.8 ストアドプロシジャの実行
- ADO.NET:B.5.8 ストアドプロシジャの実行

“図4.1 ルーチン実行時のクライアントとサーバの関係”に、図4.3 プロシジャルーチン内の処理結果の通知方法に示したプロシジャルーチン定義を実行した場合のクライアントとサーバの関係を示します。

図4.1 ルーチン実行時のクライアントとサーバの関係

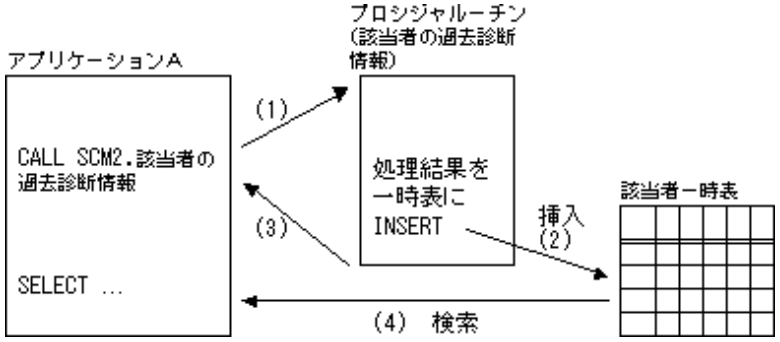


一時表を利用してプロシジャルーチンの結果を返却する方法

プロシジャルーチン内の処理で取り出したデータを呼出し側のアプリケーションに返却するには、パラメタ変数を利用します。しかし、表から抽出した大量のデータを返却する場合、パラメタ変数では実現できません。このような場合、一時表を利用することで実現できます。

以下に、概要を示します。

図4.2 一時表を利用したプロシジャルーチンとのデータの受渡し



(1) アプリケーションからプロシジャルーチン(該当者の過去診断情報)を呼び出します。

(2) プロシジャルーチンで抽出されたデータを、一時表(該当者一時表)に格納します。

(3) プロシジャルーチンが終了します。

(4) 一時表に格納されている、プロシジャルーチン内での抽出結果を、アプリケーションに取り込みます。

プロシジャルーチン内で抽出した結果を一時表に格納する例

```
CREATE PROCEDURE SCM2.該当者の過去診断情報(
  IN  Ｐ年齢範囲１  SMALLINT,
  IN  Ｐ年齢範囲２  SMALLINT,
  IN  Ｐ性別        CHAR(4),
  IN  Ｐ血液型      CHAR(5)
)
BEGIN
  DECLARE SQLSTATE      CHAR(5);
  DECLARE SQLMSG        CHAR(256);
  DECLARE Ｓ管理番号    INTEGER;
  -- 入力された条件に該当する、過去の患者の管理番号を抽出する
  DECLARE CUR01 CURSOR FOR SELECT 管理番号 FROM SCM1.患者管理表
        WHERE 年齢 BETWEEN Ｐ年齢範囲１ AND Ｐ年齢範囲２
          AND 性別   = Ｐ性別
          AND 血液型 = Ｐ血液型;
  -- ハンドラ宣言
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK WORK;  -- 例外が発生した場合、ROLLBACKして終了する
    RESIGNAL;       -- 発生した例外事象をCALL文の結果として通知する
  END;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
  END;    -- データなし例外発生時は処理を継続する

  -- 最初に初期化
  DELETE FROM SCM00.該当者一時表;
  -- 該当者の管理番号の抽出
  OPEN CUR01;
LOOP1:LOOP
    FETCH CUR01 INTO Ｓ管理番号;
    IF (SQLSTATE = '02000') THEN
        LEAVE LOOP1;
    END IF;
    -- 該当者の過去の診察情報を取り出して一時表に格納する
    INSERT INTO SCM00.該当者一時表
        SELECT 患者名, 診察日, 担当医, 診断結果, 診断詳細, 血圧
             FROM SCM1.診察結果管理
             WHERE 管理番号 = Ｓ管理番号;
  END LOOP LOOP1;
  CLOSE CUR01;
  COMMIT WORK;
END
```

備考.一時表“該当者一時表”はON COMMIT PRESERVE ROWS指定とします。

一時表に格納されている、プロシジャルーチン内での抽出結果をアプリケーションで取り出す例(RDO)

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Qry As rdoQuery
Dim Rst As rdoResultset

Dim msgstr As String
Dim i As Integer


Set Env = rdoEngine.rdoEnvironments(0)

'コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
```

```
                                False, "UID=USER01;PWD=PASS01;")

'プロシジャルーチンの呼び出し
Set Qry = Con.CreateQuery("", "CALL SCM2.該当者の過去診断情報(?,?,?,?)")

Qry.rdoParameters(0).Value = 20
Qry.rdoParameters(1).Value = 40
Qry.rdoParameters(2).Value = "男性"
Qry.rdoParameters(3).Value = "A RH+"

Qry.Execute

'一時表のデータを出力する
Set Rst = Con.OpenResultset( _
           "SELECT * FROM SCM00.該当者一時表 ORDER BY 患者名, 診察日", _
           rdOpenForwardOnly, _
           rdConcurReadOnly, _
           rdExecDirect)
i = 1
Do Until Rst.EOF
    msgstr = ""
    msgstr = "[" & i & "]"
    msgstr = msgstr & "患者名=" & Rst.rdoColumns(0).Value
    msgstr = msgstr & ", 診察日=" & Rst.rdoColumns(1).Value
    msgstr = msgstr & ", 担当医=" & Rst.rdoColumns(2).Value
    msgstr = msgstr & ", 診断結果=" & Rst.rdoColumns(3).Value
    msgstr = msgstr & ", 診断詳細=" & Rst.rdoColumns(4).Value
    msgstr = msgstr & ", 血圧=" & Rst.rdoColumns(5).Value
    MsgBox msgstr, vbOKOnly, "Resultset"
    Rst.MoveNext
    i = i + 1
Loop
Rst.Close

Con.Close

Set Rst = Nothing
Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing
```

4.4.3 プロシジャルーチンの処理結果

ここでは、プロシジャルーチン内での処理結果の確認方法と、プロシジャルーチンの呼出し元のアプリケーションでの処理結果の確認方法について説明します。

プロシジャルーチン内での処理結果の確認

プロシジャルーチン内のSQL手続き文の実行中に、例外コード40の例外(ロールバック例外)が発生した場合は、トランザクションをロールバックし、呼出し元のアプリケーションに無条件に復帰します。このときの例外事象は呼出し元のSQLSTATEおよびSQLMSGに通知されます。

プロシジャルーチン内のSQL手続き文の実行中に、例外コード40以外の例外が発生した場合は、処理結果はプロシジャルーチン内のSQLSTATEとSQLMSGに通知されます。利用者は、プロシジャルーチン内にSQLSTATEおよびSQLMSGをSQL変数として宣言しておき、その内容を参照することによって例外事象を知ることができます。

プロシジャルーチン内ではハンドラを使用することができます。ハンドラとは、プロシジャルーチン実行中に例外が発生した際に実行されるサブルーチンです。ハンドラ宣言はどのような例外が発生した場合にどのような動作を行うかを指定します。プロシジャルーチン実行中に例外コード40以外の例外が発生した際、該当するハンドラが呼び出され、特定の処理を行い、プロシジャルーチンの処理を回復することができます。ハンドラ宣言が1つ以上指定されている状態で、かつ、該当するハンドラが存在しなかった場合は、ハンドラで処理されなかったとして、発生した例外のSQLSTATEおよびSQLMSGが呼び出し元に返却されます。

注意

ハンドラ宣言が1つも指定されていない状態で、プロシジャルーチン実行中に例外コード40以外の例外が発生した場合は、発生した例外のSQLSTATEおよびSQLMSGは呼び出し元に返却されません。この場合は、プロシジャルーチン内のSQLSTATEとSQLMSGに通知された内容を参照することにより、例外事象を確認します。

また、ハンドラ動作内で新たに例外が発生した場合、ハンドラ動作は終了して、新たに発生した例外のSQLSTATEおよびSQLMSGが呼び出し元に返却されます。

プロシジャルーチンの呼出し元での処理結果の確認

CALL文を実行して、プロシジャルーチンに記述された一連の処理が中断なく実行された場合、呼出し元のSQLSTATEおよびSQLMSGには正常終了が通知されます。例外によってプロシジャルーチンが実行されない、またはプロシジャルーチン内で例外が発生して処理が中断された場合は、その例外事象が呼出し元のSQLSTATEおよびSQLMSGに通知されます。プロシジャルーチンが実行されない場合とは、アクセス規則違反や、CALL文引数とプロシジャルーチンのパラメタ間の代入エラーが発生した場合などがあります。プロシジャルーチンの処理が中断される場合とは、プロシジャルーチン内のSQL手続き文で例外コード40の例外(ロールバック例外)が発生した場合、発生した例外がハンドラで処理されない場合、またはハンドラ動作中に新たに例外が発生した場合です。

プロシジャルーチンの呼出し元での処理結果の確認(RDO)および図4.3 プロシジャルーチン内の処理結果の通知方法に、具体例を示します。

プロシジャルーチンの呼出し元での処理結果の確認(RDO)

```
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Qry As rdoQuery

Dim msgstr As String

Set Env = rdoEngine.rdoEnvironments(0)

'(1)エラー処理ルーチンの設定
On Error GoTo ErrorProc

'コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'(2)プロシジャルーチンの呼び出し
Set Qry = Con.CreateQuery("", "CALL スキーマＯＯ.PCALL000(?,?,?)")
Qry.rdoParameters(0).Value = 1
Qry.rdoParameters(1).Value = 1
Qry.rdoParameters(2).Value = "顧客１"
Qry.Execute

Con.Close

Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'(3) エラー処理
ErrorProc:
Err_Count = rdoEngine.rdoErrors.Count
If Err_Count > 0 Then
    msgstr = "プロシジャルーチン実行で異常発生" & Chr(10)
    For i = 0 To Err_Count - 1
        msgstr = msgstr & "    エラーコード     ===> " & _
```

```
                rdoEngine.rdoErrors(i).SQLState & Chr(10)
        msgstr = msgstr & "    エラーメッセージ ===> " & _
                rdoEngine.rdoErrors(i).Description & Chr(10)
    Next
    rdoEngine.rdoErrors.Clear
End If
MsgBox msgstr, vbOKOnly, "Error"

Con.Close

Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing
```

(1) エラー処理のルーチンをErrorProcに設定します。

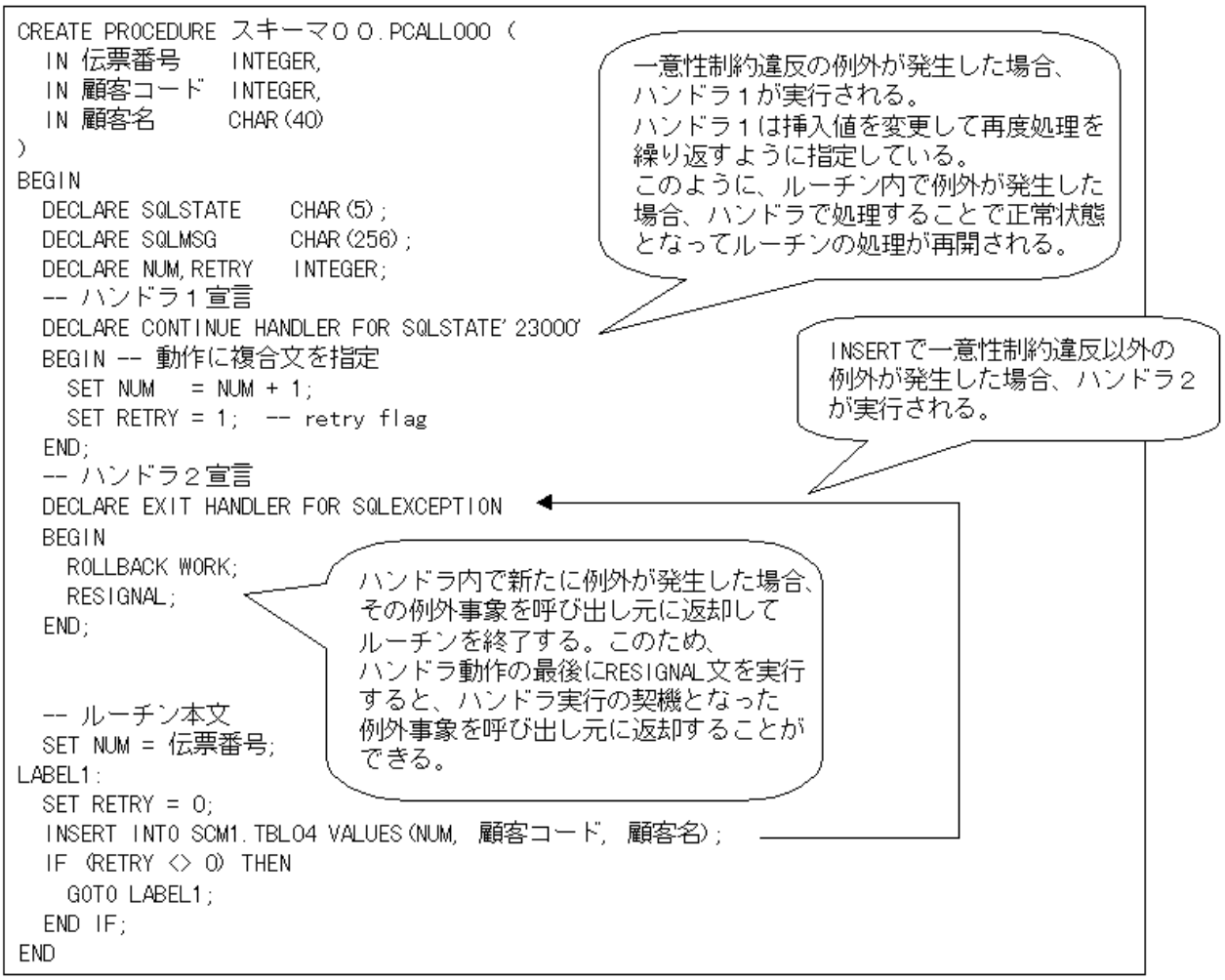
(2) プロシジャルーチンPCALL000を呼び出します。

(3) CALL文の結果がエラーだった場合、SQLSTATEおよびSQLMSGを表示して終了します。

RDOおよびRDO以外を利用したエラー処理の詳細は、それぞれ以下を参照してください。

- RDO連携時:B.2.11 エラー処理
- ADO連携時:B.3.11 エラー処理
- ADO.NET連携時:B.5.10 エラー処理

図4.3 プロシジャルーチン内の処理結果の通知方法



4.4.4 プロシジャルーチン利用時のトランザクション

ここでは、プロシジャルーチン利用時のトランザクションについて説明します。トランザクション制御の詳細については、“2.2 トランザクションと排他制御”を参照してください。

プロシジャルーチンは、呼出し元のアプリケーションと同一トランザクションとして動作します。アプリケーションから呼び出されたプロシジャルーチン内でトランザクション管理文(COMMIT文またはROLLBACK文)が実行された場合、プロシジャルーチンが実行されているトランザクションを終了します。その後、プロシジャルーチン内でトランザクションを開始するSQL文が実行された場合、プロシジャルーチン内で新しいトランザクションが開始されます。(トランザクションモードは前トランザクションと同一です。)開始されたトランザクションは、呼出し元のアプリケーションにも継続されます。

プロシジャルーチン内のSQL文の実行でエラーが起きた場合は、エラーを起こしたSQL文が無効となります。ただし、SQLSTATE値の例外コードが40の場合は、トランザクションをロールバックします。

プロシジャルーチンがSQLSTATE値の例外コードが40以外の状態で終了する場合、プロシジャルーチン内で行われたすべてのデータベース更新は無効となります。

プロシジャルーチン内でのSQL文の実行時には、ODOSデータソースに指定したCLI_TRAN_SPECの指定は無効ですが、アプリケーションに復帰した後は、CLI_TRAN_SPECの指定に従って制御されます。

4.5 アプリケーションのチューニング

Symfoware/RDBのクライアント用の動作環境ファイルに相当する指定を、Symfoware ODOSセットアップの[Symfoware/RDBオプション設定]画面で指定することができます。なお、オプションで設定できるパラメタ名の先頭文字“CLI_”を削除したパラメタ名が、Symfoware Serverのクライアント用の動作環境ファイルのパラメタと一致しています。

データソースに指定したオプション値はコネクション接続時に読み込まれます。

Symfoware ODOS セットアップの詳細については、“3.1 ODBCデータソースの登録”を参照してください。

動作環境パラメタの指定の優先順位

動作環境の設定項目の中には、システム用の動作環境ファイル、サーバ用の動作環境ファイルおよびODOSのオプションで設定できるパラメタで、重複して指定できるパラメタがあります。

参考

データベース簡単運用の場合には、Symfoware Severのインストール時に、システム用の動作環境として最適な環境が設定されるため、チューニングの必要がありません。よって、システム用の動作環境ファイルは、作成不要です。

優先順位は、以下のとおりです。

1. サーバ用の動作環境ファイル
2. ODOSのオプションで設定できるパラメタ
3. システム用の動作環境ファイル

【パラメタ】

ODOSを利用するアプリケーションから指定可能なパラメタは以下となります。

表4.1 ODOSのオプションで設定できるパラメタの種類

分類	パラメタ	概要	優先順位		
			1	2	3
			SV	ODOSのオプションで設定できるパラメタ	SY
通信	CLI_BUFFER_SIZE	通信に使用するバッファサイズ	―	○	―
	CLI_SERVER_ENV_FILE	使用するサーバ用の動作環境ファイル	―	○	―
	CLI_TRAN_SPEC	SQLエラー発生時のトランザクション	―	○	―
	CLI_TRAN_TIME_LIMIT	1つのトランザクションの最大使用可能時間	○	○	○
	CLI_WAIT_TIME	通信時の待ち時間	―	○	―
通信データの暗号化	CLI_SSL_CLI_CA_CERT_FILE	サーバ認証で使用するCA証明書ファイルの配置先	―	○ (注)	―
作業領域など	CLI_DESCRIPTOR_SPEC	SQL記述子の情報	―	○	―
	CLI_MAX_SQL	同時に操作できるSQL文の数	―	○	―
	CLI_OPL_BUFFER_SIZE	SQL文の実行手順を格納しておくバッファのサイズ	―	○	―
	CLI_RESULT_BUFFER	一括FETCHを行う場合のバッファの数とサイズ	―	○	―
	CLI_SORT_MEM_SIZE	作業用ソート領域として使うメモリサイズ	○	○	○
	CLI_WORK_ALLOC_SPACESIZE	作業用ソート領域および作業用テーブルとして使うファイルサイズ	○	○	―
	CLI_WORK_MEM_SIZE	作業用テーブルとして使うメモリのサイズ	○	○	○
	CLI_WORK_PATH	作業用テーブルおよび作業用ソート領域のパス	○	○	○
データ処理	CLI_CAL_ERROR	代入処理でオーバフローが起きた場合の処理	―	○	―
	CLI_CHARACTER_TRANSLATE	文字コードの変換をクライアントで行うか否か	―	○	―
	CLI_SURROGATE_PAIR_NUMBER	UNICODEの補助文字(1～16面の4バイト文字)の文字数	○	○	○
表・インデックス	CLI_DEFAULT_INDEX_SIZE	格納構造定義を行わないインデックスを作成する場合のインデックスのデータ格納域の初期量、拡張量、ページ長など	○ (注)	○ (注)	○
	CLI_DEFAULT_OBJECT_TABLE_SIZE	格納構造定義を行わない表を作成する場合のOBJECT構造の表のデータ格納域の初期量、拡張量、ページ長など	○ (注)	○ (注)	○
	CLI_DEFAULT_TABLE_SIZE	格納構造定義を行わない表を作成する場合の表のデータ格納域の初期量、拡張量、ページ長など	○ (注)	○ (注)	○

分類	パラメタ	概要	優先順位		
			1	2	3
			SV	ODOSのオプションで設定できるパラメタ	SY
	CLI_DSI_EXPAND_POINT	DSIの容量拡張を起動するか否か	―	○ (注)	―
	CLI_INCLUSION_DSI	アプリケーションで使用するDSIを限定する	○ (注)	○ (注)	―
	CLI_TEMPORARY_INDEX_SIZE	一時表にインデックスを定義する場合のインデックスのデータ格納域の初期量、拡張量など	○ (注)	○ (注)	○
	CLI_TEMPORARY_TABLE_SIZE	一時表を定義する場合の表のデータ格納域の初期量、拡張量など	○ (注)	○ (注)	○
排他	CLI_DSO_LOCK	使用するDSOの占有の単位、占有モード	○ (注)	○ (注)	―
	CLI_ISOLATION_WAIT	占有待ちの方式	―	○	―
	CLI_R_LOCK	占有の単位を行とするか否か	―	○	○
トランザクション	CLI_DEFAULT_ACCESS_MODE	トランザクションアクセスモードの初期値を指定する	―	○	○
	CLI_DEFAULT_ISOLATION	独立性水準の初期値を指定する	―	○	○
デバッグ	CLI_ROUTINE_SNAP	ROUTINE_SNAP機能を利用するか否か	―	○	―
	CLI_SQL_SNAP	SQL_SNAP機能を利用するか否か	―	○	―
アクセスプランおよび性能情報	CLI_ACCESS_PLAN	アプリケーション単位でアクセスプランを取得するか否かおよびSQL文に対するアドバイスを出力するか否か	―	○	―
	CLI_CHOOSE_TID_UNION	WHERE句にブール演算子”OR”、または、行値構成子を指定した場合、TIDユニオンマージのアクセスプランのみを作成するか否か	○	○	○
	CLI_GROUP_COL_COND_MOVE	導出表を絞り込む探索条件を指定した場合、その探索条件を導出表のWHERE句に移動するか否か	○	○	○
	CLI_IGNORE_INDEX	インデックスを使用しないアクセスプランを選択するか否か	○	○	―
	CLI_INACTIVE_INDEX_SCAN	非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択するか否か	○	○	○
	CLI_JOIN_ORDER	結合表と他の表のジョイン順	○	○	○
	CLI_JOIN_RULE	ジョインする方法	○	○	○
	CLI_MAX_SCAN_RANGE	インデックス、クラスタキー、または分割キーの検索範囲の最大数	○	○	○

分類	パラメタ	概要	優先順位		
			1	2	3
			SV	ODOSのオプションで設定できるパラメタ	SY
	CLI_SAME_COST_JOIN_ORDER	最適化情報を設定していない場合、ジョイン順をV5以前と同じにするか否か	○	○	○
	CLI_SCAN_KEY_ARITHMETIC_RANGE	四則演算の検索範囲について、インデックスの範囲検索、または、クラスタキー検索を行うか否か	○	○	○
	CLI_SCAN_KEY_CAST	探索条件のCASTオペランドに指定した列でインデックスの範囲検索、または、クラスタキー検索を行うか否か	○	○	○
	CLI_SORT_HASHAREA_SIZE	ソート処理がレコードをハッシングして格納するための領域サイズ	○	○	○
	CLI_SQL_TRACE	アプリケーション単位でSQL性能情報を取得するか否か	－	○	－
	CLI_SS_RATE	述語ごとの検索範囲の選択率の値	○	○	○
	CLI_TID_SORT	インデックス検索と表データ取得のアクセスモデルでTIDソートを利用するか否か	○	○	○
	CLI_TID_UNION	TIDユニオンマージのアクセスモデルを有効にするか否か	○	○	○
	CLI_USQL_LOCK	UPDATE文:探索またはDELETE文:探索の更新標的レコードを位置づける部分の占有モード	○	○	○
リカバリ	CLI_RCV_MODE	ドライバのリカバリ水準を指定する	－	○ (注)	－
予約語とSQL機能	CLI_SQL_LEVEL	アプリケーションの予約語とSQL機能のレベルを設定する	－	○	－
並列クエリ	CLI_MAX_PARALLEL	データベースを並列に検索する場合の多重度	○ (注)	○ (注)	○
	CLI_PARALLEL_SCAN	アプリケーション単位またはコネクション単位にデータベースを並列に検索するか否か	○ (注)	○ (注)	
その他	CLI_ARC_FULL	アーカイブログ満杯時にエラー復帰するか否か	－	○ (注)	○

SV:サーバ用の動作環境ファイルへの指定が可能であるか否かを表します。

SY:システム用の動作環境ファイルへの指定が可能であるか否かを表します。

○:指定可

－:指定不可

注)データベース簡単運用の場合は、指定できません。

◆通信に関する実行パラメタ

CLI_BUFFER_SIZE

【指定形式】

CLI_BUFFER_SIZE = ([初期量][,拡張量])

【実行パラメタの意味】

通信に利用するバッファサイズを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_BUFFER_SIZE = (32,32)

【パラメタの意味】

初期量:

バッファの初期量を1～10240の範囲で指定します。省略した場合は、32が指定されたものとみなします。単位はキロバイトです。

拡張量:

拡張量を1～10240の範囲で指定します。省略した場合は、32が指定されたものとみなします。単位はキロバイトです。

CLI_SERVER_ENV_FILE

【指定形式】

CLI_SERVER_ENV_FILE = (SQLサーバ名,ファイル名)

【実行パラメタの意味】

使用するサーバ用の動作環境ファイル名を指定します。コネクション(データベース環境)ごとにサーバのアプリケーション実行環境を変更する場合に指定します。

【パラメタの意味】

SQLサーバ名:

接続するデータ資源名を記述します。SQLサーバ名に、“)”、“;”、“=”および“%”は指定できません。

ファイル名:

サーバ用の動作環境ファイル名を、絶対パスで指定します。ファイル名に、“)”、“;”、“=”および“%”は指定できません。

CLI_TRAN_SPEC

【指定形式】

CLI_TRAN_SPEC = ({NONE | TRANSACTION_ROLLBACK})

【実行パラメタの意味】

SQL文が実行中にエラーとなった場合のトランザクションの対処方法を指定します。

なお、本実行パラメタは、トランザクションモニタ配下では指定できません。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_TRAN_SPEC = (NONE)

【パラメタの意味】

NONE:

各プラットフォームのトランザクションの仕様に従います。

TRANSACTION_ROLLBACK:

SQL文の実行がエラーとなった場合に、トランザクションをロールバックします。

CLI_TRAN_TIME_LIMIT

【指定形式】

CLI_TRAN_TIME_LIMIT = (最大トランザクション実行時間)

【実行パラメタの意味】

1つのトランザクションで使用可能な時間を指定します。

指定時間を超過した場合には、トランザクションをロールバックして、接続中のコネクションを切断します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_TRAN_TIME_LIMIT = (0)

【パラメタの意味】

最大トランザクション実行時間:

最大トランザクション実行を0～32767の範囲で指定します。単位は秒です。0を指定すると無制限になります。

CLI_WAIT_TIME

【指定形式】

CLI_WAIT_TIME = (待ち時間)

【実行パラメタの意味】

サーバからのデータ受信の待ち時間を指定します。

CLI_WAIT_TIMEで指定された時間内に、サーバからのデータが受信できなかった場合には、実行中のSQL文はエラーとなり、コネクションは切断されます。

なお、本実行パラメタは、トランザクションモニタ配下では指定できません。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_WAIT_TIME = (0)

【パラメタの意味】

待ち時間:

待ち時間を0～32767の範囲で指定します。単位は秒です。0を指定した場合、データが受信できるまで待ちます。

◆通信データの暗号化に関する実行パラメタ

CLI_SSL_CLI_CA_CERT_FILE

【指定形式】

CLI_SSL_CLI_CA_CERT_FILE = (CA証明書ファイル名)

【実行パラメタの意味】

サーバ認証で使用する認証局(Certificate Authority: CA)証明書ファイルの配置先を指定します。

CA証明書ファイルは、データベース管理者が認証局に発行手続きを行って取得後、アプリケーションを開発および実行するマシンに配布されます。

通信データを暗号化しない場合に本パラメタを設定するとエラーになります。

【パラメタの意味】

CA証明書ファイル名:

CA証明書のファイル名を絶対パスで指定します。

指定できるファイル形式は、PEM形式のみです。

また、証明書の鍵長は2048ビット以上を推奨します。なお、1024ビット未満の鍵長で生成された証明書は使用できません。

指定例:

```
CLI_SSL_CLI_CA_CERT_FILE = (C:¥CertificateAuthority¥CAFlie.pem)
```

◆作業領域に関する実行パラメタ

CLI_DESCRIPTOR_SPEC

【指定形式】

CLI_DESCRIPTOR_SPEC = (項目記述子域の数の初期値)

【実行パラメタの意味】

動的SQLのSQL記述子の情報を指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_DESCRIPTOR_SPEC = (100)

【パラメタの意味】

項目記述子域の数の初期値:

動的SQLで利用するSQL記述子域獲得時の項目記述子域の数の初期値を1～32767の範囲で指定します。

CLI_MAX_SQL

【指定形式】

CLI_MAX_SQL = (オブジェクトの数)

【実行パラメタの意味】

同一コネクションオブジェクト内で同時に作成できる、以下のオブジェクトの数を指定します。

- RDOの場合:rdoQueryオブジェクト、rdoResultsetオブジェクト
- ADOの場合:Commandオブジェクト、Recordsetオブジェクト
- ADO.NETの場合:OdbcCommandオブジェクト、OdbcDataAdapterオブジェクト

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_MAX_SQL = (1024)

【パラメタの意味】

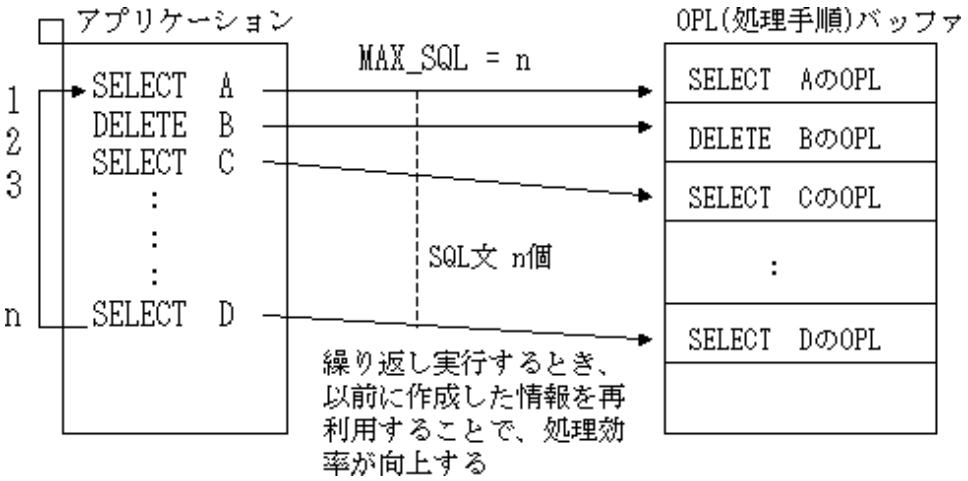
オブジェクトの数:

オブジェクトの数を2～32000の範囲で指定します。

ポイント

CLI_MAX_SQLの値を拡張して使用した場合、以下の考慮が必要です。

Symfoware/RDBでは、同一SQL文を繰り返し実行する場合の処理効率を図るため、SQL文を実行するための情報の個数をCLI_MAX_SQLに指定します。また、SQL文の処理手順を格納するOPLバッファのサイズを、CLI_OPL_BUFFER_SIZEに指定します。



SQL文情報の保持期間は、以下のようになります。

- RDOの場合:rdoQueryオブジェクト、rdoResultsetオブジェクトが作成されてから、破棄されるまで
- ADOの場合:Commandオブジェクト、Recordsetオブジェクトが作成されてから、破棄されるまで
- ADO.NETの場合:OdbcCommandオブジェクト、OdbcDataAdapterオブジェクトが作成されてから、破棄されるまで

CLI_OPL_BUFFER_SIZE

【指定形式】

CLI_OPL_BUFFER_SIZE = (バッファサイズ)

【実行パラメタの意味】

Symfoware/RDBでは、同一SQL文を複数回実行するときに、最初の実行で作成した処理手順を使用することによって処理効率の向上を図っています。CLI_MAX_SQLに指定した数の処理手順を格納するバッファのサイズを指定します。この領域は、サーバ側で獲得されます。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_OPL_BUFFER_SIZE = (8192)

【パラメタの意味】

バッファサイズ:

SQLの処理手順を格納するバッファのサイズを1～10240の範囲で指定します。単位はキロバイトです。

注意

同一SQL文を複数回実行するとき、以下の場合については最初の実行で作成した処理手順は使用されず、新たに処理手順を作成します。

- 前回の実行後、保持可能な処理手順の数(CLI_MAX_SQLに指定した値と同数)以上の異なるSQL文を実行した場合
- 前回の実行後、SET TRANSACTION文によりトランザクションモード(アクセスモードまたは独立性水準)を変更した場合
- 前回の実行後、SET CATALOG文により被準備文の対象となるデータベース名を変更した場合
- 前回の実行後、SET SCHEMA文により被準備文の省略したスキーマ名を変更した場合
- 前回の実行後、SET SESSION AUTHORIZATION文によりスコープの異なる利用者に変更した場合
- 前回の実行後、SQL文が使用するデータベース資源についてALTER TABLE文により動的に列の追加または削除を実行した場合

ー 前回の実行後、SQL文が使用するデータベース資源についてCREATE DSI文により動的にDSIの追加または削除を実行した場合

ー 前回の実行後、SQL文が使用するデータベース資源についてALTER DSI文により動的にDSIの分割値変更を実行した場合

ー 動作環境ファイルのパラメタINACTIVE_INDEX_SCANにNOを指定している場合、またはODOSのオプションで設定できるパラメタCLI_INACTIVE_INDEX_SCANにNOを指定している場合に、前回の実行後、rdbexdsiコマンドにより任意のDSIの除外または除外の解除を実行した場合

ー 前回の実行後、SQL文が使用するデータベース資源を削除した場合

ポイント

CLI_MAX_SQLの値を拡張した場合は、CLI_OPL_BUFFER_SIZEの値も変更する必要があります。以下の方法で見積もった値をキロバイト単位で指定してください。

```
SQLの処理手順を格納するバッファのサイズ
= 0.7KB + Σ アクセス対象の表単位のSQL処理手順サイズ
アクセス対象の表単位のSQL処理手順サイズ
= 4.2KB ＋ Σ SQL文単位のSQL処理手順サイズ
SQL文単位のSQL処理手順サイズ
= 0.17KB × 列数 ＋ 0.08KB × 条件数
```

ー アクセス対象の表単位のSQL処理手順サイズ

アクセス対象の表単位のSQL処理手順サイズの総和です。アクセス対象の表単位のSQL処理手順サイズを求め、それらを合計します。

ー SQL文単位のSQL処理手順サイズ

当該表をアクセスするSQL文単位のSQL処理手順サイズの総和です。当該表をアクセスするSQL文単位のSQL処理手順サイズを求め、それらを合計します。

ー 列数

当該SQL文に記述する列の数です。列に“*”を記述する場合は、表を構成する列の数になります。同一の列を選択リストや探索条件などの複数箇所に記述する場合や同一箇所に同一の列を複数記述する場合は、それぞれ列数に加算してください。

ー 条件数

当該SQL文に記述する条件(述語)の数です。

CLI_RESULT_BUFFER

【指定形式】

CLI_RESULT_BUFFER = ([個数][,バッファサイズ])

【実行パラメタの意味】

ODOSは、データを取り出すときの性能を良くするため、複数の行を一度に取り出します。この行を格納するバッファの数とサイズを指定します。また、1つのカーソルが1つのバッファを使用するので、複数のバッファを用意すれば、複数カーソルの操作の性能を良くすることができます。バッファを使用しない場合、個数に0を指定します。この領域は、クライアント側とサーバ側で獲得されます。

バッファサイズを大きくするほど性能は良くなりますが、メモリが圧迫され、他のアプリケーションの実行に支障が発生する場合があります。

バッファサイズを大きくする場合、メモリの空き容量に注意してください。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_RESULT_BUFFER = (2,32)

【パラメタの意味】

個数:

使用するバッファの個数を0～255の範囲で指定します。省略した場合は、2が指定されたものとみなします。

バッファサイズ:

使用するバッファのサイズを1～10240の範囲で指定します。省略した場合は、32が指定されたものとみなします。単位はキロバイトです。

CLI_SORT_MEM_SIZE

【指定形式】

CLI_SORT_MEM_SIZE = (メモリサイズ)

【実行パラメタの意味】

ソート処理のために作業用ソート領域としてサーバ側で使用するメモリの大きさを指定します。この領域は、RDBプロセスのローカルメモリにセション単位に獲得されます。

ソート処理のデータ量がCLI_SORT_MEM_SIZEに指定した値を超えると、二次記憶の作業用ソート領域にデータを書き出し、書き出したデータのソートを行います。このとき、二次記憶からのソートデータの読み込み回数はソートデータの全体量とCLI_SORT_MEM_SIZEに指定した値に依存します。このため、ソートデータの全体量に応じて、CLI_SORT_MEM_SIZEに指定する値を見積もってください。

ソート処理のデータ量がCLI_SORT_MEM_SIZEに指定した値の1万倍以上になると、ソート処理で必要なメモリが不足し、「JYP2221E 実行時の制限値を超えました. code:“4”」のエラーとなる場合があります。ただし、メモリ上の作業域の必要最低限なサイズは、作業用ソート領域へのレコードの格納順に依存するため、CLI_SORT_MEM_SIZEに指定した値の1万倍は目安となります。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_SORT_MEM_SIZE = (2112)

【パラメタの意味】

メモリサイズ:

サーバ側で使用するメモリの大きさを64～2097150の範囲で指定します。単位はキロバイトです。

CLI_WORK_ALLOC_SPACESIZE

【指定形式】

CLI_WORK_ALLOC_SPACESIZE = ([初期量][,[増分量][,[最大量][,[保持指定]]]])

【実行パラメタの意味】

作業用ソート領域および作業用テーブルとしてサーバ側で使用するファイルサイズの初期量、増分量、最大量、保持指定を指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_WORK_ALLOC_SPACESIZE = (10000,50000,CLI_WORK_PATHで指定したパス名のディスク容量,HOLD)

初期量、増分量、最大量、保持指定のいずれかの値が省略された場合は、その値のデフォルト値が指定されたものとみなします。

以下に指定例を示します。

例1:初期量、増分量を指定する場合

CLI_WORK_ALLOC_SPACESIZE = (10000,50000)

例2:増分量、最大量を指定する場合

CLI_WORK_ALLOC_SPACESIZE = (,50000,100000)

【パラメタの意味】

初期量:

作業用ソート領域および作業用テーブルとして外部ファイルを作成する場合の初期量を5000～50000の範囲で指定します。省略した場合は10000が指定されたものとみなします。単位はキロバイトです。

増分量:

作業用ソート領域および作業用テーブルとして作成した外部ファイルを拡張する場合の増分量を1000～100000の範囲で指定します。省略した場合は、50000が指定されたものとみなします。単位はキロバイトです。

最大量:

作業用ソート領域および作業用テーブルとして作成する外部ファイルの最大量を5000～33553408の範囲で指定します。省略した場合は、CLI_WORK_PATHで指定したパス名のディスク容量となります。単位はキロバイトです。

保持指定:

以下の中から1つを選択します。省略した場合は、HOLDが指定されたものとみなします。

－ FREE:作業用ソート領域および作業用テーブルとして作成した外部ファイルを初期量まで解放します。

－ HOLD:作業用ソート領域および作業用テーブルとして作成した外部ファイルを保持します。

CLI_WORK_MEM_SIZE

【指定形式】

CLI_WORK_MEM_SIZE = (メモリサイズ)

【実行パラメタの意味】

作業用テーブルとしてサーバ側で使用するメモリの大きさです。この領域は、RDBプロセスのローカルメモリにセション単位に獲得されます。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_WORK_MEM_SIZE = (128)

【パラメタの意味】

メモリサイズ:

サーバ側で使用するメモリの大きさを64～2097150の範囲で指定します。単位はキロバイトです。

CLI_WORK_PATH

【指定形式】

CLI_WORK_PATH = (パス名[,パス名]･･･)

【実行パラメタの意味】

サーバ側で使用するソート作業域、作業用テーブル域の獲得先ディレクトリを指定します。“Symfoware/RDBを起動するユーザID”および“RDBコマンドを実行するユーザID”には、指定するディレクトリへの書込み権が必要です。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

S L

CLI_WORK_PATH = (/var/tmp)

W

CLI_WORK_PATH = (Symfoware/RDBがインストールされているディレクトリ¥TMP)

データベース簡単運用で、いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合は、以下の値が設定されます。

CLI WORK_PATH = (データ格納先ディレクトリ¥RDBシステム名¥USR¥TMP)

参照

- 作業用ソート領域および作業用テーブルの見積りについては、“アプリケーション開発ガイド(共通編)”の“ソート作業域の見積り”を参照してください。
- データベース簡単運用の場合の省略値として設定されるデータ格納先ディレクトリについては、“データベース簡単運用ガイド”を参照してください。

【パラメタの意味】

パス名:

獲得先ディレクトリを指定します。

◆データ処理に関する実行パラメタ

CLI_CAL_ERROR

【指定形式】

CLI_CAL_ERROR = ({REJECT | NULL})

【実行パラメタの意味】

代入処理でオーバフローが発生した場合の処理を指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_CAL_ERROR = (REJECT)

【パラメタの意味】

REJECT:

例外エラーとします。

NULL:

演算結果をNULLとします。

CLI_CHARACTER_TRANSLATE

【指定形式】

CLI_CHARACTER_TRANSLATE = ({CLIENT | SERVER})

【実行パラメタの意味】

データベースシステムの文字コード系が、アプリケーションで使用している文字コード系と異なる場合、コード変換をクライアントで行うか、サーバで行うかを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_CHARACTER_TRANSLATE = (SERVER)

【パラメタの意味】

CLIENT:

クライアントでコード変換を行う場合に指定します。

SERVER:

サーバでコード変換を行う場合に指定します。

ポイント

サーバの負荷を少しでも減らしたい場合は、クライアントで行うよう指定します。

CLI_SURROGATE_PAIR_NUMBER

【指定形式】

CLI_SURROGATE_PAIR_NUMBER = ({1 | 2})

【実行パラメタの意味】

各国語文字を扱う関数は、データベースの文字コード系に従って処理されます。データベースの文字コード系がUNICODEの場合、各国語文字列を扱う関数はUNICODEの補助文字(1～16面の4バイト文字)をUCS-2形式の2文字(2バイト×2)として認識しています。本実行パラメタを指定することにより、従来2文字として認識していた補助文字を1文字として認識します。補助文字を1文字として扱うと、補助文字の文字数を意識しないで以下の関数を使うことができます。

- POSITION、CHARINDEX
- CHARACTER_LENGTH、LEN
- SUBSTRING、LEFT、RIGHT
- TRIM
- LPAD
- RPAD
- REPLACE
- REPLICATE
- REVERSE
- STUFF

本実行パラメタは、サーバおよびクライアントがWindows(R)の場合に利用できます。

本実行パラメタは、データベースの文字コード系がUNICODEの場合にのみ有効となります。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_SURROGATE_PAIR_NUMBER = (2)

【パラメタの意味】

1:

各国語文字の補助文字1文字(4バイト)を1文字として扱います。

2:

従来どおり、各国語文字の補助文字1文字(4バイト)を2文字(2バイト×2)として扱います。

◆表・インデックスに関する実行パラメタ

CLI_DEFAULT_INDEX_SIZE

【指定形式】

CLI_DEFAULT_INDEX_SIZE = (ベース部ページ長,インデックス部ページ長,ベース部初期量,インデックス部初期量[,拡張量,拡張契機])

【実行パラメタの意味】

格納構造定義を行わないインデックスを作成する場合、インデックスのベース部とインデックス部の割り付け量、ページ長などを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_DEFAULT_INDEX_SIZE = (8,8,30720,10240,10240,3072)

【パラメタの意味】

ベース部ページ長:

ベース部のページ長を1、2、4、8、16、32の中から指定します。単位はキロバイトです。

インデックス部ページ長:

インデックス部のページ長を1、2、4、8、16、32の中から指定します。単位はキロバイトです。

ベース部初期量:

ベース部の初期量を2～2097150の範囲で指定します。単位はキロバイトです。

インデックス部初期量:

インデックス部の初期量を2～2097150の範囲で指定します。単位はキロバイトです。

拡張量:

インデックスのベース部の拡張量を1～2097150の範囲で指定します。省略した場合は、10240が指定されたものとみなします。単位はキロバイトです。インデックス部の拡張量は、ベース部の5分の1の値となります。

拡張契機:

ベース部およびインデックス部の拡張を行うタイミングとして、DSIの空き容量を0～2097150の範囲で指定します。インデックスのDSIの空き容量がここで指定した値になると、インデックスのベース部およびインデックス部の拡張が行われます。省略した場合は、3072が指定されたものとみなします。単位はキロバイトです。

注意

- 自動容量拡張の拡張量と拡張契機は、ページ長単位に繰り上げますので、ページ長の倍数で指定してください。
- ベース部の5分の1がインデックス部のページ長の倍数でない場合、インデックス部のページ長の倍数に繰り上げます。
- インデックス定義時には、容量拡張を行いません。拡張量および拡張契機は、インデックス定義した後に有効となります。

CLI_DEFAULT_OBJECT_TABLE_SIZE

【指定形式】

CLI_DEFAULT_OBJECT_TABLE_SIZE = (ページ長,初期量[,拡張量,拡張契機])

【実行パラメタの意味】

格納構造定義を行わない表を作成する場合、OBJECT構造の表の、データ格納域の割り付け量、ページ長などを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_DEFAULT_OBJECT_TABLE_SIZE = (32,32768,32768,0)

【パラメタの意味】

ページ長:

データ格納域のページ長を指定します。必ず32を指定します。単位はキロバイトです。

初期量:

データ格納域の初期量を2～2097150の範囲で指定します。単位はキロバイトです。

拡張量:

データ格納域の拡張量を1～2097150の範囲で指定します。省略した場合は、32768が指定されたものとみなします。単位はキロバイトです。

拡張契機:

データ格納域の拡張を行うタイミングとして、表のDSIの空き容量を0～2097150の範囲で指定します。表のDSIの空き容量がここで指定した値になると、データ格納域の拡張が行われます。省略した場合は、0が指定されたものとみなします。単位はキロバイトです。

自動容量拡張の拡張量と拡張契機は、ページ長単位に繰り上げますので、ページ長の倍数で指定してください。

CLI_DEFAULT_TABLE_SIZE

【指定形式】

CLI_DEFAULT_TABLE_SIZE = (ページ長,初期量[,拡張量,拡張契機])

【実行パラメタの意味】

格納構造定義を行わない表を作成する場合、表のデータ格納域の割り付け量、ページ長などを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_DEFAULT_TABLE_SIZE = (32,30720,10240,3072)

【パラメタの意味】

ページ長:

データ格納域のページ長を1、2、4、8、16、32の中から指定します。単位はキロバイトです。

初期量:

データ格納域の初期量を2～2097150の範囲で指定します。単位はキロバイトです。

拡張量:

データ格納域の拡張量を1～2097150の範囲で指定します。省略した場合は、10240が指定されたものとみなします。単位はキロバイトです。

拡張契機:

データ格納域の拡張を行うタイミングとして、表のDSIの空き容量を0～2097150の範囲で指定します。表のDSIの空き容量がここで指定した値になると、表のデータ格納域の拡張が行われます。省略した場合は、3072が指定されたものとみなします。単位はキロバイトです。

自動容量拡張の拡張量と拡張契機は、ページ長単位に繰り上げますので、ページ長の倍数で指定してください。

CLI_DSI_EXPAND_POINT

【指定形式】

CLI_DSI_EXPAND_POINT={ON | OFF})

【実行パラメタの意味】

アプリケーションによるデータ操作で、DSIに指定された拡張契機(rdbalmdsiコマンドまたはDSI定義文で定義します)を無効とするか否かを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_DSI_EXPAND_POINT=(ON)

【パラメタの意味】

ON:

DSIに定義された拡張契機は有効になります。アプリケーションによるデータ操作で、DSIの空きページ容量が拡張契機に達した時点で、領域を拡張します。

OFF:

DSIに定義された拡張契機は無効になります。アプリケーションによるデータ操作で、DSIの空きページ容量が拡張契機に達しても、領域を拡張しません。この場合、DSIの空き領域が枯渇した時点で、領域を拡張します。

CLI_INCLUSION_DSI

【指定形式】

CLI_INCLUSION_DSI = (データベース名.DSI名[,データベース名.DSI名・・・])

【実行パラメタの意味】

アプリケーションで、DSIを限定したい表のDSI名を指定します。

アプリケーションでは、限定されたDSIを含む表に対しては、そのDSIだけがデータ操作の範囲となります。また、本実行パラメタの指定により、アプリケーション中での探索条件の記述が省略ができます。なお、アプリケーションでDSIを限定していない表に対しては、データ操作をすることができます。

【パラメタの意味】

データベース名.DSI名:

DSIを限定したい表のDSI名を指定します。DSI名に、“(”、“:”、“=”、“*”、“+”、“-”、“?”、“%”および“'”は指定できません。

CLI_TEMPORARY_INDEX_SIZE

【指定形式】

CLI_TEMPORARY_INDEX_SIZE = (ベース部初期量,インデックス部初期量[,拡張量,拡張契機])

【実行パラメタの意味】

一時表にインデックスを定義する場合に、インデックスのベース部とインデックス部の割付け量を指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_TEMPORARY_INDEX_SIZE = (160,64,256,0)

【パラメタの意味】

ベース部初期量:

ベース部の初期量を64～2097150の範囲で指定します。単位はキロバイトです。

インデックス部初期量:

インデックス部の初期量を64～2097150の範囲で指定します。単位はキロバイトです。

拡張量:

インデックスのベース部の拡張量を32～2097150の範囲で指定します。省略した場合は、256が指定されたものとみなします。単位はキロバイトです。インデックス部の拡張量は、ベース部の5分の1の値となります。

拡張契機:

ベース部およびインデックス部の拡張を行うタイミングとして、インデックスの空き容量を0～2097150の範囲で指定します。インデックスの空き容量がここで指定した値になると、インデックスのベース部およびインデックス部の拡張が行われます。省略した場合は、0が指定されたものとみなします。単位はキロバイトです。

注意

ー 自動容量拡張の拡張量と拡張契機は、ページ長単位に繰り上げますので、ページ長の倍数で指定してください。

－ ベース部の5分の1がインデックス部のページ長の倍数でない場合、インデックス部のページ長の倍数に繰り上げます。

CLI_TEMPORARY_TABLE_SIZE

【指定形式】

CLI_TEMPORARY_TABLE_SIZE = (初期量[,拡張量,拡張契機])

【実行パラメタの意味】

一時表を定義する場合に、表のデータ格納域の割付け量を指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_TEMPORARY_TABLE_SIZE = (256,512,0)

【パラメタの意味】

初期量:

データ格納域の初期量を64～2097150の範囲で指定します。単位はキロバイトです。

拡張量:

データ格納域の拡張量を32～2097150の範囲で指定します。省略した場合は、512が指定されたものとみなします。単位はキロバイトです。

拡張契機:

データ格納域の拡張を行うタイミングとして、表の空き容量を0～2097150の範囲で指定します。表の空き容量がここで指定した値になると、表のデータ格納域の拡張が行われます。省略した場合は、0が指定されたものとみなします。単位はキロバイトです。

注意

自動容量拡張の拡張量と拡張契機は、ページ長単位に繰り上げますので、ページ長の倍数で指定してください。

◆排他に関する実行パラメタ

CLI_DSO_LOCK

【指定形式】

CLI_DSO_LOCK = (DSO名[/[P][占有モード]][,DSO名[/[P][占有モード]]･･･])

【実行パラメタの意味】

アプリケーションで使用するDSOおよびその占有の単位、占有モードを指定します。

本パラメタを指定した場合、指定されなかったDSOについては、占有の単位がページとなります。

CLI_DSO_LOCKが指定された場合、SET TRANSACTION文、CLI_DEFAULT_ACCESS_MODEおよびCLI_DEFAULT_ISOLATIONは指定できません。

占有の単位は、CLI_DSO_LOCK、および、CLI_R_LOCKまたは動作環境ファイルのR_LOCKでの指定により決定します。

なお、CLI_R_LOCKおよび動作環境ファイルのR_LOCKの値により、CLI_DSO_LOCKを指定できない場合があります。以下にCLI_DSO_LOCKと、CLI_R_LOCKおよびR_LOCKの関係を示します。

ODOSのオプション値のCLI_R_LOCK	システム用の動作環境ファイルのR_LOCK	CLI_DSO_LOCKの指定	占有の単位
NO	NO	指定可	CLI_DSO_LOCKの指定による(注1)
	YES	指定可	CLI_DSO_LOCKの指定による(注1)
YES	NO	指定不可	行単位で占有(注2)
	YES	指定不可	行単位で占有(注2)
省略	NO	指定可	CLI_DSO_LOCKの指定による(注1)
	YES	指定不可	行単位で占有(注2)

注1) CLI_DSO_LOCKの指定を省略した場合、Symfoware/RDBによって占有の単位が選択されます。詳細は、“アプリケーション開発ガイド(共通編)”の“排他の属性と選択方法”の“Symfoware/RDBによる選択”を参照してください。

注2) CLI_DSO_LOCKを指定した場合、データベースへの接続時にエラーとなります。

【パラメタの意味】

DSO名:

アプリケーションで使用するDSO名を以下の形式で指定します。

データベース名.DSO名

P:

DSOの占有の単位をページとします。省略した場合、占有の単位はDSIになります。

DSO名に指定されたDSOにPRECEDENCE(1)が指定されている場合、本パラメタは指定できません。

占有モード:

占有のモードとして以下のいずれかを指定します。省略した場合は、EXが指定されたものとみなします。

EX:

非共有モードの排他を行います。

SH:

共有モードの排他を行います。

CLI_ISOLATION_WAIT

【指定形式】

CLI_ISOLATION_WAIT = ({WAIT | REJECT})

【実行パラメタの意味】

あるトランザクションで資源にアクセスしようとしたとき、別のトランザクションがその資源を占有していた場合に、資源の占有が解除されるまで待つかどうかを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_ISOLATION_WAIT = (WAIT)

【パラメタの意味】

WAIT:

資源の占有が解除されるまで待ちます。

REJECT:

エラーとしてアプリケーションに復帰します。

CLI_R_LOCK

【指定形式】

CLI_R_LOCK = ({YES | NO})

【実行パラメタの意味】

占有の単位を行とするかどうかを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_R_LOCK = (YES)

【パラメタの意味】

YES:

占有の単位を行とします。このパラメタを指定した場合、ODOSのオプション値のCLI_DSO_LOCKまたは、サーバ用の動作環境ファイルにDSO_LOCKパラメタを指定することはできません。

NO:

占有の単位は、CLI_DSO_LOCKまたはDSO_LOCKの指定に従います。このパラメタを指定し、かつCLI_DSO_LOCKおよびDSO_LOCKが指定されていない場合は、Symfoware/RDBによって自動的に占有の単位が選択されます。

データベース簡単運用の場合、占有の単位は、表またはインデックスとなります。

参照

詳細については、“アプリケーション開発ガイド(共通編)”の“排他制御”を参照してください。

また、データベース簡単運用の場合は、“アプリケーション開発ガイド(共通編)”の“トランザクションと排他制御(データベース簡単運用の場合)”を参照してください。

注意

- システム用の動作環境ファイルのR_LOCKまたはODOSのオプション値のCLI_R_LOCKがNOの場合、CLI_DEFAULT_ISOLATIONまたはSET TRANSACTION文にREPEATABLE_READを指定しても、独立性水準はSERIALIZABLEになります。
- システム用の動作環境ファイルのR_LOCKまたはODOSのオプション値のCLI_R_LOCKがYESの場合、CLI_DEFAULT_ISOLATIONまたはSET TRANSACTION文にSERIALIZABLEを指定しても、独立性水準はREPEATABLE READになります。
- データベース簡単運用でない場合、システム用の動作環境ファイルのR_LOCKまたはODOSのオプション値のCLI_R_LOCKがNOのときには、DSO定義でPRECEDENCE(1)が指定されたSEQUENTIAL構造の表にアクセスするアプリケーションの占有の単位はDSIになります。

◆トランザクションに関する実行パラメタ

CLI_DEFAULT_ACCESS_MODE

【指定形式】

CLI_DEFAULT_ACCESS_MODE = ({READ_ONLY | READ_WRITE})

【実行パラメタの意味】

トランザクションアクセスモードの初期値を指定します。CLI_DEFAULT_ACCESS_MODEが指定された場合、プロセスで最初に実行されるSQL文の直前でSET TRANSACTION文が実行されたことになります。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_DEFAULT_ACCESS_MODE = (READ_WRITE)

【パラメタの意味】

READ_ONLY:

トランザクションアクセスモードの初期値をREAD ONLYとします。

READ_WRITE:

トランザクションアクセスモードの初期値をREAD WRITEとします。

注意

サーバ用の動作環境ファイルのDSO_LOCKまたはODOSのオプション値のCLI_DSO_LOCKを指定して、CLI_DEFAULT_ACCESS_MODEを指定した場合は、サーバ接続時にエラーが発生します。

CLI_DEFAULT_ISOLATION

【指定形式】

CLI_DEFAULT_ISOLATION = ([独立性水準1][,独立性水準2])

独立性水準1および独立性水準2に指定可能な値は以下のとおりです。

- DEFAULT
- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

【実行パラメタの意味】

独立性水準の初期値を指定します。CLI_DEFAULT_ISOLATIONが指定された場合、プロセスで最初に実行されるSQL文の直前でSET TRANSACTION文が実行されたことになります。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_DEFAULT_ISOLATION = (DEFAULT,DEFAULT)

データベース簡単運用で、本実行パラメタの指定を省略した場合は、省略値として以下の値が設定されます。

DEFAULT_ISOLATION = (READ COMMITTED, READ COMMITTED)

【パラメタの意味】

独立性水準1:

トランザクションアクセスモードがREAD ONLYの時の独立性水準の値を指定します。

独立性水準2:

トランザクションアクセスモードがREAD WRITEの時の独立性水準の値を指定します。

独立性水準1および独立性水準2に指定する値の意味は以下のとおりです。

DEFAULT:

独立性水準はシステム用の動作環境ファイルにおけるDEFAULT_ISOLATIONの指定に従い設定されます。

READ_UNCOMMITTED:

独立性水準の初期値をREAD UNCOMMITTEDとします。

READ_COMMITTED:

独立性水準の初期値をREAD COMMITTEDとします。

REPEATABLE_READ:

独立性水準の初期値をREPEATABLE READとします。

SERIALIZABLE:

独立性水準の初期値をSERIALIZABLEとします。

注意

- CLI_DEFAULT_ACCESS_MODE に READ_ONLY を指定した場合、独立性水準はCLI_DEFAULT_ISOLATIONの指定に関係なくREAD UNCOMMITTEDとなります。
- システム用の動作環境ファイルのR_LOCKまたはODOSのオプション値のCLI_R_LOCKがNOの場合、CLI_DEFAULT_ISOLATIONまたはSET TRANSACTION文にREPEATABLE_READを指定しても、独立性水準はSERIALIZABLEとなります。
- システム用の動作環境ファイルのR_LOCKまたはODOSのオプション値のCLI_R_LOCKがYESの場合、CLI_DEFAULT_ISOLATIONまたはSET TRANSACTION文にSERIALIZABLEを指定しても、独立性水準はREPEATABLE READとなります。
- サーバ用の動作環境ファイルのDSO_LOCKまたはODOSのオプション値のCLI_DSO_LOCKを指定して、CLI_DEFAULT_ISOLATIONを指定した場合は、サーバ接続時にエラーが発生します。

◆デバッグに関する実行パラメタ

CLI_ROUTINE_SNAP

【指定形式】

CLI_ROUTINE_SNAP = ({ON | OFF},ファイル名[,出力レベル])

【実行パラメタの意味】

ROUTINE_SNAP機能を利用するかどうかを指定します。

ROUTINE_SNAP機能は、SQL手続き文の実行情報をファイルに出力する機能です。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_ROUTINE_SNAP = (OFF)

参照

ROUTINE_SNAP機能の詳細および使用方法については、“アプリケーション開発ガイド(埋込みSQL編)”の“アプリケーションのデバッグ”を参照してください。

【パラメタの意味】

ON:

ROUTINE_SNAP機能を利用する場合に指定します。

OFF:

ROUTINE_SNAP機能を利用しない場合に指定します。

ファイル名:

SQL手続き文の実行情報の出力先のサーバ側のファイル名を、絶対パスで指定します。ファイル名に、“,”、“)”、“:”、“;”、“=”、“/”、“>”および“%”は指定できません。

指定されたファイルがすでに存在する場合、情報を追加して出力します。

複数のアプリケーションが動作する場合は、個別のトレース情報を出力しません。

アプリケーションがマルチスレッド環境で動作する場合は、出力ファイル名の後にプロセスIDやセションIDなどの情報を自動的に付加して、個別のトレース情報を出力します。

出力レベル:

出力する情報のレベルとして、1または2を指定します。省略した場合、2が指定されたものとみなします。

参照

出力レベルの指定と出力情報の対応については、“アプリケーション開発ガイド(埋込みSQL編)”の“ROUTINE_SNAP機能の利用方法”を参照してください。

CLI_SQL_SNAP

【指定形式】

CLI_SQL_SNAP=(出力モード[,[ファイル名][,[出力レベル][,[繰り返し幅][,[プロセス指定]]]]])

【実行パラメタの意味】

SQL_SNAP機能を利用するかどうかを指定します。

SQL_SNAP機能は、アプリケーションが実行したSQL文の情報をファイルに出力する機能です。

SQL_SNAP機能は情報をファイルに出力するため、使用するとドライバの性能が悪くなります。必要なときにだけ指定してください。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_SQL_SNAP=(OFF)

注意

本機能の利用は性能に影響を与えるため、デバッグ作業が終了したら、本実行パラメタの設定をOFFにしてください。

また、格納データを暗号化している場合でも、スナップファイル内の情報は暗号化されません。そのため、デバッグ作業が終了したら、本実行パラメタの設定をOFFにしてください。

【パラメタの意味】

出力モード:

SQL_SNAP機能を利用するかどうかを指定します。

ON:

SQL_SNAP機能を利用する場合に指定します。

OFF:

SQL_SNAP機能を利用しない場合に指定します。

ファイル名:

出力するスナップファイルのファイル名を指定します。ファイル名にはパスと拡張子名を指定できます。ファイル名が省略された場合、sqlexec.snpというファイル名になります。拡張子が省略された場合、“.snp”という拡張子名になります。

パスが省略された場合、実行するアプリケーションのカレントディレクトリに出力します。指定されたファイルがすでに存在する場合は、情報を追加して出力します。

出力レベル:

スナップファイルの接続レベルでの出力単位を指定します。省略した場合、CONが指定されものたとみなします。

DSN:

データソース単位でスナップファイルを出力します。

CON:

コネクション単位でスナップファイルを出力します。

このパラメタが指定された場合、コネクション単位でファイル名を区別するために、指定したファイル名の後ろにコネクションを識別する10進の数値が付加されます。

例:コネクションが2つ存在する場合

```
指定したファイル名
    sqlexec.snp
実際のファイル名
    sqlexec_2.snp
    sqlexec_5.snp
```

繰り返し幅:

スナップファイルの出力方法を1～32767で指定します。

省略した場合、0になります。

0:

接続から切断するまでの間に実行したAPIのスナップ情報をすべて出力します。

0以外:

指定された値だけAPIのスナップ情報を出力します。

プロセス指定:

スナップファイルのシステムレベルでの出力単位を指定します。

省略した場合、SYSになります。

SYS:

システム単位でスナップファイルを出力します。

PRCS:

プロセス単位でスナップファイルを出力します。

このパラメタが指定された場合、プロセス単位でファイル名を区別するために、指定したファイル名の後ろにプロセスを識別する16進の数値が付加されます。

例:プロセスが2つ存在する場合

```
指定したファイル名
    sqlexec.snp
実際のファイル名
    sqlexec_2435.snp
    sqlexec_5654.snp
```

◆アクセスプランおよび性能情報に関する実行パラメタ

CLI_ACCESS_PLAN

【指定形式】

CLI_ACCESS_PLAN = ({ON | OFF},ファイル名[,[出力レベル][,[SQLアドバイザ出力レベル]]])

【実行パラメタの意味】

アプリケーション単位でアクセスプランを取得するかどうかを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_ACCESS_PLAN = (OFF)

参照

アクセスプランについては、“アプリケーション開発ガイド(共通編)”の“アクセスプラン”を参照してください。

【パラメタの意味】

ON:

アクセスプラン取得機能を利用する場合に指定します。

OFF:

アクセスプラン取得機能を利用しない場合に指定します。

ファイル名:

出力先のサーバ側のファイル名を、絶対パスで指定します。指定されたファイルがすでに存在する場合は、情報を追加して出力します。また、指定するパスが存在すること、および、指定したユーザIDに対する書込み権があることを確認してください。

出力レベル:

出力レベルには1または2を指定します。1を指定すると、アクセスプランのセクション情報のみを出力し、2を指定すると、セクション内の各エレメント詳細情報も出力します。省略した場合は、2が指定されたものとみなします。

SQLアドバイザ出力レベル:

SQLアドバイザ出力レベルには、“ADVICE”または“NOADVICE”を指定します。“ADVICE”を指定すると、SQL文に対するアドバイスを出力します。“NOADVICE”を指定すると、SQL文に対するアドバイスを出力しません。省略した場合は、“ADVICE”が指定されたものとみなします。

CLI_CHOOSE_TID_UNION

【指定形式】

CLI_CHOOSE_TID_UNION = ({YES | NO})

【実行パラメタの意味】

SQL文のWHERE句にブール演算子“OR”を指定した場合、V10.1.0以前は、TIDユニオンマージのアクセスパスのみ作成されましたが、V11.0.0以降では、TIDユニオンマージに加えて、インデックス検索、または、インデックス検索と表データ取得のアクセスプランが作成されます。ほとんどの場合、インデックス検索の方がSQL文のレスポンスが良いのですが、データの分布によっては表に対するI/O効率の高いTIDユニオンマージの方が良い場合もあります。どちらのアクセスプランを選択するかは、Symfoware/RDBにより自動的に決定されますが、最適化情報を設定しない運用の場合はこの選択を誤ってしまい、V11.0.0以降、まれにSQL文のレスポンスが悪くなることがあります。このような場合は、当パラメタにYESを指定して、V10.1.0以前と同様のアクセスプランとしてください。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_CHOOSE_TID_UNION = (NO)

【パラメタの意味】

YES:

以下のいずれかの場合、TIDユニオンマージのアクセスプランのみを作成します(V10.1.0以前のアクセスプラン)。

- WHERE句にブール演算子”OR”を指定している場合
- WHERE句に行値構成子を指定している場合

NO:

以下のいずれかの場合、TIDユニオンマージのアクセスプランとインデックス検索のアクセスプランを作成し、効率の良いアクセスプランをSymfoware/RDBが自動的に選択します。

- WHERE句にブール演算子”OR”を指定している場合
- WHERE句に行値構成子を指定している場合

CLI_GROUP_COL_COND_MOVE

【指定形式】

CLI_GROUP_COL_COND_MOVE = ({YES | NO})

【実行パラメタの意味】

V6以前のSymfoware Serverからバージョンアップした場合、最適化情報を設定しない運用では、まれにSQL文のレスポンスが悪くなることがあります。このような場合は、本パラメタにNOを指定して、V6以前のアクセスプランを選択します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_GROUP_COL_COND_MOVE = (YES)

【パラメタの意味】

YES:

以下のいずれかの場合、導出表を絞り込む条件を導出表内に移動させる(V7以降のアクセスプラン)。

- 導出表にGROUP BY句、HAVING句または集合関数を指定している場合。
- 導出表の選択リストに列指定以外を指定している場合。

NO:

以下のいずれかの場合、導出表を絞り込む条件を導出表内に移動させない(V6以前のアクセスプラン)。

- 導出表にGROUP BY句、HAVING句または集合関数を指定している場合。
- 導出表の選択リストに列指定以外を指定している場合。

CLI_IGNORE_INDEX

【指定形式】

CLI_IGNORE_INDEX = ({YES | NO})

【実行パラメタの意味】

データベースを検索する時に、インデックスを使用せずにデータベースにアクセスするか否かを指定します。データウェアハウジングにおいて大量のデータを検索および集計する場合には、インデックスだけではデータが絞りきれず、インデックスを使う分だけ無駄なオーバヘッドが発生することがあります。このような場合は、本パラメタにYESを指定することにより、表の全件検索または並列スキャンのアクセスモデルを採用します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_IGNORE_INDEX = (NO)

ただし、SQL文にASSIST指定のUSE_INDEXを指定した場合、ASSIST指定が優先され、CLI_IGNORE_INDEXの指定に関わらず、ASSIST指定に指定されたインデックスを使用した検索を行います。

【パラメタの意味】

YES:

インデックスを使用しないアクセスモデルを選択する。

NO:

インデックスが使用できる時は、インデックスを使用したアクセスモデルを選択する。

CLI_INACTIVE_INDEX_SCAN

【指定形式】

CLI_INACTIVE_INDEX_SCAN = ({YES | NO})

【実行パラメタの意味】

データベースを検索する時に、非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択するか否かを指定します。インデックスを非活性状態とし、一括更新処理でインデックスを更新しない高速なバッチ処理を行うアプリケーションの場合、本パラメタにNOを指定してください。YESが指定されている場合、バッチ処理がエラーとなる場合があります。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_INACTIVE_INDEX_SCAN = (YES)

【パラメタの意味】

YES:

非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択する。

NO:

非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択しない。

CLI_JOIN_ORDER

【指定形式】

CLI_JOIN_ORDER = ({AUTO | INSIDE | OUTSIDE})

【実行パラメタの意味】

結合表と他の表をジョインする場合のジョイン順を指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_JOIN_ORDER = (INSIDE)

データベース簡単運用で、いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合は、以下の値が設定されます。

CLI_JOIN_ORDER = (AUTO)

ただし、SQL文にASSIST指定のLEADING_TABLEを指定した場合、ASSIST指定が優先され、CLI_JOIN_ORDERの指定に関わらず、ASSIST指定に指定されたジョイン順に従います。

【パラメタの意味】

AUTO:

Symfoware/RDBが自動的に選択する。

INSIDE:

結合表から先にジョインする。

OUTSIDE:

結合表の中に指定した表と結合表の外に指定した表から先にジョインする。

CLI_JOIN_RULE

【指定形式】

CLI_JOIN_RULE = ({AUTO | MERGE | FETCH})

【実行パラメタの意味】

ジョインする方法を指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_JOIN_RULE = (AUTO)

ただし、SQL文にASSIST指定のJOIN_RULEを指定した場合、ASSIST指定が優先され、CLI_JOIN_RULEの指定に関わらず、ASSIST指定に指定されたジョイン方法に従います。

【パラメタの意味】

AUTO:

Symfoware/RDBが自動的に選択する。

MERGE:

マージジョインのアクセスモデルを優先する。

FETCH:

フェッチジョインのアクセスモデルを優先する。

CLI_MAX_SCAN_RANGE

【指定形式】

CLI_MAX_SCAN_RANGE = (検索範囲数)

【実行パラメタの意味】

インデックス、クラスタキー、または分割キーについて検索範囲を作成する場合の最大数を指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_MAX_SCAN_RANGE = (1000)

【パラメタの意味】

検索範囲数:

1～2147483647の範囲で指定します。

CLI_SAME_COST_JOIN_ORDER

【指定形式】

CLI_SAME_COST_JOIN_ORDER = ({REVERSE | ORDER})

【実行パラメタの意味】

CLI_JOIN_ORDERパラメタにINSIDEを指定するか、省略し、最適化情報を設定しない運用の場合、まれにSQL文のレスポンスが悪くなることがあります。このような場合、本パラメタにREVERSEを指定して、V5以前のジョイン順を選択します。

最適化処理は、最適化情報で処理手順のコストを見積もり、最適な処理手順を選びます。最適化情報を設定していない場合、どのジョイン順も同じコストになることがあります。この場合、V5以前は、WHERE句の表を結合する探索条件の記述の逆順でしたが、V6以降は、記述順になります。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_SAME_COST_JOIN_ORDER = (ORDER)

ただし、SQL文にASSIST指定のLEADING_TABLEを指定した場合、ASSIST指定が優先され、CLI_SAME_COST_JOIN_ORDERの指定に関わらず、ASSIST指定に指定されたジョイン順に従います。

【パラメタの意味】

REVERSE:

CLI_JOIN_ORDERパラメタにINSIDEを指定するか、省略し、最適化情報を設定しない運用の場合、WHERE句の表を結合する探索条件の記述の逆順にジョインするアクセスプランを選択する(V5以前のアクセスモデル)。

ORDER:

CLI_JOIN_ORDERパラメタにINSIDEを指定するか、省略し、最適化情報を設定しない運用の場合、WHERE句の表を結合する探索条件の記述順にジョインするアクセスプランを選択する(V6以降のアクセスモデル)。

CLI_SCAN_KEY_ARITHMETIC_RANGE

【指定形式】

CLI_SCAN_KEY_ARITHMETIC_RANGE = ({YES | NO})

【実行パラメタの意味】

四則演算の検索範囲について、インデックス範囲検索、または、クラスタキーの検索を行うか否かを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_SCAN_KEY_ARITHMETIC_RANGE = (YES)

ただし、SQL文にASSIST指定のUSE_INDEXを指定した場合、ASSIST指定が優先され、CLI_SCAN_KEY_ARITHMETIC_RANGEの指定に関わらず、ASSIST指定に指定されたインデックスを使用した検索を行います。

【パラメタの意味】

YES:

四則演算の検索値について、インデックス範囲検索、または、クラスタキーの検索を行います。

NO:

四則演算の検索値について、インデックス範囲検索、または、クラスタキーの検索を行いません。

CLI_SCAN_KEY_CAST

【指定形式】

CLI_SCAN_KEY_CAST = ({YES | NO})

【実行パラメタの意味】

探索条件のCASTオペランドに指定した列でインデックスの範囲検索、または、クラスタキー検索を行うか否かを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_SCAN_KEY_CAST = (YES)

ただし、SQL文にASSIST指定のUSE_INDEXを指定した場合、ASSIST指定が優先され、CLI_SCAN_KEY_CASTの指定に関わらず、ASSIST指定に指定されたインデックスを使用した検索を行います。

【パラメタの意味】

YES:

探索条件のCASTオペランドに指定した列でインデックスの範囲検索、または、クラスタキー検索を行います。

NO:

探索条件に指定したインデックスキーまたは、クラスタキーで範囲検索を行います。

CLI_SORT_HASHAREA_SIZE

【指定形式】

CLI_SORT_HASHAREA_SIZE = (メモリサイズ)

【実行パラメタの意味】

ソート処理がレコードをハッシングして格納するための領域のサイズです。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、メモリ上のすべてのレコードをハッシングして格納します。

【パラメタの意味】

メモリサイズ:

2112～2097150の範囲で指定します。単位はキロバイトです。

CLI_SQL_TRACE

【指定形式】

CLI_SQL_TRACE = ({ON | OFF},性能情報ファイル名[,出力レベル])

【実行パラメタの意味】

アプリケーション単位でSQL性能情報を取得するかどうかを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_SQL_TRACE = (OFF)

【パラメタの意味】

ON:

SQL性能情報取得機能を利用する場合に指定します。

OFF:

SQL性能情報取得機能を利用しない場合に指定します。

性能情報ファイル名:

出力先のサーバ側のファイル名を、絶対パスで指定します。指定されたファイルがすでに存在する場合は、情報を追加して出力します。
複数のアプリケーションが動作する場合は、個別のトレース情報を出力しません。
アプリケーションがマルチスレッド環境で動作する場合は、出力ファイル名の後にプロセスIDやセションIDなどの情報を自動的に付加して、個別のトレース情報を出力します。

出力レベル:

出力レベルには1または2を指定します。1を指定すると、DSOごとに集計された性能情報を出力します。2を指定すると、DSI単位の情報までも出力します。

データベース簡単運用の場合は、表またはインデックスに関する情報が出力されます。出力レベルによる出力内容に違いはありません。

省略した場合は、2が指定されたものとみなします。

CLI_SS_RATE

【指定形式】

CLI_SS_RATE = ([選択率1][,[選択率2][,[選択率3][,[選択率4][,[選択率5]]]]])

【実行パラメタの意味】

BETWEEN述語、比較述語、LIKE述語およびCONTAINS関数でインデックスを検索する時、インデックスの検索範囲の割合を0以上1以下の小数で指定します。値は、小数第6位まで指定できます。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_SS_RATE = (0.2,0.25,0.5,0.4,0.0001)

ただし、SQL文にASSIST指定のUSE_INDEXを指定した場合、ASSIST指定が優先され、CLI_SS_RATEの割合に関わらず、ASSIST指定に指定されたインデックスを使用した検索を行います。

【パラメタの意味】

選択率1:

BETWEEN述語を指定した場合のインデックスの検索範囲。省略した場合は、0.2が指定されたものとみなします。

選択率2:

比較述語“>”、“>=”、と“<”、“<=”でインデックスの検索開始位置および検索終了位置の両方が指定されている場合のインデックスの検索範囲。省略した場合は、0.25が指定されたものとみなします。

選択率3:

比較述語“>”、“>=”、“<”、“<=”でインデックスの検索開始位置または検索終了位置のみが指定されている場合のインデックスの検索範囲。省略した場合は、0.5が指定されたものとみなします。

選択率4:

LIKE述語を指定した場合のインデックスの検索範囲。省略した場合は、0.4が指定されたものとみなします。

選択率5:

CONTAINS関数を指定した場合のインデックスの検索範囲。省略した場合は、0.0001が指定されたものとみなします。

CLI_TID_SORT

【指定形式】

CLI_TID_SORT = ({YES | NO})

【実行パラメタの意味】

インデックス検索と表データ取得のアクセスモデルにおいて、TIDソートを利用するか否かを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_TID_SORT = (YES)

ただし、SQL文にASSIST指定のFIRST_ROWSを指定した場合、ASSIST指定が優先され、CLI_TID_SORTの指定に関わらず、TIDソートを利用しません。

【パラメタの意味】

YES:

TIDソートを利用します。

NO:

TIDソートを利用しません。

CLI_TID_UNION

【指定形式】

CLI_TID_UNION = ({YES | NO})

【実行パラメタの意味】

TIDユニオンマージのアクセスモデルを有効にするか否かを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_TID_UNION = (YES)

ただし、SQL文にASSIST指定のUSE_INDEXを指定した場合、ASSIST指定が優先され、CLI_TID_UNIONの指定に関わらず、ASSIST指定に指定されたインデックスを使用した検索を行います。

【パラメタの意味】

YES:

TIDユニオンマージのアクセスモデルを有効にします。TIDユニオンマージが効果的と判断した場合に選択します。

NO:

TIDユニオンマージのアクセスモデルを有効にしません。

CLI_USQL_LOCK

【指定形式】

CLI_USQL_LOCK = ({SH | EX})

【実行パラメタの意味】

UPDATE文:探索およびDELETE文:探索において、更新レコードを検索するアクセスモデルの表の占有モードを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_USQL_LOCK = (SH)

データベース簡単運用で、いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合は、以下の値が設定されます。

CLI_USQL_LOCK = (EX)

【パラメタの意味】

SH:

更新標的レコードを検索するアクセスモデルで共用モードで表を占有します。

EX:

更新標的レコードを検索するアクセスモデルで非共用モードで表を占有します。

◆リカバリに関する実行パラメタ

CLI_RCV_MODE

【指定形式】

CLI_RCV_MODE = ({RCV | NRCV})

【実行パラメタの意味】

アプリケーションのリカバリ水準を指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_RCV_MODE = (RCV)

【パラメタの意味】

RCV:

リカバリ機能を利用します。この場合、リカバリを適用しない(rdbrtrコマンドで利用規定にnオプションを指定している)データベースをアクセスすることはできますが、ログは取得されません。

NRCV:

リカバリ機能を利用しません。この場合、リカバリを適用する(rdbrtrコマンドで利用規定にnオプションを指定していない)データベースを更新することはできません。

◆予約語とSQL機能に関する実行パラメタ

CLI_SQL_LEVEL

【指定形式】

CLI_SQL_LEVEL = ({SQL88 | SQL92 | SQL95 | SQL96 | SQL2000 | SQL2007})

【実行パラメタの意味】

予約語とSQL機能のレベルを設定します。

予約語とSQL機能のレベルを設定することで、キーワードの範囲と利用可能なSQLの機能を変更できます。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

CLI_SQL_LEVEL = (SQL2007)

SQLの機能とそれを利用可能なレベルとの関係を以下に示します。

表に記載のない機能は、CLI_SQL_LEVELに関係なく利用可能です。

表4.2 CLI_SQL_LEVELと利用可能なSQLの機能一覧

CLI_SQL_LEVEL	SQLの機能	
SQL2007	数値関数	ACOS
		ASIN
		ATAN
		ATAN2
		COS
		EXP
		LN(注1)
		POWER
		SIGN
		SIN
		SQRT
		TAN
		ASCII
		OCTET_POSITION
	データ列値関数	LTRIM
		RTRIM
		OCTET_SUBSTRING
		CHR(注2)
	日時値関数	CNV_TIME
		CNV_TIMESTAMP
	XMLQUERY関数	
	述語	XMLEXISTS述語
	ROWNUM	
SQL2000以上	ファンクションルーチン	
	ロール	
	プロシジャ例外事象	条件宣言
		ハンドラ宣言
		SIGNAL文
		RESIGNAL文
SQL96以上	トリガ	
	行識別子	
	並列指定	
SQL95以上	プロシジャルーチン	

CLI_SQL_LEVEL	SQLの機能	
SQL92以上	定数	日時定数
		時間隔定数
	データ型	日時型
		時間隔型
		BLOB型
	順序定義	
	一時表定義	
	数値関数	POSITION
		EXTRACT
		CHAR_LENGTH
		CHARACTER_LENGTH
		OCTET_LENGTH
	データ列値関数	SUBSTRING
		UPPER
		LOWER
		TRIM
	日時値関数	CURRENT_DATE
		CURRENT_TIME
		CURRENT_TIMESTAMP
	CAST指定	
	CASE式	NULLIF
		COALESCE
		CASE
	結合表	
	カーソルのSCROLL指定	

注1) LN関数のLOGは、CLI_SQL_LEVELに関係なく利用可能です。

注2) CHR関数のCHARは、CLI_SQL_LEVELに関係なく利用可能です。

ポイント

SQL2007の関数名と同名のファンクションルーチンを定義している場合、関数とファンクションルーチンの動作の優先順位は以下になります。

ファンクションルーチン名へのスキーマ名修飾の有無	CLI_SQL_LEVEL	
	SQL2007	SQL2000
スキーマ名修飾有	ファンクションルーチン	ファンクションルーチン
スキーマ名修飾無	関数	ファンクションルーチン

SQL文にSQL2007の関数名と同名のファンクションルーチンを指定している場合、関数が優先して動作することで、ファンクションルーチンの結果と異なる結果になる場合があります。

ファンクションルーチンの動作を優先する場合は、CLI_SQL_LEVELにSQL2000を指定するか、ファンクションルーチン名をスキーマ名修飾してください。

【パラメタの意味】

SQL88:

予約語とSQL機能のレベルをSQL88とします。

SQL92:

予約語とSQL機能のレベルをSQL92とします。

SQL95:

予約語とSQL機能のレベルをSQL95とします。

SQL96:

予約語とSQL機能のレベルをSQL96とします。

SQL2000:

予約語とSQL機能のレベルをSQL2000とします。

SQL2007:

予約語とSQL機能のレベルをSQL2007とします。

参照

各予約語とSQL機能のレベルと、キーワードの関係については、“SQLリファレンス”を参照してください。

◆並列クエリに関する実行パラメタ

CLI_MAX_PARALLEL

【指定形式】

CLI_MAX_PARALLEL = (多重度)

【実行パラメタの意味】

SQL文でデータベースを並列に検索する場合の多重度を指定します。

SQL文でデータベースを並列に検索できるのは、問合せ式で並列指定を指定した場合です。表のDSIの数が、指定した多重度よりも少ない場合は、DSIの数を多重度として並列検索を行います。

なお、この実行パラメタは、サーバ用の動作環境ファイルのPARALLEL_SCANに“YES”を指定した場合、ODOSのオプションで設定できるパラメタのCLI_PARALLEL_SCANに“YES”を指定した場合、またはSQL文の問合せ式に並列指定“PARALLEL”を指定した場合に有効となります。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、n多重(注)で並列検索を行います。

注意

OSまたは仮想OSが認識している搭載CPUコア数×2が設定されます。ただし、RDB構成パラメタファイルにてRDBCPUNUMが指定されている場合は、RDBCPUNUM×2が設定されます。

【パラメタの意味】

多重度:

SQL文でデータベースを並列に検索する場合の多重度を2～100の範囲で指定します。

CLI_PARALLEL_SCAN

【指定形式】

CLI_PARALLEL_SCAN = ({YES | NO})

【実行パラメタの意味】

アプリケーション単位またはコネクション単位に、データベースを並列に検索するか否かを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が指定されたものとみなします。

CLI_PARALLEL_SCAN = (NO)

【パラメタの意味】

YES:

データベースを並列に検索します。この場合、そのアプリケーションの問合せを並列検索で実行できます。

ただし、以下のいずれかの条件を満たす場合、並列検索は実行されず従来のアクセス手順でデータベースにアクセスします。

- 表がDSI分割されていない、または1つのDSIに対するアクセスの場合
- クラスタキーを利用したデータベースアクセスが可能な場合
- 探索条件にROW_IDを指定した検索の場合
- インデックスを利用したデータベースアクセスが可能な場合

NO:

データベースを並列に検索しません。

◆その他に関する実行パラメタ

CLI_ARC_FULL

【指定形式】

CLI_ARC_FULL = ({RETURN | WAIT})

【実行パラメタの意味】

アーカイブログファイルが満杯状態になったとき、エラー復帰するか否かを指定します。

本実行パラメタの設定は、“表4.1 ODOSのオプションで設定できるパラメタの種類”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

CLI_ARC_FULL = (RETURN)

【パラメタの意味】

RETURN:

エラーとしてアプリケーションに復帰します。

WAIT:

空きのアーカイブログファイルが作成されるまで待ちます。

注意

- “WAIT”を指定した場合、空きのアーカイブログファイルが作成されるまでアプリケーションは無応答状態となりますので注意してください。
- 本実行パラメタの指定が、互いに異なる複数のアプリケーションが同時に同じDSIを扱うような運用はしないでください。
  例えば、以下の事象が発生する場合があります。
  トランザクションAは、CLI_ARC_FULL=RETURNを設定し、アーカイブログ満杯の状態であるとします。トランザクションBは、CLI_ARC_FULL=WAITを設定し、アーカイブログ満杯の状態で資源を占有しているとします。このとき、

トランザクションBは、空きのアーカイブログファイルが作成されるまで待ちます。同じ資源にアクセスしているトランザクションAは、トランザクションBの終了を待つため、CLI_ARC_FULL=RETURNを設定しているにもかかわらず、無応答状態になります。
この場合は、rdblkinfコマンドのlオプションやrdblogコマンドのVオプションかつaオプションで排他やアーカイブログの状況を確認できます。また、アーカイブログに関する以下のシステムメッセージが表示されます。これらの情報をもとに、バックアップ可能なアーカイブログファイルをバックアップするか、または新規にアーカイブログファイルを追加して対処してください。

```
rdb: WARNING: qdg13336w: 転送可能なアーカイブログ域が不足しています
rdb: ERROR: qdg03132u: アーカイブログファイルが満杯です
```

ー 以下のいずれかの処理中にシステムロググループまたはユーザロググループのアーカイブログファイルが満杯状態になった場合の振る舞いについては、本実行パラメタの指定に関わらずシステム用の動作環境ファイルに指定したARC_FULLパラメタに従います。

- SQL文の実行で自動容量拡張が動作する。
- SQL文中に順序を直接指定して番号を採番する。
- 表定義のDEFAULT句に順序を指定して、自動的に取得した値を表に挿入する。

付録A　Web連携のサンプルプログラムの使用方法

Web連携のサンプルプログラムの使用方法について説明します。

A.1　ASPのサンプルプログラム概要

Symfoware Server クライアント機能をインストールした時にインストールされるASPを使用した以下の基本的なデータベース操作のサンプルプログラムを紹介します。

- ランチデータベースのサンプルプログラム
- カーソル操作を使用したサンプルプログラム

注意

ASPを実行するには、Webサーバ上にASPがインストールされている必要があります。

記載内容は以下の環境で動作検証しています。

動作環境	製品名
Webサーバ	IIS 5.0
Webブラウザ	Internet Explorer 11

ランチデータベースのサンプルプログラム

以下にランチデータベースのサンプルプログラムを示します。

- 環境作成プログラム

  ランチデータベースプログラムを実行するための環境を作成します。

  このプログラムではテーブルおよびインデックスの作成を行います。

- 静的SQL文を使用したランチデータベースプログラム

  このプログラムでは静的SQL文を使用した基本的なデータ操作の方法を紹介します。

  静的SQL文を使用してデータの参照、追加、削除、更新および検索を行います。

- 動的SQL文を使用したランチデータベースプログラム

  このプログラムでは動的SQL文を使用した基本的なデータ操作の方法を紹介します。

  動的SQL文を使用してデータの参照、追加、削除、更新および検索を行います。

カーソル操作を使用したサンプルプログラム

このサンプルプログラムには次のプログラムがあります。

- 環境作成プログラム

  データ操作プログラムを実行するための環境を作成します。

  このプログラムではテーブルおよびインデックスの作成とサンプルデータの格納を行います。

- データ操作プログラム

  このプログラムではRecordsetオブジェクトを使用したデータ操作の方法を紹介します。

  カーソル操作を使用してデータの参照、追加、更新および削除を行います。

A.2 ASPのサンプルプログラム構成

Symfoware Server クライアント機能をインストールした時にインストールされるASPサンプルプログラムの構成およびインストール先を説明します。

Symfoware Serverに接続するプログラム

Symfoware Serverに接続する処理を記載しているサンプルプログラムです。

プログラム	処理
ConnectionDB.inc	Symfoware Serverに接続

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT)¥ODOS¥Sample¥ASPSample

ランチデータベース

環境作成プログラム

ランチデータベースを利用する前にSymfoware/RDBのデータベースの環境を作成するサンプルプログラムです。

プログラム名	処理
Lunch_Env_make.htm	ランチデータベース環境作成のメニュー
Lunch_Env_make.asp	表およびインデックスの作成と削除

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT)¥ODOS¥Sample¥ASPSample¥Lunchdb¥Lunch_Env_make

静的SQL文および動的SQL文を使用したランチデータベースプログラム

ランチデータベースを操作する静的SQL文と動的SQL文のサンプルプログラムです。

プログラム名	処理
lunch.htm	メインメニュー
entryform.htm	登録のフォーム
entry.asp	登録
look.asp	一覧表示
Search.htm	検索のフォーム
Shopsearch.asp	店名で検索
Genresearch.asp	ジャンルで検索
Pricesearch.asp	価格で検索
Profitsearch.asp	満足度で検索
Update.htm	修正のフォーム
Update.asp	修正
password.htm	パスワード入力フォーム
pdoltout.asp	削除のフォーム
boltout.asp	1行削除

プログラム名	処理
Allboltout.asp	全行削除
Ranking.asp	ランキング

インストール先フォルダ

静的SQL文を使用したランチデータベース

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT)
¥ODOS¥Sample¥ASPSample¥Lunchdb¥lunch1

動的SQL文を使用したランチデータベース

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT)
¥ODOS¥Sample¥ASPSample¥Lunchdb¥lunch2

カーソル操作を使用したサンプルプログラム

環境作成プログラム

カーソル操作を使用したサンプルプログラムを利用する前に、Symfoware/RDBのデータベース環境を作成するサンプルプログラムです。

プログラム名	処理
env_make.htm	環境作成のフォーム
env_make.asp	表およびインデックスの作成と削除、データの挿入

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT)
¥ODOS¥Sample¥ASPSample¥ODOSSample¥env_make

データ操作プログラム

カーソル操作を使用したサンプルプログラムのデータ操作を行うメインメニューを表示するサンプルプログラムです。

プログラム名	処理
ODOSSample.htm	メインメニュー

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT)
¥ODOS¥Sample¥ASPSample¥ODOSSample

カーソル操作を使用したデータの参照を行うサンプルプログラムです。

プログラム名	処理
consult.asp	Recordsetでの参照

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT) ¥ODOS¥Sample¥ASPSample¥ODOSSample¥consult

カーソル操作を使用したデータの更新を行うサンプルプログラムです。

プログラム名	処理
rec_renew.asp	Recordsetでの更新

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT) ¥ODOS¥Sample¥ASPSample¥ODOSSample¥rec_renew

SQL文を直接指定したデータの更新を行うサンプルプログラムです。

プログラム名	処理
exe_renew.asp	Executeメソッドでの更新

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT) ¥ODOS¥Sample¥ASPSample¥ODOSSample¥exe_renew

インクルードファイル

カーソル操作を使用するサンプルプログラムで利用しているインクルードです。

インクルードファイル名	内容
adoodos.inc	ADO定数の定義
Loop.inc	カーソルの移動操作のプロシジャ
table.inc	テーブルタグのプロシジャ

インストール先フォルダ

Symfoware Server クライアント機能のインストールフォルダ(C:¥SFWCLNT) ¥ODOS¥Sample¥ASPSample¥ODOSSample¥inc

A.3 ASPのサンプルプログラム実行手順

ASPのサンプルプログラムを実行する前に必要な準備について説明します。

IISの設定については、“4.3.3 環境設定”を参照してください。

動作させるためにASPファイルの以下の個所を変更してください。

1. ConnectionDB.incファイルを開きます。

2. ConnectionDB.incファイルの6行目と14行目のスクリプト("DSN=SmpODOS;UID=USER01;PWD=PASS01"部分)を、登録したデータソース名と、RDBが使えるユーザ名とパスワードに書き換えてください。インストール時は、データソース名をSmpODOS、ユーザ名をUSER01、パスワードをPASS01と設定されています。

   6行目

```
OBJdbCommand.ActiveConnection =
"DSN=SmpODOS;UID=USER01;PWD=PASS01"
```

   14行目

```
OBJdbConnection.Open
"DSN=SmpODOS;UID=USER01;PWD=PASS01"
```

3. ブラウザのアドレスに以下のURLを指定してください。

```
http:// Webサーバ名/ASPSamp/index.htm
```

4. 下記画面が表示されます。

A.4 SQL文実行までの手順

ASPに付属のADOを使用することで、ランチデータベースに格納されている情報にアクセスすることができます。ここでは、ランチデータベースへアクセスし、SQL文を実行するまでの手順について説明します。

SQL文実行までの手順

1. Connectionオブジェクトのインスタンスを作成します。

```
<%Set OBJdbConnection = Server.CreateObject("ADODB.Connection")%>
```

2. ランチデータベースに接続するために、ConnectionオブジェクトのOpenメソッドを使用します。

   データソース名、RDBが使えるユーザ名とパスワードを指定します。

```
<%OBJdbConnection.Open "DSN=SmpODOS;UID=USER01;PWD=PASS01"%>
```

   これは、ODBCデータソース名をSmpODOS、ユーザ名をUSER01、パスワードをPASS01としたときのコード例です。

3. SQL文を作成します。変数SQLQueryにSQL文を格納します。

```
SQLQuery = "SELECT ジャンル,店名,メニュー,価格 FROM LUNCHSCH.LUNCHTBL WHERE ジャンル=N'"
SQLQuery = SQLQuery + GENRE + "'"
```

   GENREはFormsコレクションより取得した情報を格納した変数です。+演算子(または &演算子)により、変数SQLQueryに格納されている文字列のあとに変数GENREに格納されている文字列を連結しています。その後、さらに“'”を連結しています。

4. ConnectionオブジェクトのExecuteメソッドを使用してSQL文実行します。

   返されたレコードを結果セット (LunchList) に格納します。

```
<%Set LunchList = OBJdbConnection.Execute(SQLQuery)%>
```

また、SQL文を変数SQLQueryに割り当てずに直接Executeメソッドに渡すこともできます。

```
<%Set LunchList = OBJdbConnection.Execute("SELECT ジャンル,店名,メニュー,価格 FROM LUNCHSCH.LUNCHTBL
WHERE ジャンル=N'" + GENRE + "'")%>
```

ただし、SQL文が長い場合には、文字列をいったんSQLQueryなどの変数名に割り当ててからそれをExecuteメソッドに渡したほうが、コードが読みやすくなります。

結果セットの表示方法は、“A.5 結果セットを表示するには”を参照してください。

A.5 結果セットを表示するには

結果セットの表示方法について説明します。結果セットは、SQLのSELECT文の中で指定された列名によって、結果セットの構造が決まります。このため、返却された結果セットの行は、結果セットの行に対してループを実行するだけで簡単に表示できます。

以下に示すコード例では、返されたデータはHTMLのテーブルの行として表示されます。

1. HTMLのテーブルのヘッダを定義します。

```
<TABLE BORDER=0>
<TR>
<TD ALIGN=CENTER WIDTH=100 BGCOLOR="#800000">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=3>ジャンル</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=200 BGCOLOR="#6090EF">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=3>店名</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=200 BGCOLOR="#6090EF">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=3>メニュー</FONT>
</TD>
<TD ALIGN=CENTER WIDTH=150 BGCOLOR="#6090EF">
<FONT STYLE="ARIAL NARROW" COLOR="#ffffff" SIZE=3>価格</FONT>
</TD>

</TR>
```

2. Do. . . Loopステートメントによりスクリプトコマンドのほか、HTML のテキストやタグを繰り返すことができます。ループが実行されるごとにテーブルの行がHTMLによって作成され、返されたデータがスクリプトコマンドによって挿入されます。

```
<% Do While Not LunchList.EOF %>
      <TR>
      <TD BGCOLOR="f7efde" ALIGN=CENTER>
      <FONT STYLE="ARIAL NARROW" SIZE=3>
      <%= LunchList("ジャンル")%>
      </FONT></TD>
      <TD BGCOLOR="f7efde" ALIGN=CENTER>
      <FONT STYLE="ARIAL NARROW" SIZE=3>
      <%= LunchList("店名")%>
      </FONT></TD>
      <TD BGCOLOR="f7efde" ALIGN=CENTER>
      <FONT STYLE="ARIAL NARROW" SIZE=3>
      <%= LunchList("メニュー") %>
      </FONT></TD>
      <TD BGCOLOR="f7efde" ALIGN=CENTER>
      <FONT STYLE="ARIAL NARROW" SIZE=3>
      <%Select Case LunchList("価格")
              Case "1"
              Response.Write " 500円以下"
              Case "2"
```

```
                Response.Write " 500～1000円"
                Case "3"
                Response.Write " 1000～1500円"
                Case "4"
                Response.Write " 1500円以上"
        End Select
        %>
        </FONT></TD>
        </TR>
 </TABLE>

<%
LunchList.MoveNext
Loop
%>
```

ループが完了したら、MoveNextメソッドを使って結果セットの行ポインタを1行下に移動します。このステートメントもDo. . . Loop ステートメントの内部にあるため、ファイルの終端に達するまで繰り返されます。

3. ランチデータベースとの接続を切断するために、ConnectionオブジェクトのCloseメソッドを使用します。

```
<%OBJdbConnection.Close%>
```

A.6 静的SQL文と動的SQL文のデータ操作プログラム

静的SQL文と動的SQL文のデータ操作プログラムについて説明します。

A.6.1 プログラムの処理

ランチデータベース

サンプルプログラムで行っている処理はそれぞれ以下のとおりです。

環境作成プログラム

プログラム名	処理
Lunch_Env_make.htm	ランチデータベース環境作成の各処理を選択するメニューを表示します。ここで、各処理に切り分けます。
Lunch_Env_make.asp	ADO接続を使用してコネクション環境を作成し、Executeメソッドで以下のSQL文を実行します。 ・ CREATE TABLE文 ・ CREATE INDEX文

静的SQL文と動的SQL文を使用したランチデータベースプログラム

プログラム名	処理
lunch.htm	ランチデータベースの各処理を選択するメニューを表示します。ここで、各処理に切り分けます。
entryform.htm	登録のフォームを表示します。
entry.asp	登録情報をフォームより受け取ります。ADO接続を使用してコネクション環境を作成し、Executeメソッドで以下のSQL文を実行します。 ・ SELECT文で集合関数MAXを使い追番の最大値を調べる ・ INSERT文
look.asp	ADO接続を使用してコネクション環境を作成します。Executeメソッドで SELECT文を実行し一覧を表示します。

プログラム名	処理
search.htm	検索のフォームを表示します。
shopsearch.asp	店名をフォームより受け取ります。ADO接続を使用してコネクション環境を作成します。ExecuteメソッドでSELECT文を実行し検索結果を表示します。
genresearch.asp	ジャンルをフォームより受け取ります。ADO接続を使用してコネクション環境を作成します。ExecuteメソッドでSELECT文を実行し検索結果を表示します。
pricesearch.asp	価格をフォームより受け取ります。ADO接続を使用してコネクション環境を作成します。ExecuteメソッドでSELECT文を実行し検索結果を表示します。
profitsearch.asp	満足度をフォームより受け取ります。ADO接続を使用してコネクション環境を作成します。ExecuteメソッドでSELECT文を実行し検索結果を表示します。
update.htm	修正のフォームを表示します。
update.asp	修正情報をフォームより受け取ります。ADO接続を使用してコネクション環境を作成し、ExecuteメソッドでUPDATE文を実行します。
password.htm	パスワード入力フォームを表示します。
pdoltout.asp	パスワードを確認し、削除のフォームを表示します。
boltout.asp	削除する行の追番をフォームより受け取ります。ADO接続を使用してコネクション環境を作成し、ExecuteメソッドでDELETE文:探索を実行します。
allboltout.asp	ADO接続を使用してコネクション環境を作成し、Executeメソッドで DELETE文を実行します。
ranking.asp	ADO接続を使用してコネクション環境を作成し、Executeメソッドで SELECT文を実行し結果を表示します。

静的SQL文と動的SQL文のコーディングの違い

ASPでの静的SQL文と動的SQL文のコーディングの違いについて、コーディング例で説明します。

ランチデータベースプログラムのgenresearch.aspより、SQL文の生成から実行までのコーディングの異なる部分を抜粋しましたので参考にしてください。

静的SQL文のコーディング例

```
'Connectionオブジェクトを作成
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
'DB接続
OBJdbConnection.Open "DSN=SmpODOS;UID=RDASV;PWD=RDASV"
'SQL文　生成
SQLQuery = "SELECT ジャンル,店名,メニュー,価格 FROM LUNCHSCH.LUNCHTBL WHERE ジャンル=N'"
SQLQuery = SQLQuery + GENRE + "'"
'SQL文を実行
Set LunchList = OBJdbConnection.Execute(SQLQuery)
```

動的SQL文のコーディング例

```
'Commandオブジェクトを作成
Set OBJdbCommand = Server.CreateObject("ADODB.Command")
'開いている接続とCommandオブジェクトとを関連付ける
OBJdbCommand.ActiveConnection = "DSN=SmpODOS;UID=RDASV;PWD=RDASV"
'SQLステートメント生成
ExSQL = "SELECT ジャンル,店名,メニュー,価格 FROM LUNCHSCH.LUNCHTBL WHERE ジャンル=?"
'コマンドのテキスト版(SQLステートメント)などを定義する
OBJdbCommand.CommandText = ExSQL
```

```
'Parameter オブジェクトを作成
set Pm1 = OBJdbCommand.CreateParameter("GE",200,1,8,GENRE)
'Parameters コレクションに追加
OBJdbCommand.Parameters.Append Pm1
'コマンドの実行
Set LunchList = OBJdbCommand.Execute
```

A.6.2 起動画面

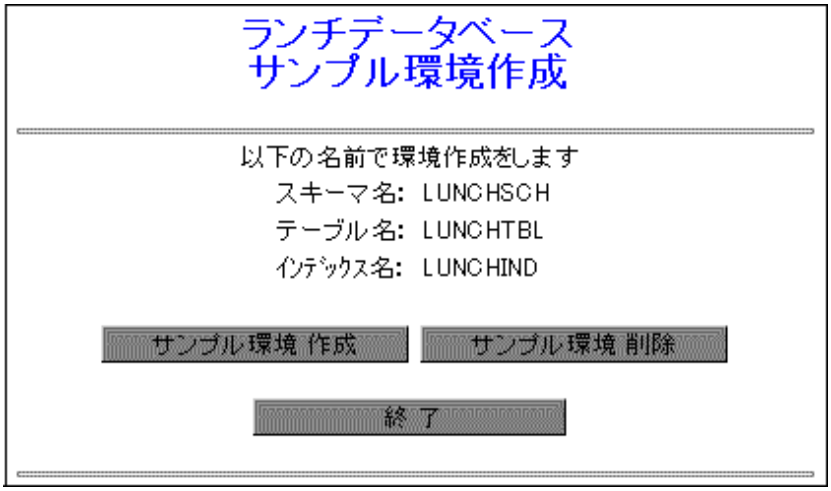
ランチデータベースの起動画面の操作方法を説明します。

実行したい処理をクリックしてください。

- ランチデータベース環境作成
- 静的SQL文を使用したランチデータベース
- 動的SQL文を使用したランチデータベース

A.6.3 環境作成プログラム

環境作成プログラムの操作方法を説明します。



環境を作成する場合は[サンプル環境 作成]ボタンをクリックします。

環境を削除する場合は[サンプル環境 削除]ボタンをクリックします。

[終了]ボタンで環境作成プログラムを終了し、“起動画面”に戻ります。

A.6.4 ランチデータベース操作方法

ランチデータベース操作方法(メインメニュー)

ランチデータベースの操作方法を説明します。

静的SQL文を使用したランチデータベースと、動的SQL文を使用したランチデータベースで操作方法に違いはありません。

実行したい処理をクリックしてください。

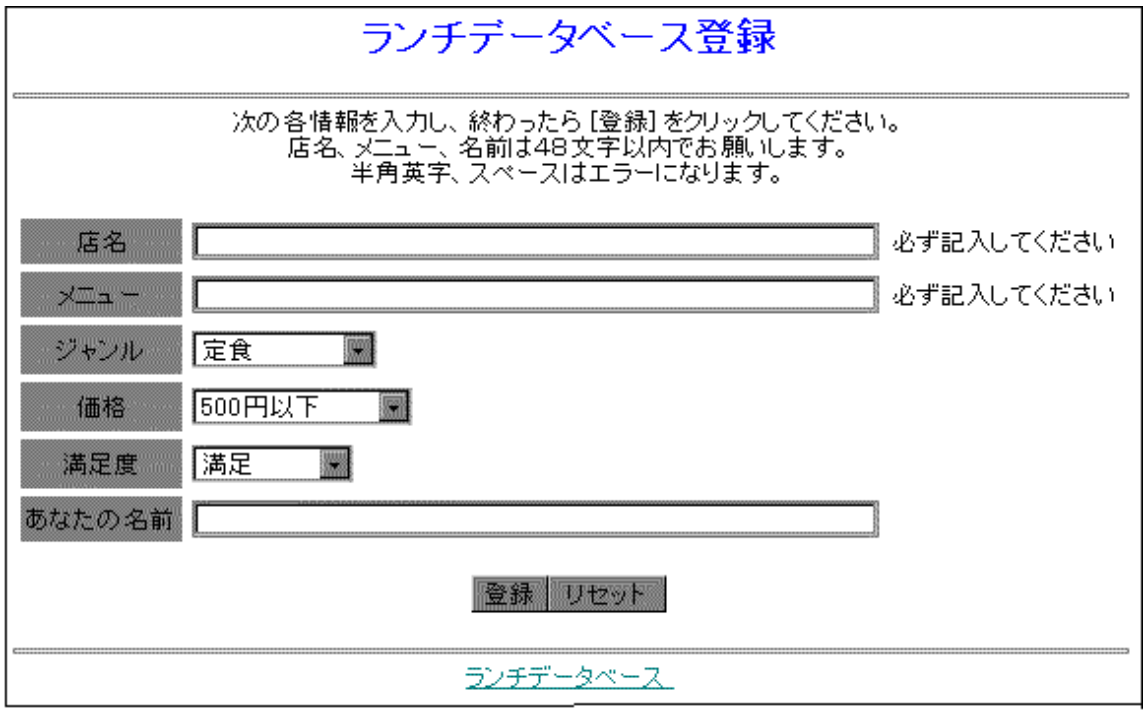
- ランチデータベース登録
- ランチデータベース一覧表示
- ランチデータベース検索
- ランチデータベース修正
- ランチデータベース削除
- よく行くお店ランキング

LUNCH DB INDEXをクリックすると起動画面に戻ります。

ランチデータベース操作方法(ランチデータベース登録)

ランチデータベース登録の操作方法を説明します。

1. メインメニューで[ランチデータベース登録]ボタンをクリックすると下記画面が表示されます。



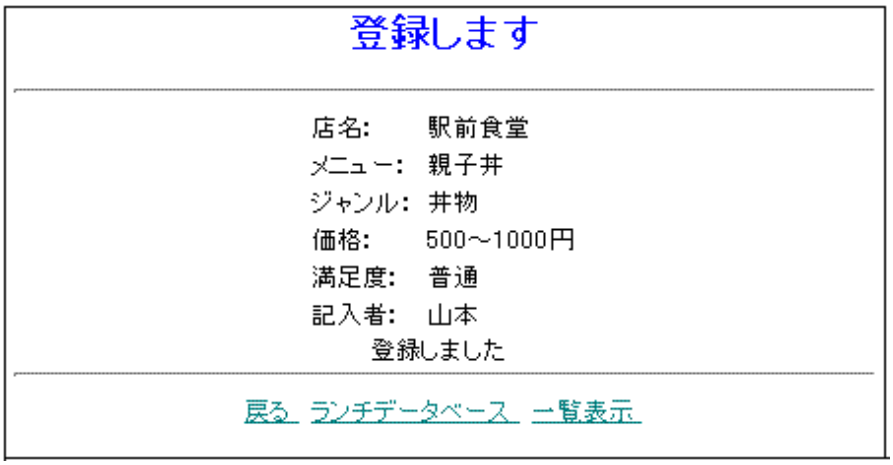
2. 各項目にそれぞれ店名、メニュー、ジャンル、価格、満足度および記入者の名前を入力します。

   店名とメニューは、必ず入力してください。

3. [登録]ボタンをクリックします。

   なお、[リセット]ボタンは、入力内容を初期化します。

4. 下記画面が表示され、登録されます。



5. 続けて登録する場合は[戻る]をクリックしてください。

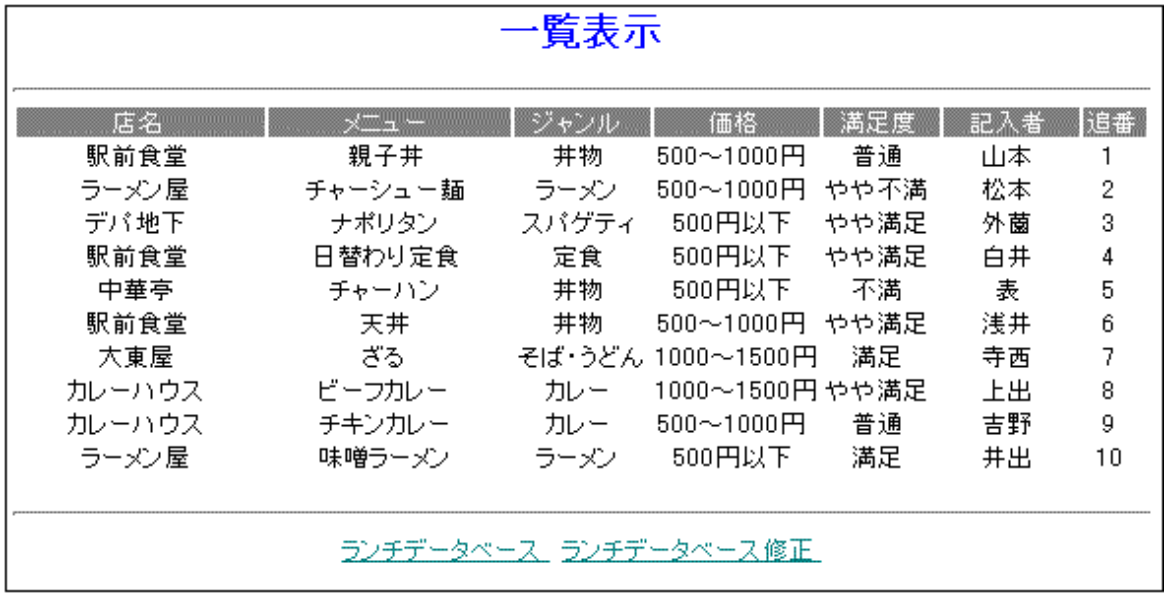
   メインメニューに戻る場合は[ランチデータベース]をクリックしてください。

   登録されたことを確認する場合は[一覧表示]をクリックしてください。

ランチデータベース操作方法(ランチデータベース一覧表示)

ランチデータベース一覧表示の操作方法を説明します。

1. メインメニューで[ランチデータベース一覧表示]ボタンをクリックすると下記画面が表示されます。



一覧表示

店名	メニュー	ジャンル	価格	満足度	記入者	追番
駅前食堂	親子丼	丼物	500～1000円	普通	山本	1
ラーメン屋	チャーシュー麺	ラーメン	500～1000円	やや不満	松本	2
デパ地下	ナポリタン	スパゲティ	500円以下	やや満足	外薗	3
駅前食堂	日替わり定食	定食	500円以下	やや満足	白井	4
中華亭	チャーハン	丼物	500円以下	不満	表	5
駅前食堂	天丼	丼物	500～1000円	やや満足	浅井	6
大東屋	ざる	そば・うどん	1000～1500円	満足	寺西	7
カレーハウス	ビーフカレー	カレー	1000～1500円	やや満足	上出	8
カレーハウス	チキンカレー	カレー	500～1000円	普通	吉野	9
ラーメン屋	味噌ラーメン	ラーメン	500円以下	満足	井出	10

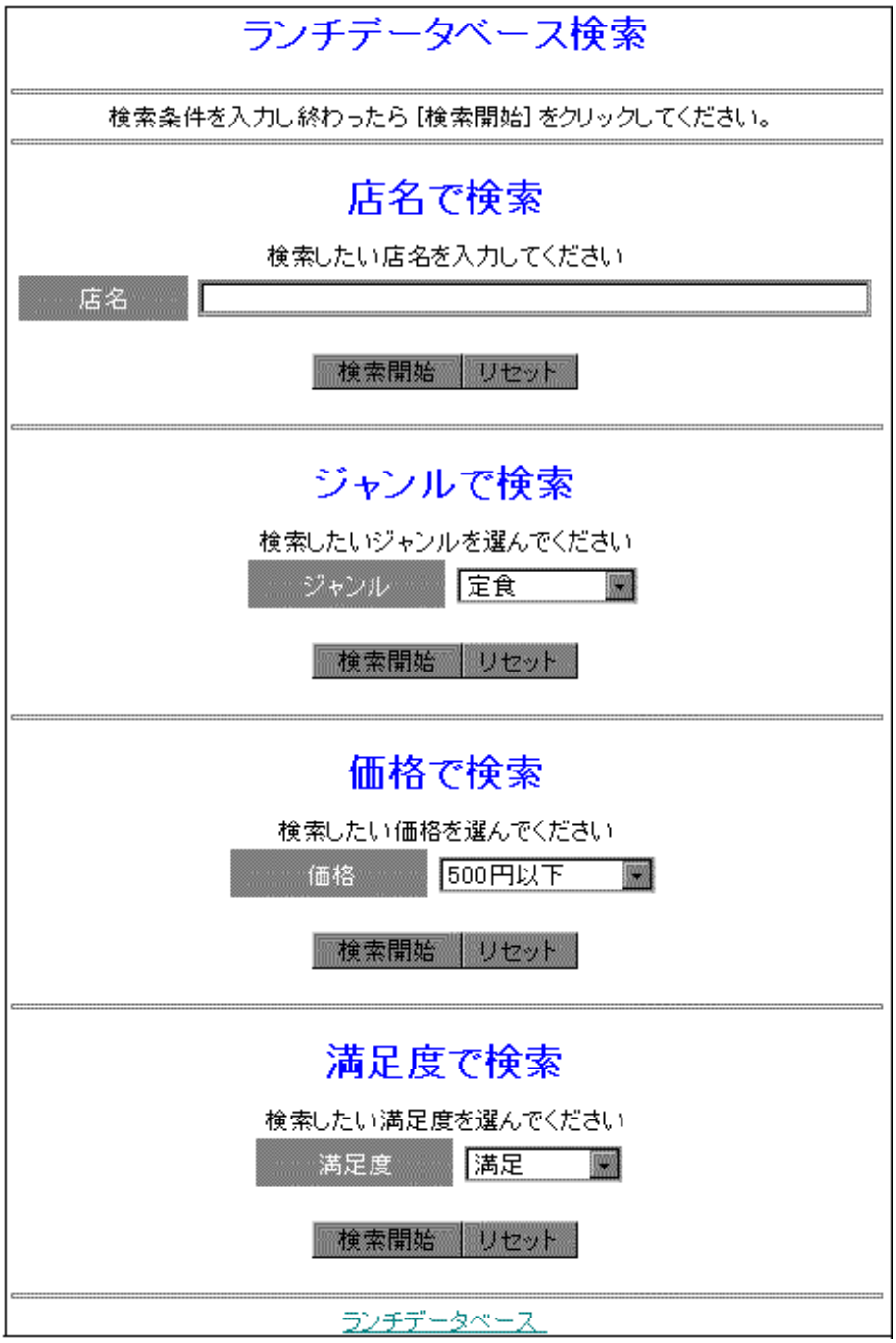
ランチデータベース　ランチデータベース修正

2. メインメニューに戻る場合は[ランチデータベース]をクリックしてください。
   入力に間違いがある場合は[ランチデータベース修正]をクリックしてください。

ランチデータベース操作方法(ランチデータベース検索)

ランチデータベース検索の操作方法を説明します。

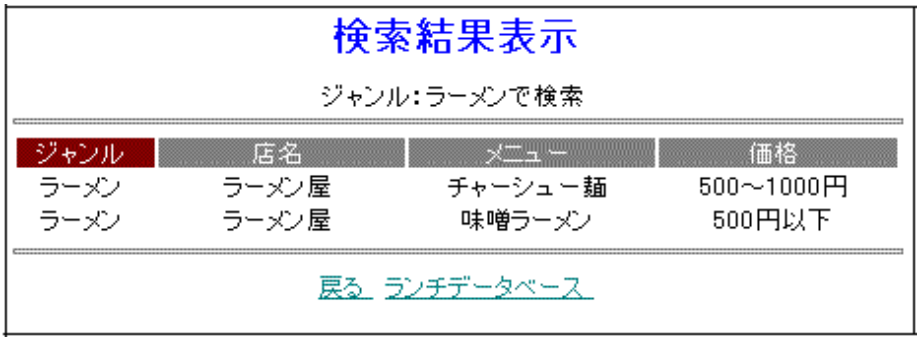
1. メインメニューで[ランチデータベース検索]ボタンをクリックすると下記画面が表示されます。



2. 検索したい項目に入力します。
3. 検索したい項目の[検索開始]ボタンをクリックします。

   なお、[リセット]ボタンは、入力内容を初期化します。

4. 下記画面が表示されます。



5. 他の項目で検索したい場合は[戻る]をクリックしてください。

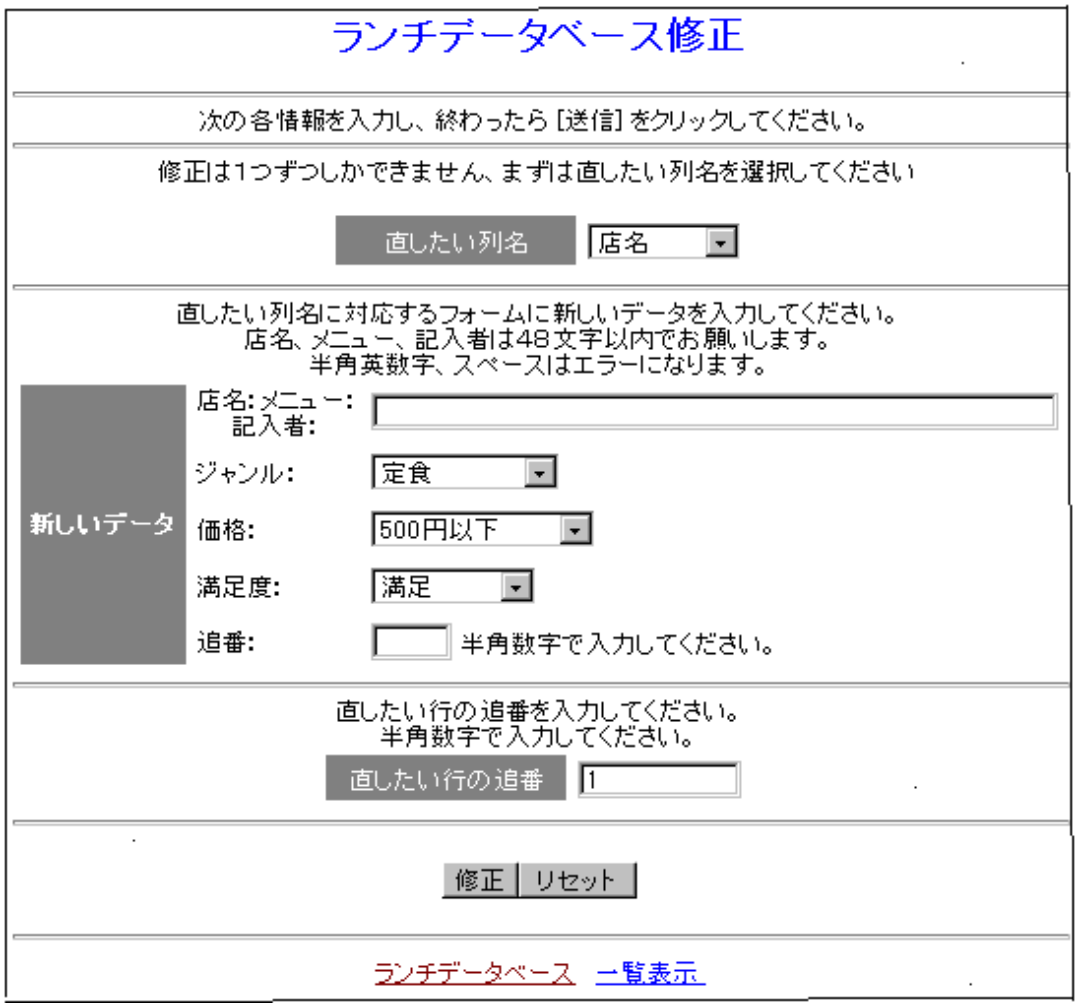
   メインメニューに戻る場合は[ランチデータベース]をクリックしてください。

ランチデータベース操作方法(ランチデータベース修正)

ランチデータベース修正の操作方法を説明します。

1. メインメニューで[ランチデータベース修正]ボタンをクリックすると下記画面が表示されます。

   修正は1つずつしかできません。



2. 直したい列名を選択します。
3. 直したい列名に対応する項目に新しいデータを入力します。

4. 直したい行の追番を入力します。
5. [修正]ボタンをクリックします。
   なお、[リセット]ボタンは、入力内容を初期化します。

   下記画面が表示され、修正されます。

6. 続けて修正する場合は[戻る]をクリックしてください。

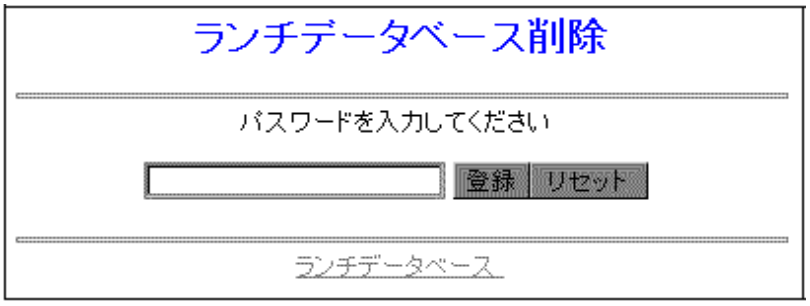
   メインメニューに戻る場合は[ランチデータベース]をクリックしてください。

   修正されているか確認したい場合は[一覧表示]をクリックしてください。

ランチデータベース操作方法(ランチデータベース削除)

ランチデータベース削除の操作方法を説明します。

1. メインメニューで[ランチデータベース削除]ボタンをクリックすると下記画面が表示されます。

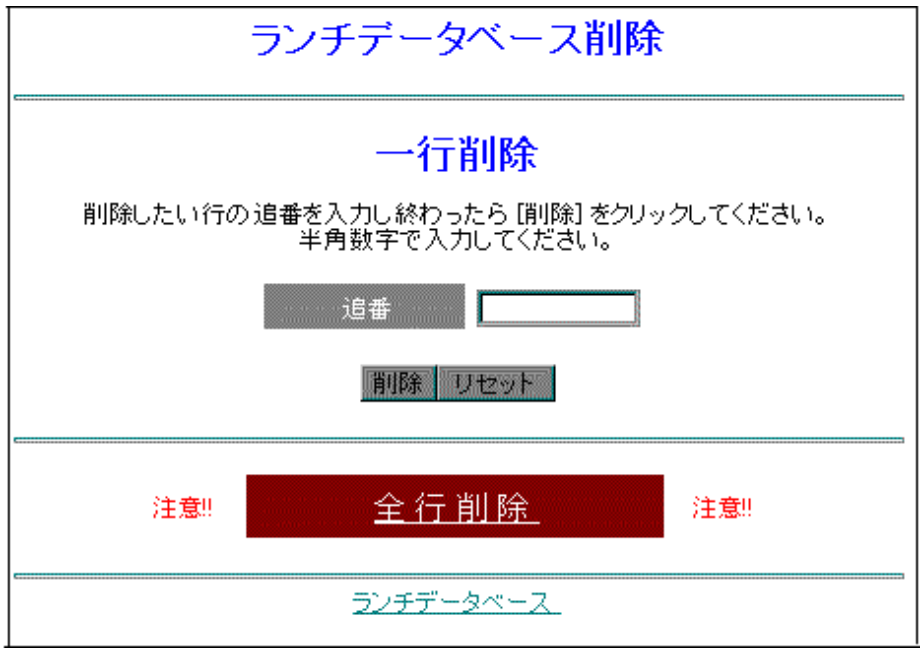


2. パスワードを入力します。

   パスワード：looklookokdes

3. [登録]ボタンをクリックすると下記画面が表示されます

   なお、[リセット]ボタンは、入力内容を初期化します。

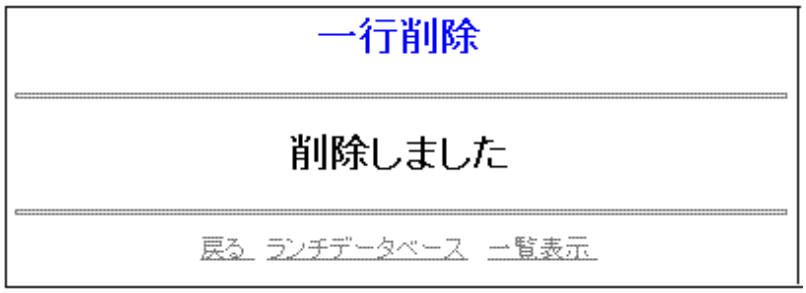


4. 一行削除したい場合は削除したい行の追番を入力し[削除]ボタンをクリックしてください。

   なお、[リセット]ボタンで入力内容を初期化します。

   全行削除したい場合は[全行削除]ボタンをクリックしてください。

5. 下記画面が表示され、削除されます。



6. 続けて削除する場合は[戻る]をクリックしてください。

   メインメニューに戻る場合は[ランチデータベース]をクリックしてください。

   削除されているか確認したい場合は[一覧表示]をクリックしてください。

ランチデータベース操作方法(よく行くお店ランキング)

よく行くお店ランキングの操作方法を説明します。

1. メインメニューで[よく行くお店ランキング]ボタンをクリックすると下記画面が表示されます。

2. メインメニューに戻る場合は[ランチデータベース]をクリックしてください。

   検索したい場合は[検索]をクリックしてください。

A.7 カーソル操作プログラム

カーソル操作を使用したサンプルプログラムについて説明します。

A.7.1 プログラムの処理

カーソル操作を使用したサンプルプログラム

サンプルプログラムで行っている処理はそれぞれ以下のとおりです。

環境作成プログラム

プログラム名	処理
env_make.htm	環境作成のフォームを表示します。
env_make.asp	ADO接続を使用してコネクション環境を作成し、Executeメソッドで以下のSQL文を実行します。 ・ CREATE TABLE文 ・ CREATE INDEX文 ・ INSERT文(100回)

データ操作プログラム

プログラム名	処理
ODOSSample.htm	各処理を選択するメニューを表示します。ここで、各処理に切り分けます。
Consult.asp	ADO接続を使用してRecordsetオブジェクトを開き、Fieldオブジェクトの各プロパティを参照して、接続先のレコード情報を表示します。
rec_renew.asp	ADO接続を使用してRecordsetオブジェクトを開き、開いたオブジェクトに対して以下の処理を行います。 ・ Deleteメソッドを使用したカレント行の削除 ・ AddNewメソッドを使用した行の追加 ・ Updateメソッドを使用したカレント行の更新

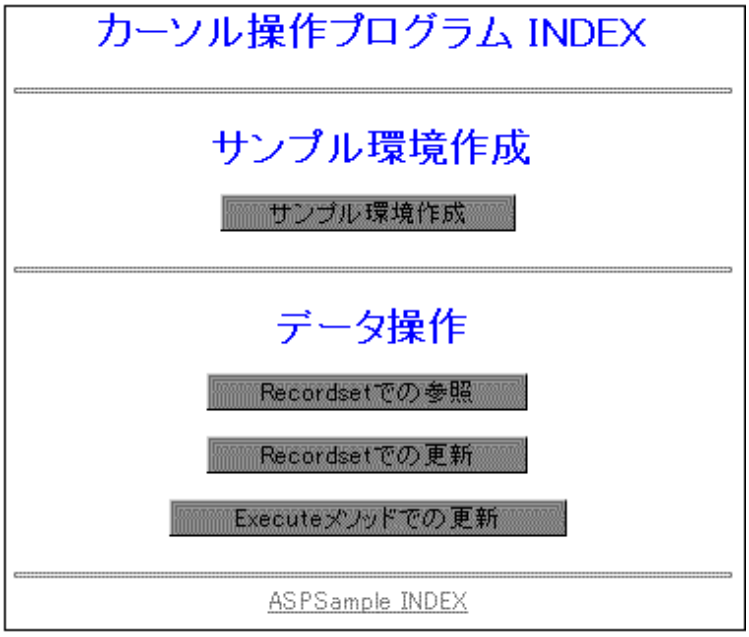
プログラム名	処理
exe_renew.asp	ADO接続を使用してコネクション環境を作成し、Executeメソッドで SQL文を実行します。

インクルードファイル

インクルードファイル名	内容
adoodos.inc	ADO定数の定義
Loop.inc	カーソルを指定された位置まで移動する操作とカーソルを末尾へ移動する操作のプロシジャ
table.inc	テーブルタグのプロシジャ

A.7.2 起動画面

カーソル操作プログラムの起動画面の操作方法を説明します。



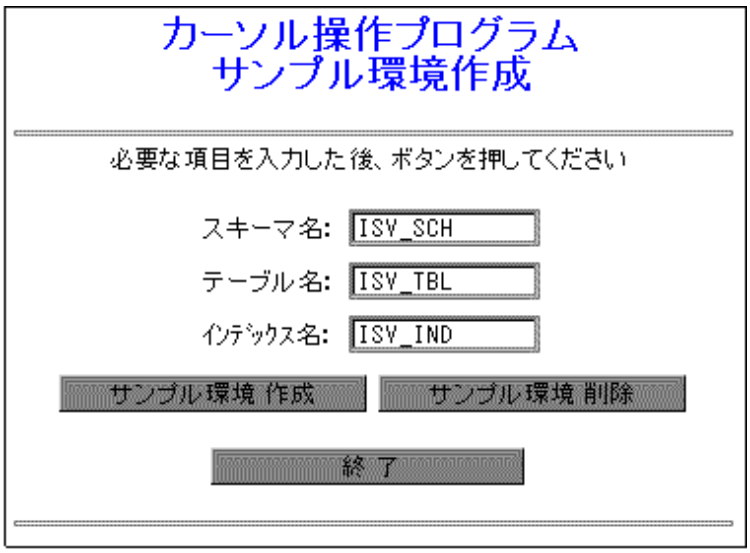
実行したい処理のボタンをクリックしてください。

- サンプル環境作成プログラム
  - サンプル環境作成
- データ操作プログラム
  - Recordsetでの参照
  - Recordsetでの更新
  - Executeメソッドでの更新

A.7.3 環境作成プログラム

環境作成プログラムの操作方法を説明します。

1. 起動画面で[サンプル環境作成]ボタンをクリックすると下記画面が表示されます。

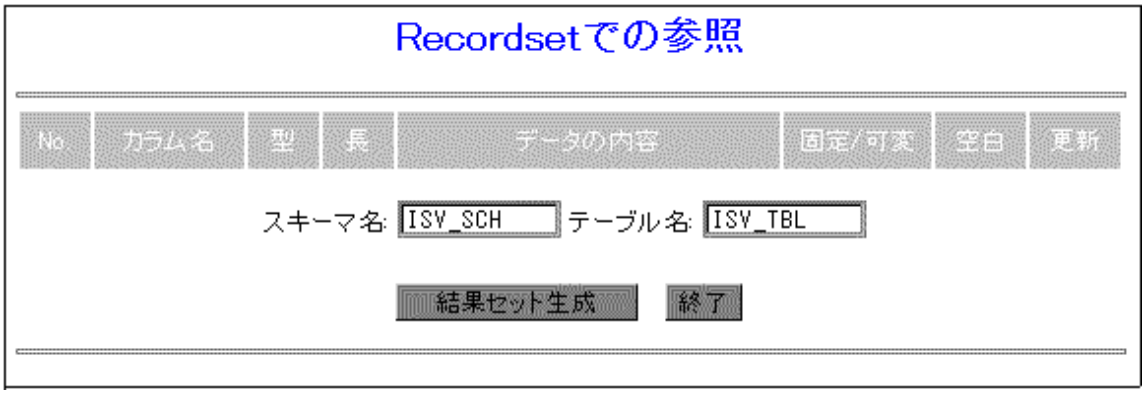


2. 各項目にそれぞれスキーマ名、テーブル名、インデックス名を入力します。
   環境を作成する場合は[サンプル環境 作成]ボタンをクリックします。
   環境を削除する場合は[サンプル環境 削除]ボタンをクリックします。
   [終了]ボタンをクリックすると“起動画面”に戻ります。

A.7.4 [Recordsetでの参照(ADO)]画面

データ操作プログラムの[Recordsetでの参照]画面の操作方法を説明します。

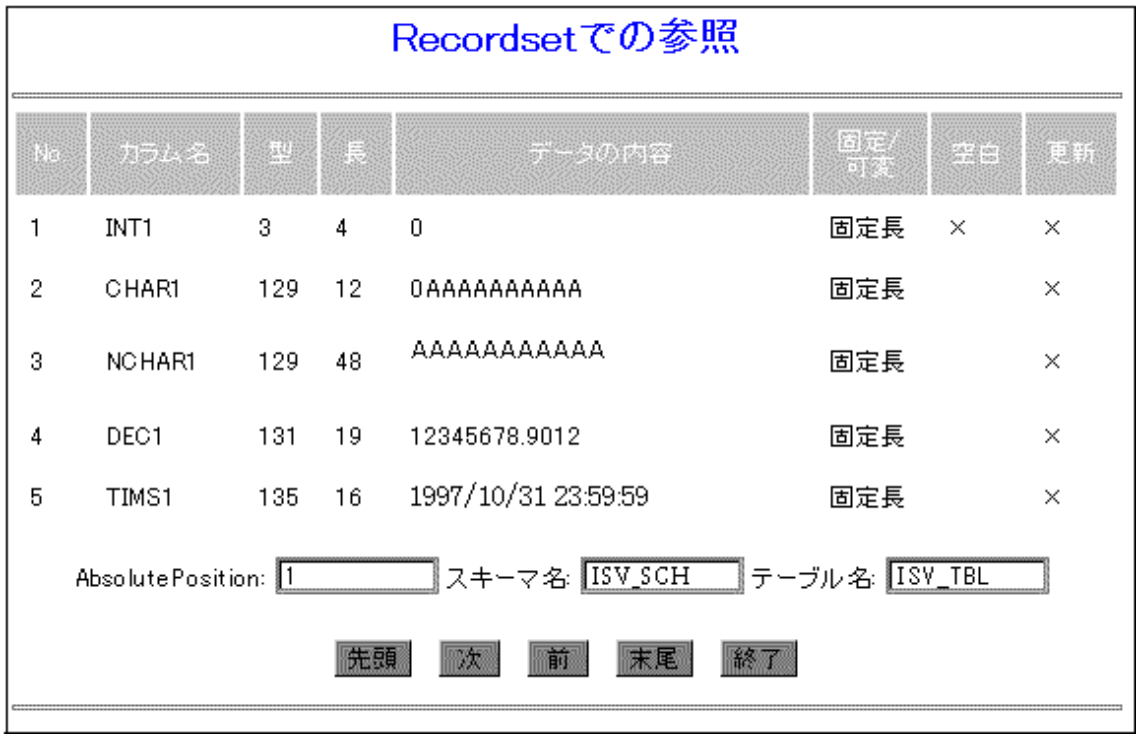
1. 起動画面で[Recordsetでの参照]ボタンをクリックすると下記画面が表示されます。



2. スキーマ名とテーブル名を各項目に入力します。

3. [結果セット生成]ボタンをクリックすると下記画面が表示されます。

   [終了]ボタンをクリックすると、“起動画面”に戻ります。



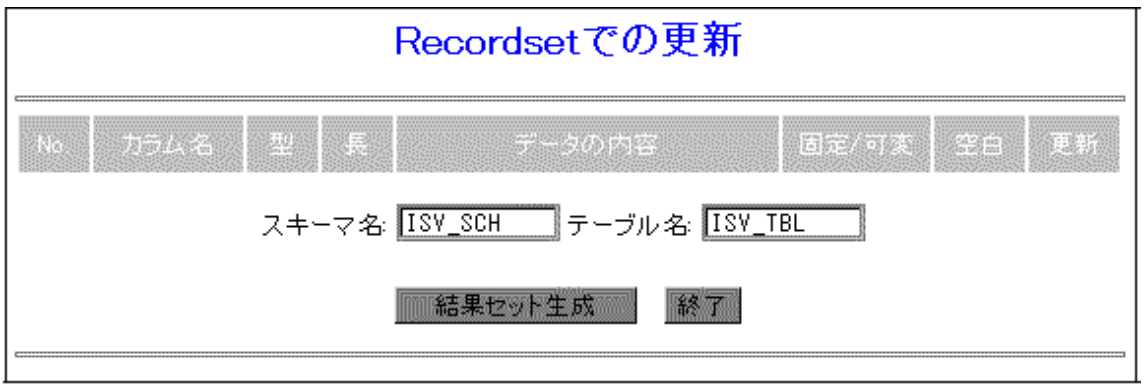
4. [先頭]、[次]、[前]および[末尾]の各ボタンをクリックするとそれぞれカレント行が変更されます。

   [終了]ボタンをクリックすると、“起動画面”に戻ります。

A.7.5 [Recordsetでの更新(ADO)]画面

データ操作プログラムの[Recordsetでの更新]画面の操作方法を説明します。

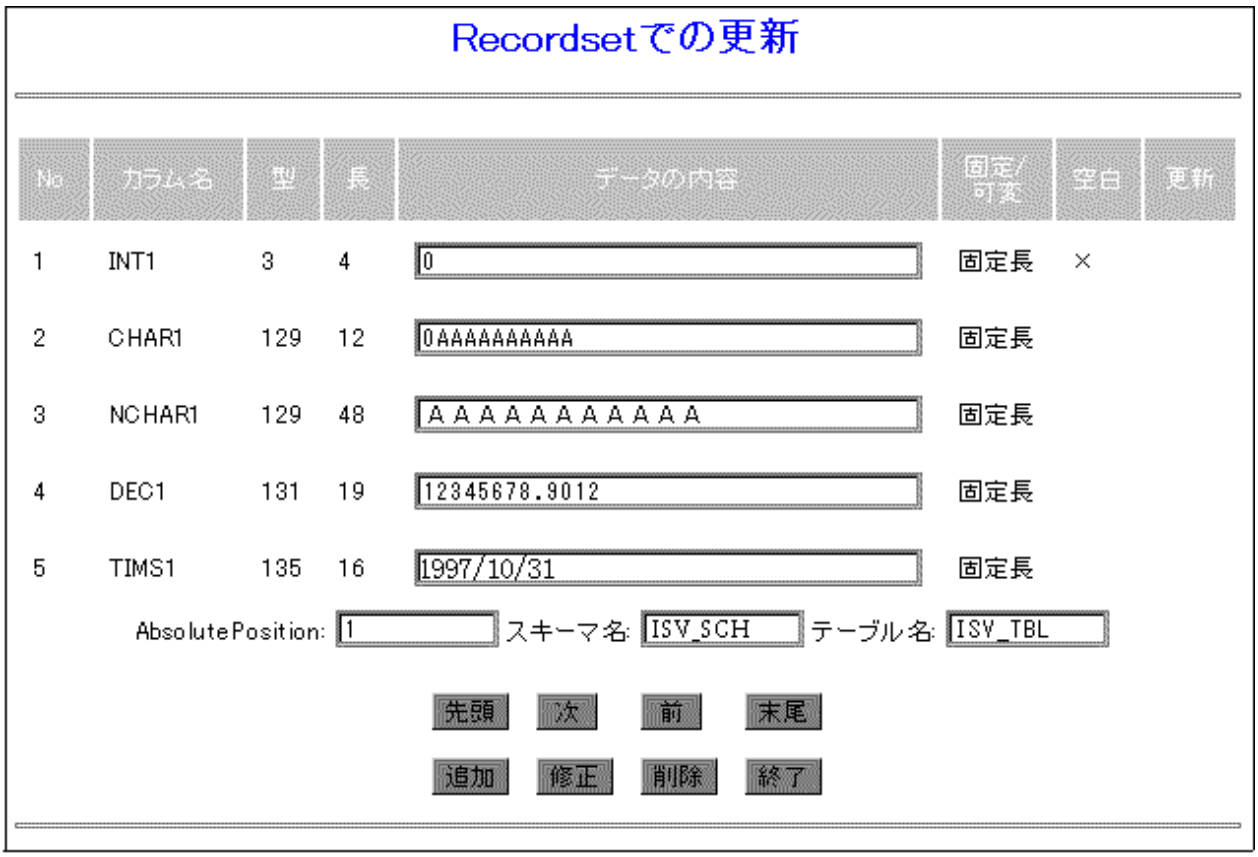
1. 起動画面で[Recordsetでの更新]ボタンをクリックすると下記画面が表示されます。



2. スキーマ名とテーブル名を各項目に入力します。

3. [結果セット生成]ボタンをクリックすると、下記画面が表示されます。

   [終了]ボタンをクリックすると、“起動画面”に戻ります。



4. [先頭]、[次]、[前]および[末尾]の各ボタンをクリックするとそれぞれカレント行が表示されます。

   [データの内容]に値を記述し[追加]ボタンをクリックすると、テーブルの最後にデータが追加されます。

   [データの内容]の値を修正し[修正]ボタンをクリックすると、カレント行のデータが更新されます。

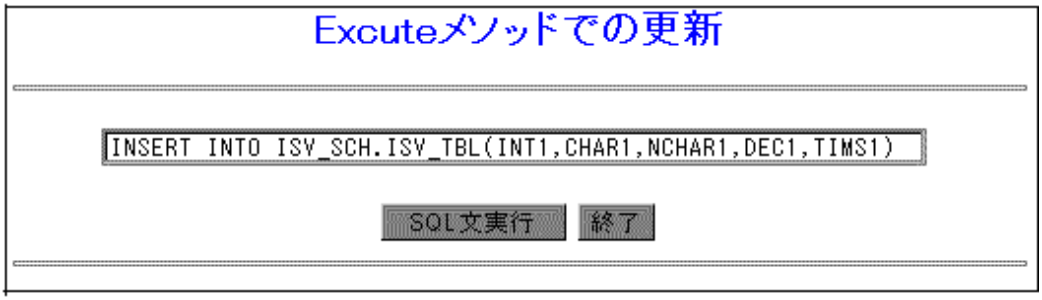
   [削除]ボタンをクリックするとカレント行がテーブルから削除されます。

   [終了]ボタンをクリックすると、“起動画面”に戻ります。

A.7.6 [Executeメソッドでの更新(ADO)]画面

データ操作プログラムの[Executeメソッドでの更新]画面の操作方法を説明します。

1. 起動画面で[Executeメソッドでの更新]ボタンをクリックすると下記画面が表示されます。



2. [SQL文]に実行したいSQL文を記述します。

   [SQL文実行]ボタンをクリックすると記述したSQL文を実行します。

   [終了]ボタンをクリックすると、“起動画面”に戻ります。

付録B　Windows(R)アプリケーションのサンプル

本付録では、Windows(R)アプリケーションのサンプルコードおよび留意事項について説明します。

B.1　サンプル実行前の準備

本付録に記載されているサンプルコードは、以下のデータベース定義環境への接続を想定しています。

参照

データベースの作成についての詳細は、“RDB運用ガイド(データベース定義編)”を参照してください。

また、データベース簡単運用でのデータベース作成については、“データベース簡単運用ガイド”を参照してください。

データベース定義環境

サンプルコードを実行する場合は、ODBCデータソースのデフォルトスキーマ名に以下のスキーマ名を指定してください。

- スキーマ名：SAMPLE1

表の定義

表名：TESTTBL

列名	データ型	列制約定義
KEY_C	INTEGER	NOT NULL PRIMARY KEY
DEC_C	DECIMAL(10,3)	
DAT_C	DATE	
CHA_C	CHAR(250)	

表の定義

表名：IMGTBL

列名	データ型	列制約定義
KEY_C	INTEGER	NOT NULL PRIMARY KEY
BLB_C	BINARY LARGE OBJECT (1M)	

プロシジャルーチンの定義

ルーチン名：COUNTPRC

パラメタモード	パラメタ名	データ型
IN	DAT_P	DATE
OUT	OUT_P	INTEGER
OUT	PRCSTATE	CHAR(5)
OUT	PRCMSG	CHAR(255)
複合文		
DECLARE SQLSTATE CHAR(5); DECLARE SQLMSG CHAR(255); SELECT COUNT(*) INTO OUT_P FROM SAMPLE1.TESTTBL WHERE DAT_C > DAT_P;		

パラメタモード	パラメタ名	データ型

SET PRCSTATE = SQLSTATE;
SET PRCMSG = SQLMSG;

B.2 Visual BasicでのRDOのサンプル

Visual BasicでのRDOのサンプルコードおよび留意事項について説明します。

B.2.1 接続および切断

接続および切断をするサンプルコードについて説明します。

本サンプルコードは、接続後メッセージボックスで接続できたことを表示します。

アプリケーションの手順

1. rdoEnvironmentオブジェクトを生成します。
2. rdoEnvironment.OpenConnectionメソッドでコネクションを接続しrdoConnectionオブジェクトを生成します。
   OpenConnectionメソッドの第1引数にデータソース名を、第4引数に認可識別子・パスワードを指定します。
3. rdoConnection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection

'1. rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'2. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'メッセージボックスの表示
MsgBox "接続できました", vbOKOnly, "Connect"

'3. コネクション切断
Con.Close

'4. オブジェクトの破棄
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.2 データの参照

Visual BasicのRDOとの連携でデータの参照を行う方法について説明します。

本サンプルコードは、取得データをメッセージボックスにて表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.OpenResultsetメソッドでrdoResultsetオブジェクトを生成します。
3. rdoResultsetオブジェクトよりデータを取得します。
   rdoResultset.EOFプロパティでカレント行がEOFかを確認します。
   rdoColumns.Countプロパティで列数を確認します。
   rdoColumn.Valueプロパティでデータを取得します。
   rdoResultset.MoveNextメソッドでカレント行を次の行へと移動します。
4. rdoResultset.CloseメソッドでrdoResultsetオブジェクトを閉じます。
5. コネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Rst As rdoResultset

Dim i As Integer
Dim msgstr As String

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. rdoResultsetオブジェクトの生成
Set Rst = Con.OpenResultset("SELECT * FROM TESTTBL", _
                        rdOpenForwardOnly, _
                        rdConcurReadOnly, _
                        rdExecDirect)

'3. データの取得
'EOFまで繰り返し
Do Until Rst.EOF
    'データ取得文字列の初期化
    msgstr = ""
    '列数の取得
    For i = 0 To Rst.rdoColumns.Count - 1
        'データの取得
        msgstr = msgstr & Rst.rdoColumns(i).Value & " "
    Next
    msgstr = msgstr & Chr(10)
    '行の位置づけ
    Rst.MoveNext
    'メッセージボックスの表示
    MsgBox msgstr, vbOKOnly, "Resultset"
Loop

'4. rdoResultsetオブジェクトを閉じる
Rst.Close
```

```
'5.コネクション切断
Con.Close

'6. オブジェクトを破棄
Set Rst = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.3 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

本サンプルコードは、GetChunkメソッドで16384バイトずつBLOB型データを取得します。取得したBLOB型データはOpenステートメントを使用してファイルに出力します。出力したデータをピクチャボックスコントロールで表示します。事前にフォームへピクチャボックスコントロール(PictureBox1)を追加してください。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.OpenResultsetメソッドでrdoResultsetオブジェクトを生成します。
3. rdoResultsetオブジェクトよりデータを取得します。
   rdoResultset.EOFプロパティでカレント行がEOFかを確認します。
   rdoColumn.ColumnSizeメソッドで取得データのバイト数を確認します。
   rdoColumn.GetChunkメソッドで指定バイト数分データを取得します。
   rdoResultset.MoveNextメソッドでカレント行を次の行へと移動します。
4. rdoResultset.CloseメソッドでrdoResultsetオブジェクトを閉じます。
5. コネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Rst As rdoResultset

Dim DataFile As Integer
Dim Chunks As Integer
Dim Fl As Long
Dim Fragment As Integer
Dim Chunk() As Byte
Const ChunkSize As Integer = 16384
Dim k As Integer
Dim i As Integer

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
```

```
                            False, "UID=USER01;PWD=PASS01;")

'2. rdoResultsetオブジェクトの生成
Set Rst = Con.OpenResultset("SELECT * FROM IMGTBL", _
                        rdOpenForwardOnly, _
                        rdConcurReadOnly, _
                        rdExecDirect)

'行カウントの初期化
i = 1

'3. データの取得
'EOFまで繰り返し
Do Until Rst.EOF

    'BLOB型データのサイズを取得
    Fl = Rst.rdoColumns("BLB_C").ColumnSize

    'BLB_C列にデータがあるか
    If Fl > 0 Then
        'BLOB型データが存在する場合

        '表示しているイメージを消す
        Picture1.Picture = Nothing
        '規定量にデータを分けて取得するため、分割数と残量を求める
        Chunks = Fl ¥ ChunkSize
        Fragment = Fl Mod ChunkSize
        ReDim Chunk(Fragment - 1)
        '残量分のデータ取得
        Chunk() = Rst.rdoColumns("BLB_C").GetChunk(Fragment)

        'FileStreamオブジェクトの生成(ファイルを開く)
        DataFile = FreeFile
        Open "pictemp" For Binary Access Write As DataFile

        'BLOB型データをファイルに出力する
        Put DataFile, , Chunk()

        ReDim Chunk(ChunkSize)
        '分割数分繰り返し
        For k = 1 To Chunks
            '規定量分のデータ取得
            Chunk() = Rst.rdoColumns("BLB_C").GetChunk(ChunkSize)
            'BLOB型データをファイルに出力する
            Put DataFile, , Chunk()
        Next k

        'FileStreamオブジェクトを閉じて破棄する
        Close DataFile
        'Set DataFile = Nothing

        '出力したファイルをPictureBoxコントロールで表示する
        Picture1.AutoSize = True
        Picture1.Picture = LoadPicture("pictemp")
        Kill "pictemp"

    End If
    'メッセージボックスの表示
    MsgBox i & "行目を取得しました", _
                    vbOKOnly, "Message"
    '行の位置づけ
    Rst.MoveNext
    i = i + 1
```

```
Loop

'4. rdoRecordsetオブジェクトを閉じる
Rst.Close

'5. コネクション切断
Con.Close

'6. オブジェクトの破棄
Set Rst = Nothing
Set Con = Nothing
Set Env = Nothing

'表示しているイメージを消す
Picture1.Picture = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.4 データの挿入

Visual BasicのRDOとの連携でデータの更新を行う方法について説明します。

本サンプルコードは、INSERT文実行後、メッセージボックスで完了を知らせます。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.CreateQuery メソッドでrdoQueryオブジェクトを生成します。(SQLプロパティへINSERT文を設定します)
3. rdoConnection.ExecuteメソッドでINSERT文を実行します。(rdExecDirectオプションを指定します)
4. コネクションを切断します。
5. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Qry As rdoQuery

'rdoEnvironmentオブジェクトの生成

Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. rdoQueryオブジェクトの生成
Set Qry = Con.CreateQuery("", _
      "INSERT INTO TESTTBL VALUES(200,1000.025,DATE'2007-04-10','INSERT DATA')")

'3. INSERT文実行
Qry.Execute (rdExecDirect)
```

```
'メッセージボックスの表示
MsgBox "行を挿入しました", vbOKOnly, "Normal End"

'4. コネクション切断
Con.Close

'5. オブジェクトの破棄
Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.5 パラメタマーカを使用したSQL文での更新

パラメタマーカ('?')を使用したSQL文でのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.CreateQueryメソッドでrdoQueryオブジェクトを生成します。(SQLプロパティへUPDATE文を設定します)
3. rdoParameter.Valueプロパティへ値を設定します。
4. rdoQuery.ExecuteメソッドでUPDATE文を実行します。
   rdoQuery.RowsAffectedプロパティで、Executeメソッドで影響を受けた行数を取得します。
5. コネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Qry As rdoQuery

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. rdoQueryオブジェクトの生成
Set Qry = Con.CreateQuery("", _
      "UPDATE TESTTBL SET DEC_C=?,DAT_C=?,CHA_C=? WHERE KEY_C=?")

'3. rdoParameterオブジェクトへ値を設定
Qry.rdoParameters(0).Value = 22222.222
Qry.rdoParameters(1).Value = "2007/04/10"
Qry.rdoParameters(2).Value = "UPDATE DATA"
```

```
Qry.rdoParameters(3).Value = 200

'4. UPDATE文の実行
Qry.Execute

'メッセージボックスの表示
MsgBox Qry.RowsAffected & "行更新されました", _
                vbOKOnly, "Normal End"

'5. コネクション切断
Con.Close

'6. オブジェクトの破棄
Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.6 カーソル位置づけでのデータ更新

カーソル位置づけでのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、rdoResultsetオブジェクトを使用してデータ更新を行います。rdoResultsetオブジェクトに行が1行もない場合には行を挿入し、1行以上ある場合には1行目を更新します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.OpenResultsetメソッドでrdoResultsetオブジェクトを生成します。
3. データを編集します(行がない場合は行を挿入します)。
   rdoResultset.BOFプロパティでカレント行がBOFかを確認します。
   rdoResultset.EOFプロパティでカレント行がEOFかを確認します。
   rdoResultset.AddNewメソッドでrdoResultsetオブジェクトに新しい行を作成します。
   rdoResultset.Editメソッドでカレント行の値を編集できるようにします。
   rdoColumn.Valueプロパティへ値を設定します。
4. rdoResultset.UpdateメソッドでrdoResultsetオブジェクトの編集内容によりINSERT文かUPDATE文を実行します。
5. rdoResultset.CloseメソッドでrdoResultsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Rst As rdoResultset

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
```

```
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. rdoResultsetオブジェクトの生成
Set Rst = Con.OpenResultset("SELECT * FROM TESTTBL", _
                        rdOpenStatic, _
                        rdConcurLock, _
                        rdExecDirect)

'3. データの編集(rdoResultsetに1行もデータがない場合は行を追加する)
If Rst.BOF And Rst.EOF Then
    Rst.AddNew
Else
    Rst.Edit
End If

Rst.rdoColumns("KEY_C").Value = 300
Rst.rdoColumns("DEC_C").Value = 4444.444
Rst.rdoColumns("DAT_C").Value = "2007/04/10"
Rst.rdoColumns("CHA_C").Value = "UPDATE DATA"

'4. INSERT文/UPDATE文の実行
Rst.Update

'メッセージボックスの表示
MsgBox "行を更新しました", vbOKOnly, "Normal End"

'5. rdoResultsetオブジェクトを閉じる
Rst.Close

'6. コネクション切断
Con.Close

'7. オブジェクトの破棄
Set Rst = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.7 BLOB型データの更新

BLOB型データを更新するサンプルコードについて説明します。

本サンプルコードは、Openステートメントを使用してファイルより取得したBLOB型データを、AppendChunkメソッドで16384バイトずつrdoParameterオブジェクトへ追加します。UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

定義長が32Kバイト未満のBLOB型に対するrdoParameterオブジェクトには、AppendChunkメソッドは使用できません。rdoParameter.Valueプロパティにファイルより取得したBLOB型データを設定してください。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.CreateQueryメソッドでrdoQueryオブジェクトを生成します。(SQLプロパティへUPDATE文を設定します)
3. rdoParameter.AppendChunkメソッドでBLOB型データを追加します。

4. rdoQuery.ExecuteメソッドでUPDATE文を実行します。
   rdoQuery.RowsAffectedプロパティで、Executeメソッドで影響を受けた行数を取得します。
5. コネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Qry As rdoQuery

Dim DataFile As Integer
Dim Chunks As Integer
Dim Fl As Long
Dim Fragment As Integer
Dim Chunk() As Byte
Const ChunkSize As Integer = 16384
Dim k As Integer

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. rdoQueryオブジェクトの生成
Set Qry = Con.CreateQuery("", "UPDATE IMGTBL SET BLB_C=? WHERE KEY_C=1")

'ファイルを開く
DataFile = FreeFile
Open "C.BMP" For Binary Access Read As DataFile

'BLOB型データのサイズを取得
Fl = LOF(DataFile)
If Fl = 0 Then Close DataFile: Exit Sub

'規定量にデータを分けて追加するため、分割数と残量を求める
Chunks = Fl ¥ ChunkSize
Fragment = Fl Mod ChunkSize

ReDim Chunk(Fragment - 1)
'BLOB型データをファイルから取得する
Get DataFile, , Chunk()

'3. rdoParameterオブジェクトへBLOB型データを追加する
Qry.rdoParameters(0).AppendChunk (Chunk)

ReDim Chunk(ChunkSize - 1)
'分割数分繰り返し
For k = 1 To Chunks
    'BLOB型データをファイルから取得する
    Get DataFile, , Chunk()
    '３．規定量分のデータ追加
    Qry.rdoParameters(0).AppendChunk (Chunk)
Next k

'4. UPDATE文を実行します
Qry.Execute
```

```
MsgBox Qry.RowsAffected & "行更新されました", _
                vbOKOnly, "Normal End"

'ファイルを閉じる
Close DataFile

'5. コネクション切断
Con.Close

'6. オブジェクトの破棄
Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.8 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.CreateQueryメソッドでrdoQueryオブジェクトを生成します。(SQLプロパティへCALL文を設定します)
3. ストアドプロシジャの入力パラメタのrdoParameter.Valueプロパティへ値を設定します。
4. rdoQuery.ExecuteメソッドでCALL文を実行します。
   ストアドプロシジャの出力パラメタのrdoParameter.Valueプロパティで結果を取得します。
5. コネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection
Dim Qry As rdoQuery

Dim msgstr As String

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc

'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. rdoQueryオブジェクトの生成
Set Qry = Con.CreateQuery("", "CALL COUNTPRC(?,?,?,?)")

'3. 入力パラメタ値の設定
```

```
Qry.rdoParameters(0).Value = "2007/04/10"

'4. CALL文の実行
Qry.Execute

'結果の表示
msgstr = "COUNT: " & Qry.rdoParameters(1).Value & Chr(10)
msgstr = msgstr & "PRCSTATE: " & Qry.rdoParameters(2).Value & Chr(10)
msgstr = msgstr & "PRCMSG: " & Qry.rdoParameters(3).Value
MsgBox msgstr, vbOKOnly, "PROCEDURE DATA"

'5. コネクション切断
Con.Close

'6. オブジェクトの破棄
Set Qry = Nothing
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理

ErrorProc:

'エラー処理ルーチンを記述します
```

B.2.9 トランザクション制御

Visual BasicのRDOとの連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

Connection.BeginTransメソッド、Connection.CommitTransメソッドまたは Connection.RollbackTransメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードと独立性水準の設定または変更は、SET TRANSACTION文で行います。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.2.1 接続および切断”を参照してください。
2. rdoConnection.ExecuteメソッドでSET TRANSACTION文を実行します。
3. rdoConnection.BeginTransメソッドでトランザクションを開始します。
4. データ操作が正常終了した場合、rdoConnection.CommitTransメソッドを実行します。
   データ操作が異常終了した場合、rdoConnection. RollbackTransメソッドを実行します。
5. rdoConnection.Close()メソッドでコネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“B.2.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection

'rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

On Error GoTo ErrorProc
```

```
'1. コネクション接続
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=PASS01;")

'2. SET TRANSACTION文を実行する
Con.Execute "SET TRANSACTION READ WRITE, ISOLATION LEVEL READ COMMITTED", rdExecDirect

'3. トランザクション開始
Con.BeginTrans


'データ操作を行う処理を記述


'4. コミット
Con.CommitTrans

'5. コネクション切断
Con.Close

'6. オブジェクトの破棄
Set Con = Nothing
Set Env = Nothing

Exit Sub

'エラー処理
ErrorProc:

'4. ロールバック
Con.RollbackTrans

'エラー処理ルーチンを記述します
```

B.2.10 コネクションプーリング

プーリング機能は、初期値では無効に設定されています。したがって、プーリング機能を有効にするためにアプリケーションは、コネクションを接続する前にSQLSetEnvAttr関数を発行する必要があります。また、SQLDriverConnect関数を使用して接続を行う場合は、SQL_DRIVER_NOPROMPTパラメタを指定する必要があります。プールされたコネクションは、同一のドライバの間でのみ再利用することができます。

RDOとの連携時にプーリング機能を利用する場合のサンプルプログラムを以下に示します。

```
Option Explicit

Const dbconnstring = "DSN=DSN01;uid=USER01;pwd=PASS01;"  '接続文字列
Const SQL_ATTR_CONNECTION_POOLING = 201
Const SQL_CP_ONE_PER_DRIVER = 1
Const SQL_IS_INTEGER = -6
Const SQL_CP_OFF = 0

Private Declare Function SQLSetEnvAttr Lib "odbc32.dll"( _
                    ByVal EnvironmentHandle As Long, _
                    ByVal EnvAttribute As Long, _
                    ByVal ValuePtr As Long, _
                    ByVal StringLength As Long) As Integer

Private Sub Command1_Click()
     Dim i As Long

     For i = 1 To 10
         Dim en as rdoEnvironment
```

```
        Dim cn As rdoConnection
        Set en = rdoEngine.rdoEnvironments(0)
        Set cn = en.OpenConnection("", rdDriverNoPrompt, _
                                   False, dbconnstring)
        '切断要求されたコネクションは、プールに保存されます。
        cn.Close
        Set cn = Nothing
        Set en = Nothing
    Next
    MsgBox "Connection finished"
End Sub

Private Sub Form_Load()
   Dim rc As Long

   'プーリング機能を有効にします。
   'これは、RDOオブジェクトを呼び出す前に行わなければなりません。
   '1プロセスに付き1回発行する必要があります。
   rc = SQLSetEnvAttr(  0%, _
                        SQL_ATTR_CONNECTION_POOLING, _
                        SQL_CP_ONE_PER_DRIVER, _
                        SQL_IS_INTEGER )
   If rc <> 0 Then
        Debug.Print "SQLSetEnvAttr Error " & rc
   End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
   'プーリング機能を無効にします。
   Call SQLSetEnvAttr( 0&, _
                       SQL_ATTR_CONNECTION_POOLING, _
                       SQL_CP_OFF, _
                       SQL_IS_INTEGER )
End Sub
```

B.2.11 エラー処理

エラー処理を行うサンプルコードについて説明します。

On Errorステートメントで、エラー発生時のエラー処理ルーチンを指定します。RDOを使用してODBCエラーが発生した場合、rdoErrorオブジェクトにSQLSTATEやエラーメッセージなどが格納されます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

ODBC以外のエラーは、Errオブジェクトにエラー情報が格納されます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio.NETドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。

発生したエラーのメッセージとSQLSTATEをメッセージボックスに表示します。

アプリケーションの手順

1. rdoEnvironmentオブジェクトを生成します。
2. On Errorステートメントを使用して、エラー処理ルーチンを設定します。
3. rdoEnvironment.OpenConnectionメソッドでコネクションを接続しrdoConnectionオブジェクトを生成します。(パスワードを間違えて指定しエラーを発生させます)

4. エラー処理ルーチンを記述します。
   rdoErrors.Countプロパティで格納されているrdoErrorオブジェクト数を取得します。
   rdoError.DescriptionプロパティでODBCのエラーメッセージを取得します。
   rdoError.SQLStateプロパティでSQLSTATEを取得します。
   rdoErrors.Clearメソッドで rdoErrorオブジェクトを削除します。
   Err.Descriptionプロパティでエラーメッセージを取得します。
   Err.ClearメソッドでErrオブジェクトを削除します。

```
'オブジェクト宣言
Dim Env As rdoEnvironment
Dim Con As rdoConnection

Dim i As Integer
Dim Err_Count As Integer
Dim msgstr As String

'1. rdoEnvironmentオブジェクトの生成
Set Env = rdoEngine.rdoEnvironments(0)

'2. エラー処理ルーチンの設定
On Error GoTo ErrorProc

'3, コネクション接続(パスワードを間違えて設定：エラー発生)
Set Con = Env.OpenConnection("DSN01", rdDriverNoPrompt, _
                             False, "UID=USER01;PWD=XXXXXX;")

'コネクション切断
Con.Close

'オブジェクトの破棄
Set Con = Nothing
Set Env = Nothing

Exit Sub

'4. エラー処理ルーチン
ErrorProc:
Err_Count = rdoEngine.rdoErrors.Count
If Err_Count > 0 Then
    For i = 0 To Err_Count - 1
        msgstr = msgstr & rdoEngine.rdoErrors(i).Description & Chr(10)
        msgstr = msgstr & "SQLSTATE: " & rdoEngine.rdoErrors(i).SQLState & Chr(10)
    Next
    rdoEngine.rdoErrors.Clear
Else
    msgstr = msgstr & Err.Description
    Err.Clear
End If
MsgBox msgstr, vbOKOnly, "Error"
'オブジェクトの破棄
Set Env = Nothing
```

B.3 Visual BasicでのADOのサンプル

Visual BasicでのADOのサンプルコードおよび留意事項について説明します。

B.3.1 接続および切断

接続または切断をするサンプルコードについて説明します。

本サンプルコードは、接続後メッセージボックスで接続できたことを表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   Connection.ConnectionStringプロパティへ接続文字列を設定します。
2. Connection.Openメソッドでコネクションを接続します。
3. Connection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load



        'オブジェクト宣言
        Dim Con As ADODB.Connection

        '1. Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '2. コネクション接続
        Con.Open

        'メッセージボックスの表示
        MsgBox("接続できました",, "Connect")

        '3. コネクション切断
        Con.Close

        '4. オブジェクトの破棄
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述します
        MsgBox("▲▼  COMMUNICATION OFF LINE  ▲▼",, "No Connect")

    End Sub
End Class
```

B.3.2 データの参照

ADO連携でデータの参照を行う方法について説明します。

本サンプルコードは、取得データをメッセージボックスにて表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。

4. Recordsetオブジェクトよりデータを取得します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します
   Fields.Countプロパティで列数を確認します。
   Field.Valueプロパティでデータを取得します。
   Recordset.MoveNextメソッドでカレント行を次の行へと移動します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

注意

Recordset.EOFプロパティでカレント行がEOFかどうか確認後は、以下のエラーがアプリケーション側で認識できなくなります。

- JYP2099E デッドロックが発生しました.
- JYP5014E スキーマ“@1@”の表“@2@”は占有中です.

これらのエラーをアプリケーション側で認識するタイミングは、Recordset.MoveFirstメソッドを実行するか、データを参照した時です。上記エラーを認識するためには、EOFプロパティを確認せずにMoveFirstメソッドの実行またはデータを参照してください。

ただし、SELECT文の検索結果が0件であった場合、Visual Basicの実行時エラー“3021”が発生します。エラー処理については、“B.3.11 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        Dim i As Integer
        Dim msgstr As String

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"


        On Error GoTo ErrorProc
        '1. コネクション接続
        Con.Open

        '2. Recordsetオブジェクトの生成
        Rst = New ADODB.Recordset

        '3. Recordsetオブジェクトを開く
        Rst.Open("SELECT * FROM TESTTBL", Con,
         ADODB.CursorTypeEnum.adOpenForwardOnly,
         ADODB.LockTypeEnum.adLockReadOnly,
         ADODB.CommandTypeEnum.adCmdText)


        '4. データの取得
        'EOFまで繰り返し
        Do Until Rst.EOF
            'データ取得文字列の初期化
            msgstr = ""
```

```
            '列数の取得
            For i = 0 To Rst.Fields.Count - 1
                'データの取得
                msgstr = msgstr & Rst.Fields(i).Value() & " "
            Next
            'メッセージボックスの表示
            MsgBox(msgstr, vbOKOnly, "Recordset")
            '行の位置づけ
            Rst.MoveNext
        Loop

        '5. Recordsetオブジェクトを閉じる
        Rst.Close

        '6. コネクション切断
        Con.Close

        '7. オブジェクトの破棄
        Rst = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述します
        MsgBox("▲▼  COMMUNICATION OFF LINE  ▲▼",, "No Connect")

    End Sub
End Class
```

B.3.3 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

本サンプルコードは、GetChunkメソッドで16384バイトずつBLOB型データを取得します。取得したBLOB型データはOpenステートメントを使用してファイルに出力します。出力したデータをピクチャボックスコントロールで表示します。事前にフォームへピクチャボックスコントロール(PictureBox1)を追加してください。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. Recordsetオブジェクトよりデータを取得します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Field.ActualSizeプロパティで取得データのバイト数を確認します。
   Field.GetChunkメソッドで指定バイト数分データを取得します 。
   Recordset.MoveNextメソッドでカレント行を次の行へと移動します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection
Dim Rst As Recordset
```

```
Dim DataFile As Integer
Dim Chunks As Integer
Dim Fl As Long
Dim Fragment As Integer
Dim Chunk() As Byte
Const ChunkSize As Integer = 16384
Dim k As Integer
Dim i As Integer


'Connectionオブジェクトの生成と設定
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1. コネクション接続
Con.Open

'2. Recordsetオブジェクトの生成
Set Rst = New Recordset

'3. Recordsetオブジェクトを開く
Rst.Open "SELECT * FROM IMGTBL", Con, _
         adOpenForwardOnly, _
         adLockReadOnly, _
         adCmdText

'行カウントの初期化
i = 1

'4. データの取得
'EOFまで繰り返し
Do Until Rst.EOF

    'BLOB型データのサイズを取得
    Fl = Rst.Fields("BLB_C").ActualSize

    'BLB_C列にデータがあるか
    If Fl > 0 Then
        'BLOB型データが存在する場合

        '表示しているイメージを消す
        Picture1.Picture = Nothing

        '規定量にデータを分けて取得するため、分割数と残量を求める
        Chunks = Fl ¥ ChunkSize
        Fragment = Fl Mod ChunkSize

        '残量分のデータ取得
        ReDim Chunk(Fragment - 1)
        Chunk = Rst.Fields("BLB_C").GetChunk(Fragment)

        'FileStreamオブジェクトの生成(ファイルを開く)
        DataFile = FreeFile
        Open "pictemp" For Binary Access Write As DataFile

        'BLOB型データをファイルに出力する
        Put DataFile, , Chunk()

        ReDim Chunk(ChunkSize - 1)
        '分割数分繰り返し
```

```
        For k = 1 To Chunks
            '規定量分のデータ取得
            Chunk = Rst.Fields("BLB_C").GetChunk(ChunkSize)
            'BLOB型データをファイルに出力する
            Put DataFile, , Chunk()
        Next k

        'FileStreamオブジェクトを閉じる
        Close DataFile

        '出力したファイルをPictureBoxコントロールで表示する
        Picture1.AutoSize = True
        Picture1.Picture = LoadPicture("pictemp")
        Kill "pictemp"

    End If
    'メッセージボックスの表示
    MsgBox i & "行目を取得しました", _
                    vbOKOnly, "Message"
    '行の位置づけ
    Rst.MoveNext
    i = i + 1
Loop

'5. Recordsetオブジェクトを閉じる
Rst.Close

'6. コネクション切断
Con.Close

'7. オブジェクトの破棄
Set Rst = Nothing
Set Con = Nothing

'表示しているイメージを消す
Picture1.Picture = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.3.4 データの挿入

ADO連携でデータの更新を行う方法について説明します。

本サンプルコードは、INSERT文実行後、メッセージボックスで完了を知らせます。

アプリケーションの手順

1. コネクションを接続します。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにINSERT文を設定します。
   Command.CommandTypeプロパティにadCmdTextを設定します。
3. Command.ExecuteメソッドでINSERT文を実行します。
4. コネクションを切断します。
5. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection
Dim Com As Command

'Connectionオブジェクトの生成
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1. コネクション接続
Con.Open

'2. Commandオブジェクトの生成と設定
Set Com = New Command
Com.ActiveConnection = Con
Com.CommandText = "INSERT INTO TESTTBL VALUES(106,1000.025,DATE'2007-04-10','INSERT DATA')"
Com.CommandType = adCmdText

'3. INSERT文実行
Com.Execute

'メッセージボックスの表示
MsgBox "行を挿入しました", vbOKOnly, "Normal End"

'4. コネクション切断
Con.Close

'5. オブジェクトの破棄

Set Com = Nothing
Set Con = Nothing

Exit Sub


'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.3.5 パラメタマーカを使用したSQL文での更新

パラメタマーカ('?')を使用したSQL文でのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにUPDATE文を設定します。
   Command.CommandTypeプロパティにadCmdTextを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、パラメタが入力パラメタ(adParamInput)であることを設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。

Parameter.Valueプロパティには、値を設定します。
Parameter.Precisionプロパティには、精度を設定します。
Parameter.NumericScaleプロパティには、値の小数点以下の桁数を設定します。

4. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。
5. Command.ExecuteメソッドでINSERT文を実行します。
   引数RecordsAffectedを指定しExecuteメソッドで影響を受けた行数を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection
Dim Com As Command
Dim Par1 As Parameter, Par2 As Parameter, Par3 As Parameter, Par4 As Parameter

Dim ra As Long

'Connectionオブジェクトの生成
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1. コネクション接続
Con.Open

'2. Commandオブジェクトの生成と設定
Set Com = New Command
Com.ActiveConnection = Con
Com.CommandText = "UPDATE TESTTBL SET DEC_C=?,DAT_C=?,CHA_C=? WHERE KEY_C=?"
Com.CommandType = adCmdText

'3. Parameterオブジェクトの生成と設定
Set Par1 = Com.CreateParameter("DEC_C", adDecimal, adParamInput, , 2000.222)
Par1.Precision = 10
Par1.NumericScale = 3
Set Par2 = Com.CreateParameter("DAT_C", adDate, adParamInput, , "2007/04/10")
Set Par3 = Com.CreateParameter("CHA_C", adChar, adParamInput, 250, "UPDATE DATA")
Set Par4 = Com.CreateParameter("KEY_C", adInteger, adParamInput, , 100)

'4. ParametersコレクションにParameterオブジェクトを追加
Com.Parameters.Append Par1
Com.Parameters.Append Par2
Com.Parameters.Append Par3
Com.Parameters.Append Par4

'5. UPDATE文実行
Com.Execute ra

'メッセージボックスの表示
MsgBox ra & "行更新されました", vbhOKOnly, "Normal End"

'6. コネクション切断
Con.Close

'7. オブジェクトの破棄
Set Par1 = Nothing
Set Par2 = Nothing
Set Par3 = Nothing
```

```
Set Par4 = Nothing
Set Com = Nothing
Set Con = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.3.6 カーソル位置づけでのデータ更新

カーソル位置づけでのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、Recordsetオブジェクトを使用してデータ更新を行います。Recordsetオブジェクトに行が1行もない場合には行を挿入し、1行以上ある場合には1行目を更新します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. データを編集します。(行がない場合は行を挿入します)
   Recordset.BOFプロパティでカレント行がBOFかを確認します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Recordset.AddNewメソッドでRecordsetオブジェクトに新しい行を作成します。
   Field.Valueプロパティへ値を設定します。
5. Recordset.UpdateメソッドでRecordsetオブジェクトの編集内容によりINSERT文かUPDATE文を実行します。
6. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
7. コネクションを切断します。
8. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection
Dim Rst As Recordset

'Connectionオブジェクトの生成と設定
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1. コネクション接続
Con.Open

'2. Recordsetオブジェクトの生成
Set Rst = New Recordset

'3. Recordsetオブジェクトを開く
Rst.Open "SELECT * FROM TESTTBL", Con, _
         adOpenStatic, _
         adLockPessimistic, _
         adCmdText

'4. データの編集(Recordsetに1行もデータがない場合は行を追加する)
```

```
If Rst.BOF And Rst.EOF Then
    Rst.AddNew
End If

Rst.Fields("KEY_C").Value = 200
Rst.Fields("DEC_C").Value = 11111.111
Rst.Fields("DAT_C").Value = "2007/04/10"
Rst.Fields("CHA_C").Value = "UPDATE DATA"

'5. INSERT文/UPDATE文の実行
Rst.Update

'メッセージボックスの表示
MsgBox "行を更新しました", vbOKOnly, "Normal End"

'6. Recordsetオブジェクトを閉じる
Rst.Close

'7. コネクション切断
Con.Close

'8. オブジェクトの破棄
Set Rst = Nothing
Set Con = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.3.7 BLOB型データの更新

BLOB型データを更新するサンプルコードについて説明します。

本サンプルコードは、Openステートメントを使用してファイルより取得したBLOB型データを、AppendChunkメソッドで16384バイトずつRecordsetオブジェクトへ追加します。UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにUPDATE文を設定します。
   Command.CommandTypeプロパティにadCmdTextを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名を任意で設定します。
   Parameter.Typeプロパティには、adLongVarBinaryを設定します。
   Parameter.Directionプロパティには、パラメタが入力パラメタ(adParamInput)であることを設定します。
   Parameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
4. Parameter.AppendChunkメソッドでBLOB型データを追加します。
5. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。
6. Command.ExecuteメソッドでUPDATE文を実行します。
   引数RecordsAffectedを指定しExecuteメソッドで影響を受けた行数を取得します。
7. コネクションを切断します。

8. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection
Dim Com As Command
Dim Par1 As Parameter

Dim DataFile As Integer
Dim Chunks As Integer
Dim Fl As Long
Dim Fragment As Integer
Dim Chunk() As Byte
Const ChunkSize As Integer = 16384
Dim k As Integer
Dim ra As Long

'Connectionオブジェクトの生成と設定
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1.コネクション接続
Con.Open

'2. Commandオブジェクトの生成と設定
Set Com = New Command
Com.ActiveConnection = Con
Com.CommandText = "UPDATE IMGTBL SET BLB_C=? WHERE KEY_C=1"
Com.CommandType = adCmdText

'3. Parameterオブジェクトの生成と設定
Set Par1 = Com.CreateParameter("BLB_C", adLongVarBinary, adParamInput)

'ファイルを開く
DataFile = FreeFile
Open "C.BMP" For Binary Access Read As DataFile

'BLOB型データのサイズを取得
Fl = LOF(DataFile)
If Fl = 0 Then Close DataFile: Exit Sub
Par1.Size = Fl

'規定量にデータを分けて追加するため、分割数と残量を求める
Chunks = Fl ¥ ChunkSize
Fragment = Fl Mod ChunkSize

ReDim Chunk(Fragment - 1)
'BLOB型データをファイルから取得する
Get DataFile, , Chunk()

'4. Parameterオブジェクトへ BLOB型データを追加する
Par1.AppendChunk (Chunk)

ReDim Chunk(ChunkSize - 1)
'分割数分繰り返し
For k = 1 To Chunks
    'BLOB型データをファイルから取得する
    Get DataFile, , Chunk()
    '４．規定量分のデータ追加
    Par1.AppendChunk (Chunk)
Next k
```

```
'5. ParametersコレクションにParameterオブジェクトを追加
Com.Parameters.Append Par1

'6. UPDATE文を実行します
Com.Execute ra

'メッセージボックスの表示
MsgBox ra & "行更新されました", vbOKOnly, "Normal End"

'ファイルを閉じる
Close DataFile

'7. コネクション切断
Con.Close

'8. オブジェクトの破棄
Set Par1 = Nothing
Set Com = Nothing
Set Con = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.3.8 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにストアドプロシジャ名を設定します。
   Command.CommandTypeプロパティにadCmdStoredProcを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、入力パラメタ(adParamInput)、出力パラメタ(adParamOutput)のいずれかであることを設定します。
   入力パラメタのParameter.Valueプロパティには値を設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。
5. Command.Executeメソッドでストアドプロシジャを実行します。
   ストアドプロシジャの結果を、出力パラメタのParameter.Valueプロパティで結果を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection
Dim Com As Command
Dim Par1 As Parameter, Par2 As Parameter, Par3 As Parameter, Par4 As Parameter

Dim msgstr As String

'Connectionオブジェクトの生成と設定
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1. コネクション接続
Con.Open

'2. Commandオブジェクトの生成と設定
Set Com = New Command
Com.ActiveConnection = Con
Com.CommandText = "COUNTPRC"
Com.CommandType = adCmdStoredProc

'3. Parameteオブジェクトの生成と設定
Set Par1 = Com.CreateParameter("DAT_P", adDate, adParamInput, , "2007/04/10")
Set Par2 = Com.CreateParameter("OUT_P", adInteger, adParamOutput)
Set Par3 = Com.CreateParameter("PRCSTATE", adChar, adParamOutput, 5)
Set Par4 = Com.CreateParameter("PRCMSG", adChar, adParamOutput, 255)

'4. ParametersコレクションにParameterオブジェクトを追加
Com.Parameters.Append Par1
Com.Parameters.Append Par2
Com.Parameters.Append Par3
Com.Parameters.Append Par4

'5. ストアドプロシジャの実行
Com.Execute

'メッセージボックスで結果を表示
msgstr = "COUNT: " & Com.Parameters("OUT_P").Value & Chr(10)
msgstr = msgstr & "PRCSTATE: " & Com.Parameters("PRCSTATE").Value & Chr(10)
msgstr = msgstr & "PRCMSG: " & Com.Parameters("PRCMSG").Value
MsgBox msgstr, vbOKOnly, "PROCEDURE DATA"

'6. コネクション切断
Con.Close

'7. オブジェクトの破棄
Set Par1 = Nothing
Set Par2 = Nothing
Set Par3 = Nothing
Set Par4 = Nothing
Set Com = Nothing
Set Con = Nothing

Exit Sub

'エラー処理
ErrorProc:

'エラー処理ルーチンを記述します
```

B.3.9 トランザクション制御

ADO連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

Connection.BeginTransメソッド、Connection.CommitTransメソッドまたは Connection.RollbackTransメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードの設定または変更は、SET TRANSACTION文または Connection.Modeプロパティで行います。Connection.Modeプロパティはコネクションが切れている時だけ設定または変更することができます。コネクション接続中に変更が可能なSET TRANSACTION文による設定または変更を推奨します。

独立性水準の設定または変更は、Connection.IsolationLevelプロパティまたはSET TRANSACTION文で行います。SET TRANSACTION文で設定または変更を行う場合は、Connection.BeginTransメソッド実行時にConnection.IsolationLevelプロパティ設定値に変更されるため、Connection.BeginTransメソッド実行後に行う必要があります。このため、Connection.IsolationLevelプロパティによる設定または変更を推奨します。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.3.1 接続および切断”を参照してください。
2. Connection.ExecuteメソッドでSET TRANSACTION文を実行しアクセスモードを設定します。
3. Connection.IsolationLevelプロパティ独立性水準(adXactReadCommitted)を設定します。
4. Connection.BeginTransメソッドでトランザクションを開始します。
5. データ操作が正常終了した場合、Connection.CommitTransメソッドを実行します。
   データ操作が異常終了した場合、Connection.RollbackTransメソッドを実行します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.3.11 エラー処理”を参照してください。

```
'オブジェクト宣言
Dim Con As Connection

'Connectionオブジェクトの生成と設定
Set Con = New Connection
Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

On Error GoTo ErrorProc

'1. コネクション接続
Con.Open

'2. アクセスモードの設定
Con.Execute "SET TRANSACTION READ WRITE", , adCmdText

'3. IsolationLevelの設定
Con.IsolationLevel = adXactReadCommitted

'4. トランザクション開始
Con.BeginTrans

'データ操作を行う処理を記述

'5. コミット
Con.CommitTrans

'6. コネクション切断
```

```
Con.Close

'7. オブジェクトの破棄
Set Con = Nothing

Exit Sub

'エラー処理
ErrorProc:

'5. ロールバック
Con.RollbackTrans

'エラー処理ルーチンを記述
```

B.3.10 コネクションプーリング

プーリング機能は、初期値では無効に設定されています。したがって、プーリング機能を有効にするためにアプリケーションは、コネクションを接続する前にSQLSetEnvAttr関数を発行する必要があります。また、SQLDriverConnect関数を使用して接続を行う場合は、SQL_DRIVER_NOPROMPTパラメタを指定する必要があります。さらに、ODBCドライバマネージャのプーリング機能を利用するパラメタを接続文字列に記載する必要があります。

プールされたコネクションは、同一のドライバの間でのみ再利用することができます。

ADOとの連携時にプーリング機能を利用する場合のサンプルプログラムを以下に示します。

```
Option Explicit

Const dbconnstring = "DSN=DSN01;uid=USER01;pwd=PASS01;OLE DB Services=-2" '接続文字列
Const SQL_ATTR_CONNECTION_POOLING = 201
Const SQL_CP_ONE_PER_DRIVER = 1
Const SQL_IS_INTEGER = -6
Const SQL_CP_OFF = 0

Private Declare Function SQLSetEnvAttr Lib "odbc32.dll"( _
                    ByVal EnvironmentHandle As Long, _
                    ByVal EnvAttribute As Long, _
                    ByVal ValuePtr As Long, _
                    ByVal StringLength As Long) As Integer

Private Sub Command1_Click()
     Dim i As Long

     For i = 1 To 10
         Dim cn As ADODB.Connection
         Set cn = New ADODB.Connection
         cn.Open dbconnstring
         '切断要求されたコネクションは、プールに保存されます。
         cn.Close
         Set cn = Nothing
     Next
     MsgBox "Connection finished"
End Sub

Private Sub Form_Load()
   Dim rc As Long

   'プーリング機能を有効にします。
   'これは、ADOオブジェクトを呼び出す前に行わなければなりません。
   '1プロセスに付き1回発行する必要があります。
   rc = SQLSetEnvAttr(  0%, _
                        SQL_ATTR_CONNECTION_POOLING, _
                        SQL_CP_ONE_PER_DRIVER, _
```

```
                        SQL_IS_INTEGER )
   If rc <> 0 Then
        Debug.Print "SQLSetEnvAttr Error " & rc
   End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
   'プーリング機能を無効にします。
   Call SQLSetEnvAttr( 0&, _
                       SQL_ATTR_CONNECTION_POOLING, _
                       SQL_CP_OFF, _
                       SQL_IS_INTEGER )
End Sub
```

B.3.11 エラー処理

エラー処理を行うサンプルコードについて説明します。

On Errorステートメントで、エラー発生時のエラー処理ルーチンを指定します。ADOを使用してODBCエラーが発生した場合、ErrorオブジェクトにSQLSTATEやエラーメッセージなどが格納されます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

ODBC以外のエラーは、Errオブジェクトにエラー情報が格納されます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio .NETドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。

発生したエラーのメッセージとSQLSTATEをメッセージボックスに表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   (Connection. ConnectionStringプロパティに接続文字列を間違えて設定します。)
2. On Errorステートメントを使用して、エラー処理ルーチンを設定します。
3. Connection.Openメソッドでコネクションを接続します。
   (パスワードを間違えて指定しエラーを発生させます)
4. エラー処理ルーチンを記述します。
   Errors.Countプロパティで格納されているErrorオブジェクト数を取得します。
   Error.DescriptionプロパティでODBCのエラーメッセージを取得します。
   Error.SQLStateプロパティでSQLSTATEを取得します。
   Error.Clearメソッドでオブジェクトを削除します。
   Err.Descriptionプロパティでエラーメッセージを取得します。
   Err.Clearメソッドでオブジェクトを削除します。

```
'オブジェクト宣言
Dim Con As Connection

Dim i As Integer
Dim Err_Count As Integer
Dim msgstr As String

'1. Connectionオブジェクトの生成と設定(接続文字列を間違えて設定)
Set Con = New Connection
Con.ConnectionString = " DSN=DSN01;UID=USER01;PWD=XXXXXX;"
```

```
'2. エラー処理ルーチンの設定
On Error GoTo ErrorProc

'3. コネクション接続(エラー発生)
Con.Open

'コネクション切断
Con.Close

'オブジェクトの破棄
Set Con = Nothing

Exit Sub

'4. エラー処理ルーチン
ErrorProc:
Err_Count = Con.Errors.Count
If Err_Count > 0 Then
    For i = 0 To Err_Count - 1
        msgstr = msgstr & Con.Errors(i).Description & Chr(10)
        msgstr = msgstr & "SQLSTATE: " & Con.Errors(i).SQLState & Chr(10)
    Next
    Con.Errors.Clear
Else
    msgstr = msgstr & Err.Description
    Err.Clear
End If
MsgBox msgstr, vbOKOnly, "Error"
'オブジェクトの破棄
Set Con = Nothing
```

B.4 Visual Basic .NETでのADOのサンプル

Visual Basic .NETでのADOのサンプルコードおよび留意事項について説明します。

なお、このサンプルコードを実行する場合は、Visual Studio 2017を使用してください。

B.4.1 接続および切断

接続および切断をするサンプルコードについて説明します。

本サンプルコードは、接続後メッセージボックスで接続できたことを表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   Connection.ConnectionStringプロパティへ接続文字列を設定します。
2. Connection.Openメソッドでコネクションを接続します。
3. Connection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection

        '1. ADODB.Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
```

```
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '2. コネクション接続

        Con.Open()

        'メッセージボックスの表示
        MessageBox.Show("接続できました", "Connect",
                MessageBoxButtons.OK, MessageBoxIcon.None)

        '3. コネクション切断
        Con.Close()

        '4. オブジェクトの破棄
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
                MessageBoxButtons.OK, MessageBoxIcon.None)


    End Sub
End Class
```

B.4.2 データの参照

ADO連携でデータの参照を行う方法について説明します。

本サンプルコードは、取得データをメッセージボックスにて表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. Recordsetオブジェクトよりデータを取得します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Fields.Countプロパティで列数を確認します。
   Field.Valueプロパティでデータを取得します。
   Recordset.MoveNextメソッドでカレント行を次の行へと移動します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

注意

Recordset.EOFプロパティでカレント行がEOFかどうか確認後は、以下のエラーがアプリケーション側で認識できなくなります。

- JYP2099E デッドロックが発生しました.

・ JYP5014Eスキーマ“@1@”の表“@2@”は占有中です.

これらのエラーをアプリケーション側で認識するタイミングは、Recordset.MoveFirstメソッドを実行するか、データを参照した時です。上記エラーを認識するためには、EOFプロパティを確認せずにMoveFirstメソッドの実行またはデータを参照してください。

ただし、SELECT文の検索結果が0件であった場合、Visual Basicの実行時エラー“3021”が発生します。エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        Dim i As Integer
        Dim msgstr As String

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Recordsetオブジェクトの生成
        Rst = New ADODB.Recordset()

        '3. Recordsetオブジェクトを開く
        Rst.Open("SELECT * FROM TESTTBL", Con,
         ADODB.CursorTypeEnum.adOpenForwardOnly,
         ADODB.LockTypeEnum.adLockReadOnly,
         ADODB.CommandTypeEnum.adCmdText)

        '4. データの取得
        'EOFまで繰り返し
        Do Until Rst.EOF
            'データ取得文字列の初期化
            msgstr = ""
            '列数の取得
            For i = 0 To Rst.Fields.Count - 1
                'データの取得
                msgstr &= Rst.Fields(i).Value() & " "
            Next
            'メッセージボックスの表示
            MessageBox.Show(msgstr, "Recordset", MessageBoxButtons.OK,
                                    MessageBoxIcon.None)
            '行の位置づけ
            Rst.MoveNext()
        Loop

        '5. Recordsetオブジェクトを閉じる
        Rst.Close()

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Rst = Nothing
        Con = Nothing
```

```
        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)


    End Sub
End Class
```

B.4.3 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

本サンプルコードは、GetChunkメソッドで16384バイトずつBLOB型データを取得します。取得したBLOB型データはFileStreamオブジェクトを使用してファイルに出力します。出力したデータをピクチャボックスコントロールで表示します。事前にフォームへピクチャボックスコントロール(PictureBox1)を追加してください。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. Recordsetオブジェクトよりデータを取得します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Field.ActualSizeプロパティで取得データのバイト数を確認します。
   Field.GetChunkメソッドで指定バイト数分データを取得します。
   Recordset.MoveNextメソッドでカレント行を次の行へと移動します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        Dim MyImage As Bitmap
        Dim fs As System.IO.FileStream
        Dim Chunks As Short
        Dim Fl As Integer
        Dim Fragment As Short
        Dim Chunk() As Byte
        Const ChunkSize As Short = 16384
        Dim k As Integer
        Dim i As Integer

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc
```

```
'1. コネクション接続
Con.Open()

'2. Recordsetオブジェクトの生成
Rst = New ADODB.Recordset()

'3. Recordsetオブジェクトを開く
Rst.Open("SELECT * FROM IMGTBL", Con,
 ADODB.CursorTypeEnum.adOpenForwardOnly,
 ADODB.LockTypeEnum.adLockReadOnly,
 ADODB.CommandTypeEnum.adCmdText)

'行カウントの初期化
i = 1

'4. データの取得
'EOFまで繰り返し
Do Until Rst.EOF

    'BLOB型データのサイズを取得
    Fl = Rst.Fields("BLB_C").ActualSize

    'BLB_C列にデータがあるか
    If Fl > 0 Then
        'BLOB型データが存在する場合

        'イメージオブジェクトの破棄
        If Not (MyImage Is Nothing) Then
            MyImage.Dispose()
            MyImage = Nothing
            PictureBox1.Image = Nothing
        End If

        '規定量にデータを分けて取得するため、分割数と残量を求める
        Chunks = Fl ¥ ChunkSize
        Fragment = Fl Mod ChunkSize

        '残量分のデータ取得
        ReDim Chunk(Fragment - 1)
        Chunk = Rst.Fields("BLB_C").GetChunk(Fragment)

        'FileStreamオブジェクトの生成(ファイルを開く)
        fs = New System.IO.FileStream _
("C.BMP", System.IO.FileMode.OpenOrCreate,
  System.IO.FileAccess.Write)

        'BLOB型データをファイルに出力する
        fs.Write(Chunk, 0, Fragment)

        ReDim Chunk(ChunkSize - 1)
        '分割数分繰り返し
        For k = 1 To Chunks
            '規定量分のデータ取得
            Chunk = Rst.Fields("BLB_C").GetChunk(ChunkSize)
            'BLOB型データをファイルに出力する
            fs.Write(Chunk, 0, ChunkSize)
        Next k

        'FileStreamオブジェクトを閉じて破棄する
        fs.Close()
        fs = Nothing
        'Bitmapオブジェクトの生成
```

```
                MyImage = New Bitmap("C.BMP")
                '出力したファイルをPictureBoxコントロールで表示する
                PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
                PictureBox1.Image = CType(MyImage, Image)
                PictureBox1.ImageLocation = "C.BMP"
                PictureBox1.Refresh()
            End If
            'メッセージボックスの表示
            MessageBox.Show(i & "行目を取得しました",
              "Message", MessageBoxButtons.OK, MessageBoxIcon.None)
            '行の位置づけ
            Rst.MoveNext()
            i += 1
        Loop

        '5. Recordsetオブジェクトを閉じる
        Rst.Close()

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Rst = Nothing
        Con = Nothing
        'イメージオブジェクトの破棄
        If Not (MyImage Is Nothing) Then
            MyImage.Dispose()
            MyImage = Nothing
            PictureBox1.Image = Nothing
        End If


        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.4 データの挿入

ADO連携でデータの挿入を行う方法について説明します。

本サンプルコードは、INSERT文実行後、メッセージボックスで完了を知らせます。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにINSERT文を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdTextを設定します。
3. Command.ExecuteメソッドでINSERT文を実行します。
4. コネクションを切断します。
5. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command

        'Connectionオブジェクトの生成
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Commandオブジェクトの生成と設定
        Com = New ADODB.Command()
        Com.ActiveConnection = Con
        Com.CommandText = "INSERT INTO TESTTBL VALUES(300,1000.025,DATE'2100-08-06','INSERT DATA')"
        Com.CommandType = ADODB.CommandTypeEnum.adCmdText

        '3. INSERT文実行
        Com.Execute()

        'メッセージボックスの表示
        MessageBox.Show("行を挿入しました", "Normal End",
         MessageBoxButtons.OK, MessageBoxIcon.None)

        '4. コネクション切断
        Con.Close()

        '5. オブジェクトの破棄
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.5 パラメタマーカを使用したSQL文での更新

パラメタマーカ('?')を使用したSQL文でのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。

2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにUPDATE文を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdTextを設定します。

3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、パラメタが入力パラメタ(adParamInput)であることを設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
   Parameter.Valueプロパティには、値を設定します。
   Parameter.Precisionプロパティには、精度を設定します。
   Parameter.NumericScaleプロパティには、値の小数点以下の桁数を設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。
5. Command.ExecuteメソッドでINSERT文を実行します。
   引数RecordsAffectedを指定しExecuteメソッドで影響を受けた行数を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command
        Dim Par1, Par2, Par3, Par4 As ADODB.Parameter

        Dim ra As Long

        'Connectionオブジェクトの生成
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"
        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Commandオブジェクトの生成と設定
        Com = New ADODB.Command()
        Com.ActiveConnection = Con
        Com.CommandText = "UPDATE TESTTBL SET DEC_C=?,DAT_C=?,CHA_C=? WHERE KEY_C=?"
        Com.CommandType = ADODB.CommandTypeEnum.adCmdText

        '3. Parameterオブジェクトの生成と設定
        Par1 = Com.CreateParameter("DEC_C", ADODB.DataTypeEnum.adDecimal,
                    ADODB.ParameterDirectionEnum.adParamInput, , 2000.222)
        Par1.Precision = 10
        Par1.NumericScale = 3
        Par2 = Com.CreateParameter("DAT_C", ADODB.DataTypeEnum.adDate,
                    ADODB.ParameterDirectionEnum.adParamInput, , "2018-08-07")
        Par3 = Com.CreateParameter("CHA_C", ADODB.DataTypeEnum.adChar,
                    ADODB.ParameterDirectionEnum.adParamInput, 250, "UPDATE DATA")
        Par4 = Com.CreateParameter("KEY_C", ADODB.DataTypeEnum.adInteger,
                    ADODB.ParameterDirectionEnum.adParamInput, , 300)

        '4. ParametersコレクションにParameterオブジェクトを追加
        Com.Parameters.Append(Par1)
        Com.Parameters.Append(Par2)
        Com.Parameters.Append(Par3)
        Com.Parameters.Append(Par4)

        '5. UPDATE文実行
```

```
        Com.Execute(ra)

        'メッセージボックスの表示
        MessageBox.Show(ra & "行更新されました", "Normal End",
         MessageBoxButtons.OK, MessageBoxIcon.None)

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Par1 = Nothing
        Par2 = Nothing
        Par3 = Nothing
        Par4 = Nothing
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.6 カーソル位置づけでのデータ更新

カーソル位置づけでのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、Recordsetオブジェクトを使用してデータ更新を行います。Recordsetオブジェクトに行が1行もない場合には行を挿入し、1行以上ある場合には1行目を更新します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. データを編集します。(行がない場合は行を挿入します)
   Recordset.BOFプロパティでカレント行がBOFかを確認します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Recordset.AddNewメソッドでRecordsetオブジェクトに新しい行を作成します。
   Field.Valueプロパティへ値を設定します。
5. Recordset.UpdateメソッドでRecordsetオブジェクトの編集内容によりINSERT文かUPDATE文を実行します。
6. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
7. コネクションを切断します。
8. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        'オブジェクト宣言
```

```
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Recordsetオブジェクトの生成
        Rst = New ADODB.Recordset()

        '3. Recordsetオブジェクトを開く
        Rst.Open("SELECT * FROM TESTTBL", Con,
         ADODB.CursorTypeEnum.adOpenStatic,
         ADODB.LockTypeEnum.adLockPessimistic,
         ADODB.CommandTypeEnum.adCmdText)

        '4. データの編集(Resultsetに1行もデータがない場合は行を追加する)
        If Rst.BOF And Rst.EOF Then
            Rst.AddNew()
        End If

        Rst.Fields("KEY_C").Value = 301
        Rst.Fields("DEC_C").Value = 11111.111
        Rst.Fields("DAT_C").Value = "2018-08-07"
        Rst.Fields("CHA_C").Value = "UPDATE DATA2"

        '5. INSERT文/UPDATE文の実行
        Rst.Update()

        'メッセージボックスの表示
        MessageBox.Show("行を更新しました",
         "Normal End", MessageBoxButtons.OK, MessageBoxIcon.None)

        '6. Recordsetオブジェクトを閉じる
        Rst.Close()

        '7. コネクション切断
        Con.Close()

        '8. オブジェクトの破棄
        Rst = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.7 BLOB型データの更新

BLOB型データを更新するサンプルコードについて説明します。

本サンプルコードは、FileStreamオブジェクトを使用してファイルより取得したBLOB型データを、AppendChunkメソッドで16384バイトずつRecordsetオブジェクトへ追加します。UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、"B.4.1 接続および切断"を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにUPDATE文を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdTextを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名を任意で設定します。
   Parameter.Typeプロパティには、DataTypeEnum.adLongVarBinaryを設定します。
   Parameter.Directionプロパティには、パラメタが入力パラメタ(adParamInput)であることを設定します。
   Parameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
4. Parameter.AppendChunkメソッドでBLOB型データを追加します。
5. ParametersコレクションにParameterオブジェクトを追加します。
6. Command.ExecuteメソッドでUPDATE文を実行します。
   引数RecordsAffectedを指定しExecuteメソッドで影響を受けた行数を取得します。
7. コネクションを切断します。
8. オブジェクトを破棄します。

エラー処理については、"B.4.10 エラー処理"を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command
        Dim Par1 As ADODB.Parameter

        Dim Chunks As Short
        Dim fs As System.IO.FileStream
        Dim Fl As Integer
        Dim Fragment As Short
        Dim Chunk() As Byte
        Const ChunkSize As Short = 16384
        Dim k As Integer
        Dim ra As Long

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1.コネクション接続
        Con.Open()

        '2, Commandオブジェクトの生成と設定
        Com = New ADODB.Command()
        Com.ActiveConnection = Con
        Com.CommandText = "UPDATE IMGTBL SET CLB_C=? WHERE KEY_C=1"
```

```
        Com.CommandType = ADODB.CommandTypeEnum.adCmdText

        '3. Parameterオブジェクトの生成と設定
        Par1 = Com.CreateParameter("CLB_C", ADODB.DataTypeEnum.adLongVarChar,
ADODB.ParameterDirectionEnum.adParamInput)

        'ファイルを開く
        fs = New System.IO.FileStream _
 ("C:¥yamasaki¥CLOB.TXT", System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.Read)

        'CLOB型データのサイズを取得
        Fl = fs.Length
        Par1.Size = Fl

        '規定量にデータを分けて追加するため、分割数と残量を求める
        Chunks = Fl ¥ ChunkSize
        Fragment = Fl Mod ChunkSize

        ReDim Chunk(Fragment - 1)
        'CLOB型データをファイルから取得する
        fs.Read(Chunk, 0, Fragment)

        '4. ParameterオブジェクトへCLOB型データを追加する
        Par1.AppendChunk(Chunk)

        ReDim Chunk(ChunkSize - 1)
        '分割数分繰り返し
        For k = 1 To Chunks
            'CLOB型データをファイルから取得する
            fs.Read(Chunk, 0, ChunkSize)
            '規定量分のデータ追加
            Par1.AppendChunk(Chunk)
        Next k

        '5. ParametersコレクションにParameterオブジェクトを追加
        Com.Parameters.Append(Par1)

        '6. UPDATE文を実行します
        Com.Execute(ra)

        'メッセージボックスの表示
        MessageBox.Show(ra & "行更新されました", "Normal End",
         MessageBoxButtons.OK, MessageBoxIcon.None)

        'ファイルを閉じる
        fs.Close()
        fs = Nothing

        '7. コネクション切断
        Con.Close()

        '8. オブジェクトの破棄
        Par1 = Nothing
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
```

```
        MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.8 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をメッセージボックスで表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにストアドプロシジャ名を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum. adCmdStoredProcを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、入力パラメタ“adParamInput”、出力パラメタ“adParamOutput”のいずれかであることを設定します。
   入力パラメタのParameter.Valueプロパティには値を設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。
5. Command . Executeメソッドでストアドプロシジャを実行します。
   ストアドプロシジャの結果を、出力パラメタのParameter.Valueプロパティで結果を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command
        Dim Par1, Par2, Par3, Par4 As ADODB.Parameter

        Dim msgstr As String

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Commandオブジェクトの生成と設定
        Com = New ADODB.Command()
        Com.ActiveConnection = Con
        Com.CommandText = "COUNTPRC"
        Com.CommandType = ADODB.CommandTypeEnum.adCmdStoredProc
        '3. rametеオブジェクトの生成と設定
```

```
        Par1 = Com.CreateParameter("DAT_P", ADODB.DataTypeEnum.adDate,
         ADODB.ParameterDirectionEnum.adParamInput, , "2018-08-07")
        Par2 = Com.CreateParameter("OUT_P", ADODB.DataTypeEnum.adInteger,
         ADODB.ParameterDirectionEnum.adParamOutput)
        Par3 = Com.CreateParameter("PRCSTATE", ADODB.DataTypeEnum.adChar,
         ADODB.ParameterDirectionEnum.adParamOutput, 5)
        Par4 = Com.CreateParameter("PRCMSG", ADODB.DataTypeEnum.adChar,
         ADODB.ParameterDirectionEnum.adParamOutput, 255)

        '4. rametersコレクションにParameterオブジェクトを追加
        Com.Parameters.Append(Par1)
        Com.Parameters.Append(Par2)
        Com.Parameters.Append(Par3)
        Com.Parameters.Append(Par4)

        '5. ストアドプロシジャの実行
        Com.Execute()

        'メッセージボックスで結果を表示
        msgstr = "COUNT: " & Com.Parameters("OUT_P").Value & ControlChars.Cr
        msgstr &= "PRCSTATE: " & Com.Parameters("PRCSTATE").Value & ControlChars.Cr
        msgstr &= "PRCMSG: " & Com.Parameters("PRCMSG").Value
        MessageBox.Show(msgstr, "PROCEDURE DATA",
          MessageBoxButtons.OK, MessageBoxIcon.None)

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Par1 = Nothing
        Par2 = Nothing
        Par3 = Nothing
        Par4 = Nothing
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.9 トランザクション制御

ADO連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

Connection.BeginTransメソッド、Connection.CommitTransメソッドまたは Connection.RollbackTransメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードの設定または変更は、SET TRANSACTION 文または Connection.Modeプロパティで行います。Connection.Modeプロパティはコネクションが切れている時だけ設定または変更することができます。コネクション接続中に変更が可能なSET TRANSACTION文による設定または変更を推奨します。

独立性水準の設定または変更は、Connection.IsolationLevelプロパティまたは SET TRANSACTION文で行います。SET TRANSACTION文で設定または変更を行う場合は、Connection.BeginTransメソッド実行時にConnection.IsolationLevelプロパティ設定値に変更されるため、Connection.BeginTransメソッド実行後に行う必要があります。このため、Connection.IsolationLevelプロパティによる設定または変更を推奨します。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.4.1 接続および切断”を参照してください。
2. Connection.ExecuteメソッドでSET TRANSACTION文を実行しアクセスモードを設定します。
3. Connection.IsolationLevelプロパティ独立性水準 (IsolationLevelEnum.adXactReadCommitted)を設定します。
4. Connection.BeginTransメソッドでトランザクションを開始します。
5. データ操作が正常終了した場合、Connection.CommitTransメソッドを実行します。
   データ操作が異常終了した場合、Connection.RollbackTransメソッドを実行します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.4.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. アクセスモードの設定
        Con.Execute("SET TRANSACTION READ WRITE", , ADODB.CommandTypeEnum.adCmdText)

        '3. IsolationLevelの設定
        Con.IsolationLevel = ADODB.IsolationLevelEnum.adXactReadCommitted

        '4. トランザクション開始
        Con.BeginTrans()

        'データ操作を行う処理を記述

        '5. コミット
        Con.CommitTrans()

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:
```

```
        '5. ロールバック
        Con.RollbackTrans()

        'エラー処理ルーチンを記述
        MessageBox.Show("▲▼  COMMUNICATION OFF LINE  ▲▼", "No Connect",
         MessageBoxButtons.OK, MessageBoxIcon.None)

    End Sub
End Class
```

B.4.10 エラー処理

エラー処理を行うサンプルコードについて説明します。

On Errorステートメントで、エラー発生時のエラー処理ルーチンを指定します。ADOを使用してODBCエラーが発生した場合、ErrorオブジェクトにSQLSTATEやエラーメッセージなどが格納されます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

ODBC以外のエラーは、Errオブジェクトにエラー情報が格納されます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio .NETドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。発生したエラーのメッセージとSQLSTATEをメッセージボックスに表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   (Connection. ConnectionStringプロパティに接続文字列を間違えて設定します。)
2. On Errorステートメントを使用して、エラー処理ルーチンを設定します。
3. Connection.Openメソッドでコネクションを接続します。
   (パスワードを間違えて指定しエラーを発生させます)
4. エラー処理ルーチンを記述します。
   Errors.Countプロパティで格納されているErrorオブジェクト数を取得します。
   Error.DescriptionプロパティでODBCのエラーメッセージを取得します。
   Error.SQLStateプロパティでSQLSTATEを取得します。
   Error.Clearメソッドでオブジェクトを削除します。
   Err.Descriptionプロパティでエラーメッセージを取得します。
   Err.Clearメソッドでオブジェクトを削除します。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection

        Dim i As Integer
        Dim Err_Count As Integer
        Dim msgstr As String

        '1. Connectionオブジェクトの生成と設定(接続文字列を間違えて設定)
        Con = New ADODB.Connection()
        Con.ConnectionString = " DSN=DSN01;UID=USER01;PWD=XXXXXX;"

        '2. エラー処理ルーチンの設定
        On Error GoTo ErrorProc
```

```
        '3. コネクション接続(エラー発生)

        Con.Open()

        'コネクション切断
        Con.Close()

        'オブジェクトの破棄
        Con = Nothing

        Exit Sub

        '4. エラー処理ルーチン
ErrorProc:
        Err_Count = Con.Errors.Count
        If Err_Count > 0 Then
            For i = 0 To Err_Count - 1
                msgstr &= Con.Errors(i).Description & ControlChars.Cr
                msgstr &= "SQLSTATE: " & Con.Errors(i).SQLState & ControlChars.Cr
            Next
            Con.Errors.Clear()
        Else
            msgstr &= Err.Description
            Err.Clear()
        End If
        MessageBox.Show(msgstr, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        'オブジェクトの破棄
        Con = Nothing


    End Sub
End Class
```

B.5 Visual Basic .NETでのADO.NETのサンプル

Visual Basic .NETでのADO.NETのサンプルコードおよび留意事項について説明します。

なお、このサンプルコードを実行する場合は、Visual Studio 2017を使用してください。

B.5.1 接続および切断

接続および切断をするサンプルコードについて説明します。

本サンプルコードは、接続後メッセージボックスで接続できたことを表示します。

アプリケーションの手順

1. 接続文字列を指定して、OdbcConnectionオブジェクトを生成します。
2. OdbcConnection.Openメソッドでコネクションを接続します。
3. OdbcConnection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        '1. OdbcConnectionオブジェクトの生成
```

```
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '2. コネクション接続
            con.Open()
            'メッセージ表示
            Label1.Text = "□■接続できました■□"
            '3. コネクション切断
            con.Close()
            '4. オブジェクトの破棄
            con.Dispose()
            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException
            'エラー処理ルーチンを記述
            Label1.Text = "▲▼接続不可▲▼"
        End Try

    End Sub
End Class
```

B.5.2 前方向読み取り専用での参照

OdbcDataReaderオブジェクトを使用して、前方向スクロール読み取り専用でデータ参照するサンプルコードについて説明します。

OdbcDataReaderオブジェクトでの参照方法では、参照レコードの取り出し方向はNEXTだけでカーソル位置づけによるデータ更新はできません。
本サンプルコードは、取得データをメッセージボックスで表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。
2. OdbcCommandオブジェクトを生成します。
   (CommandTextプロパティへSELECT文を設定します)
3. OdbcCommand.ExecuteReaderメソッドでOdbcDataReaderオブジェクトを生成します。
4. OdbcDataReaderオブジェクトよりデータを取得します。
   OdbcDataReader.Readメソッドで次の行へ位置づけます。
   OdbcDataReader.FieldCountプロパティで列数を取得します。
   OdbcDataReader.GetValueメソッドでデータを取得します。
5. OdbcDataReader.CloseメソッドでOdbcDataReaderオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand
        Dim drd As System.Data.Odbc.OdbcDataReader

        Dim i As Integer
        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")
```

```
        Try

            '1. コネクション接続
            con.Open()

            '2. OdbcCommandオブジェクトの生成
            com = New System.Data.Odbc.OdbcCommand("SELECT * FROM SAMPLE1.TESTTBL", con)
            '3. OdbcDataReaderオブジェクトの生成
            drd = com.ExecuteReader

            '4. データの取得
            '行の位置づけ
            While drd.Read()
                'データ取得文字列の初期化
                msgstr = ""
                '列数の取得
                For i = 0 To drd.FieldCount - 1
                    'データの取得
                    msgstr &= drd.GetValue(i) & " "
                Next
                'メッセージボックス出力
                MessageBox.Show(msgstr, "DataReader", MessageBoxButtons.OK,
                                            MessageBoxIcon.None)
            End While

            '5.OdbcDataReaderオブジェクトを閉じる
            drd.Close()

            '6.コネクション切断
            con.Close()

            '7.オブジェクトの破棄
            con.Dispose()
            com.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            msgstr = "▲▼接続不可▲▼"
            MessageBox.Show(msgstr, "▲▼接続不可▲▼", MessageBoxButtons.OK,
                                            MessageBoxIcon.None)
        End Try


    End Sub
End Class
```

B.5.3 任意の方向で更新可能な参照

DataSetオブジェクトを使用して、任意の方向で更新可能なデータ参照をするサンプルコードについて説明します。

カーソルとは異なりデータをVisual Basic.NETの内部にコレクション(DataSet)として取り込んで参照します。

更新方法については"B.5.6 データのメモリ内キャッシュを使用した更新"を参照してください。

本サンプルコードでは、取得データをデータグリッドコントロールで表示します。事前にフォームへデータグリッドコントロール(DataGrid1)を追加してください。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、"B.5.1 接続および切断"を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandTextプロパティへSELECT文を設定します。)

3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. DataSet のデータをデータグリッドコントロールにて参照します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim adp As System.Data.Odbc.OdbcDataAdapter
        Dim dst As System.Data.DataSet
        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcDataAdapterオブジェクトの生成
            adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM SAMPLE1.TESTTBL", con)
            '3. DataSetオブジェクトの生成
            dst = New DataSet()
            '4. DataSet オブジェクトへデータを取得
            adp.Fill(dst, "TESTTBL")

            '5. DataSet のデータを DataGrid コントロールで参照する
            DataGrid1.SetDataBinding(dst, "TESTTBL")
            DataGrid1.Refresh()

            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            con.Dispose()
            adp.Dispose()
            dst.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            msgstr = "▲▼　接続不可　▲▼"
            MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                            MessageBoxIcon.None)


        End Try

    End Sub
End Class
```

B.5.4 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

取得したBLOB型データはFileStreamオブジェクトを使用してファイルに出力します。出力したデータをピクチャボックスコントロールで表示します。事前にフォームへピクチャボックスコントロール(PictureBox1)を追加してください。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandTextプロパティへSELECT文を設定します。)
3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. DataSetTables.Rows.Countプロパティで行数を取得します。
6. DataRowオブジェクトへ1行分のデータを取り出します。
7. IsDBNull関数でBLOB型データ列“BLB_C”にデータが存在するかを確認します。
8. DataRowオブジェクトからBLOB型データ列を指定してバイト配列へデータを取り出します。
9. コネクションを切断します。
10. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim adp As System.Data.Odbc.OdbcDataAdapter
        Dim dst As DataSet
        Dim MyImage As Bitmap
        Dim myRow As DataRow
        Dim MyData() As Byte
        Dim fs As System.IO.FileStream
        Dim K As Long
        Dim i As Integer
        Dim msgstr As String
        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")
        Try
            '1. コネクション接続
            con.Open()
            '2. OdbcDataAdapterオブジェクトの生成
            adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM SAMPLE1.IMGTBL", con)
            '3. DataSetオブジェクトの生成
            dst = New DataSet()
            '4. DataSet オブジェクトへデータを取得
            adp.Fill(dst, "IMGTBL")
            '5. 取り出したレコード数分繰り返し
            For i = 0 To dst.Tables("IMGTBL").Rows.Count - 1
                '6. 1行分のデータをDataRowオブジェクトへ取り出す
                myRow = dst.Tables("IMGTBL").Rows(i)
                '7. BLB_C列にデータがあるか
                If Not IsDBNull(myRow("BLB_C")) Then
                    'BLB_C列にデータが存在する場合
                    'Bitmapオブジェクトがすでに存在する場合オブジェクトを破棄
                    If Not (MyImage Is Nothing) Then
                        MyImage.Dispose()
                        MyImage = Nothing
                        PictureBox1.Image = Nothing
                    End If
```

```
                    '8. BLOB型データの取り出し
                    MyData = myRow("BLB_C")
                    'FileStreamオブジェクトの生成(ファイルを開く)
                    fs = New System.IO.FileStream _
                    ("C.BMP", System.IO.FileMode.OpenOrCreate,
                      System.IO.FileAccess.Write)
                    'BLOB型データのサイズを取得
                    K = UBound(MyData)
                    'BLOB型データをファイルへ書き込む
                    fs.Write(MyData, 0, K)
                    'FileStreamオブジェクトを閉じて破棄する
                    fs.Close()
                    fs = Nothing
                    'Bitmapオブジェクトの生成
                    MyImage = New Bitmap("C.BMP")
                    '出力したファイルをPictureBoxコントロールで表示する
                    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
                    PictureBox1.Image = CType(MyImage, Image)
                    PictureBox1.Refresh()
                End If
                'メッセージボックスの表示
                MessageBox.Show(i + 1 & "行目を取得しました",
                 "Message", MessageBoxButtons.OK, MessageBoxIcon.None)
            Next
            '9. コネクション切断
            con.Close()
            '10. オブジェクトの破棄
            con.Dispose()
            dst.Dispose()
            adp.Dispose()
            If Not (MyImage Is Nothing) Then
                MyImage.Dispose()
                MyImage = Nothing
                PictureBox1.Image = Nothing
            End If
            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException
            'エラー処理ルーチンを記述
            msgstr = "▲▼　接続不可　▲▼"
            MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                                 MessageBoxIcon.None)

        End Try

    End Sub
End Class
```

B.5.5 探索条件付きSQL文での更新

探索条件付きSQL文でADO.NET連携でデータの更新を行うサンプルコードについて説明します。

本サンプルコードは、INSERT文実行後、メッセージボックスで完了を知らせます。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。
2. OdbcCommandオブジェクトを生成します。(CommandTextプロパティへUPDATE文を設定します。)
3. OdbcCommand.Parameters.AddメソッドでOdbcParameterオブジェクトの生成と設定を行います。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名を任意で設定します。

OdbcParameter.Valueプロパティへパラメタの値を設定します。
日時型のパラメタにはOdbcParameter.OdbcTypeプロパティを設定します。

4. OdbcCommand.ExecuteNonQueryメソッドでUPDATE文を実行します。
   ExecuteNonQueryメソッドより更新された行数が返されます。
5. 更新行数をメッセージボックスで表示します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand

        Dim i As Integer
        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcCommandオブジェクトの生成
           com = New System.Data.Odbc.OdbcCommand("UPDATE SAMPLE1.TESTTBL SET DEC_C=?,DAT_C=?,CHA_C = ? WHERE
KEY_C = ?", con)

            '3. OdbcParameterオブジェクトの生成と設定
            com.Parameters.AddWithValue("DEC_C", 111111.111)
            com.Parameters.AddWithValue("DAT_C", "2099/12/31")
            com.Parameters("DAT_C").OdbcType = System.Data.Odbc.OdbcType.Date
            com.Parameters.AddWithValue("CHA_C", "更新データ")
            com.Parameters.AddWithValue("KEY_C", 100)


            '4. UPDATE文の実行
            i = com.ExecuteNonQuery
            '5. 更新行数をメッセージボックスで表示する
            MessageBox.Show(i & "行更新されました", "RowCount", MessageBoxButtons.OK,
                                                    MessageBoxIcon.None)

            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            con.Dispose()
            com.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            msgstr = "▲▼  COMMUNICATION OFF LINE  ▲▼"
            MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                            MessageBoxIcon.None)
```

```
        End Try


    End Sub
End Class
```

B.5.6 データのメモリ内キャッシュを使用した更新

DataSetオブジェクトを使用して更新を行うサンプルコードについて説明します。

OdbcCommandBuilderオブジェクトを使用した自動生成コマンドは使用できません。明示的にコマンドを設定してください。

本サンプルコードは、UPDATE文実行後、影響を受けた行数をメッセージボックスで表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandTextプロパティへSELECT文を設定します)
3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. OdbcDataAdapter.InsertCommandプロパティへINSERT文を設定します。
6. InsertCommandプロパティへ設定したパラメタに対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcDataAdapter.InsertCommand.Parameters.Addメソッドで OdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名を任意で設定します。
   OdbcParameter.OdbcTypeプロパティを設定します。
   OdbcParameter.SourceColumnプロパティへパラメタの値(Valueプロパティ)に使用するDataSetの列名を設定します。
7. OdbcDataAdapter.UpdateCommandプロパティへUPDATE文を設定します。
8. UpdateCommandプロパティへ設定したパラメタに対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcDataAdapter.UpdateCommand.Parameters.Addメソッドで OdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名を任意で設定します。
   OdbcParameter.OdbcTypeプロパティを設定します。
   OdbcParameter.SourceColumnプロパティへパラメタの値(Valueプロパティ)に使用するDataSetの列名を設定します。
   探索条件(WHERE句)に設定するOdbcParameter.SourceVersionプロパティにDataRowVersion.Originalを設定します。
9. データの編集をします。
   OdbcDataAdapter.Tables.Rows.Countメソッドを使用してDataSetに取り出した行数を取得します。
   行がある場合は1行目を更新対象行としてDataRowオブジェクトへ取り出します。
   行がない場合はDataSet.Tables.NewRowメソッドで新しい行を作成しDataRowオブジェクトへ取り出します。
   DataRow.BeginEditメソッドで編集モードにしてデータの編集をします
   DataRow.EndEditメソッドで編集を終了します。
   NewRowメソッドにより作成したDataRowオブジェクトをDataSet.Tables.Rows.Addメソッドで追加します。
10. OdbcDataAdapter.UpdateメソッドでDataSetの編集操作の内容により設定したINSERT文かUPDATE文を実行します。
11. DataSet.AcceptChangesメソッドでDataSetに対して行った編集内容をコミットします。
12. コネクションを切断します。
13. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
'オブジェクト宣言
Dim con As System.Data.Odbc.OdbcConnection
Dim adp As System.Data.Odbc.OdbcDataAdapter
Dim dst As DataSet
Dim myRow As DataRow
Dim msgstr As String


'OdbcConnectionオブジェクトの生成
con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01;")

Try
    '1. コネクション接続
    con.Open()

    '2. OdbcDataAdapterオブジェクトの生成
    adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM SAMPLE1.TESTTBL", con)
    '3. DataSetオブジェクトの生成
    dst = New DataSet()
    '4. DataSet オブジェクトへデータを取得
    adp.Fill(dst, "TESTTBL")

    '5. OdbcDataAdapter.InsertCommandプロパティの設定
    adp.InsertCommand = New System.Data.Odbc.OdbcCommand( _
     "INSERT INTO TESTTBL VALUES(?,?,?,?)", con)
    '6. OdbcParameterオブジェクトの生成と設定
    adp.InsertCommand.Parameters.Add("KEY_C", System.Data.Odbc.OdbcType.Int)
    adp.InsertCommand.Parameters("KEY_C").SourceColumn = "KEY_C"
    adp.InsertCommand.Parameters.Add("DEC_C", System.Data.Odbc.OdbcType.Decimal)
    adp.InsertCommand.Parameters("DEC_C").SourceColumn = "DEC_C"
    adp.InsertCommand.Parameters.Add("DAT_C", System.Data.Odbc.OdbcType.Date)
    adp.InsertCommand.Parameters("DAT_C").SourceColumn = "DAT_C"
    adp.InsertCommand.Parameters.Add("CHA_C", System.Data.Odbc.OdbcType.Char)
    adp.InsertCommand.Parameters("CHA_C").SourceColumn = "CHA_C"

    '7. OdbcDataAdapter.UpdateCommandプロパティの設定
    adp.UpdateCommand = New System.Data.Odbc.OdbcCommand( _
     "UPDATE TESTTBL SET KEY_C=?, DEC_C=?, DAT_C=?, CHA_C=? WHERE KEY_C=?", con)
    '8. OdbcParameterオブジェクトの生成と設定
    adp.UpdateCommand.Parameters.Add("KEY_C", System.Data.Odbc.OdbcType.Int)
    adp.UpdateCommand.Parameters("KEY_C").SourceColumn = "KEY_C"
    adp.UpdateCommand.Parameters.Add("DEC_C", System.Data.Odbc.OdbcType.Decimal)
    adp.UpdateCommand.Parameters("DEC_C").SourceColumn = "DEC_C"
    adp.UpdateCommand.Parameters.Add("DAT_C", System.Data.Odbc.OdbcType.Date)
    adp.UpdateCommand.Parameters("DAT_C").SourceColumn = "DAT_C"
    adp.UpdateCommand.Parameters.Add("CHA_C", System.Data.Odbc.OdbcType.Char)
    adp.UpdateCommand.Parameters("CHA_C").SourceColumn = "CHA_C"
    adp.UpdateCommand.Parameters.Add("KEY", System.Data.Odbc.OdbcType.Int)
    adp.UpdateCommand.Parameters("KEY").SourceColumn = "KEY_C"
    adp.UpdateCommand.Parameters("KEY").SourceVersion = DataRowVersion.Original

    '9. データの更新 (DataSetに1行もデータがない場合は行を追加する)
    If Not dst.Tables("TESTTBL").Rows.Count = 0 Then
        myRow = dst.Tables("TESTTBL").Rows(0)
    Else
        myRow = dst.Tables("TESTTBL").NewRow
    End If


    myRow.BeginEdit()
    myRow("KEY_C") = 400
    myRow("DEC_C") = 9999999.999
```

```
            myRow("DAT_C") = "2018/08/03"
            myRow("CHA_C") = "TEST DATA"
            myRow.EndEdit()
            If dst.Tables("TESTTBL").Rows.Count = 0 Then
                dst.Tables("TESTTBL").Rows.Add(myRow)
            End If
            '10. INSERT文／UPDATE文の実行
            adp.Update(dst, "TESTTBL")
            '11.DataSet オブジェクトに対して行われたすべての編集をコミットする
            dst.AcceptChanges()

            MessageBox.Show("データベースにデータが保存されました", _
                            "Normal End", MessageBoxButtons.OK, MessageBoxIcon.None)

            '12.コネクション切断
            con.Close()

            '13.オブジェクトの破棄
            con.Dispose()
            dst.Dispose()
            adp.Dispose()

        'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            msgstr = "▲▼  COMMUNICATION OFF LINE  ▲▼"
            MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                    MessageBoxIcon.None)

        End Try

    End Sub
End Class
```

B.5.7 BLOB型データの更新

BLOB型データの更新を行うサンプルコードについて説明します。

本サンプルコードは、FileStreamオブジェクトを使用してファイルより取得したBLOB型データを、挿入または更新します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandプロパティへSELECT文を設定します)
3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. OdbcDataAdapter.InsertCommandプロパティへINSERT文を設定します。
6. InsertCommandプロパティへ設定したパラメタマーカ('?')に対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcDataAdapter.InsertCommand.Parameters.AddメソッドでOdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名を任意で設定します。
   OdbcParameter.OdbcTypeプロパティを設定します。
   OdbcParameter.SourceColumnプロパティへパラメタの値(Valueプロパティ)に使用するDataSetの列名を設定します。
7. OdbcDataAdapter.UpdateCommandプロパティへUPDATE文を設定します。

8. UpdateCommandプロパティへ設定したパラメタに対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcDataAdapter.UpdateCommand.Parameters.AddメソッドでOdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名を任意で設定します。
   OdbcParameter.OdbcTypeプロパティを設定します。
   OdbcParameter.SourceColumnプロパティへパラメタの値(Valueプロパティ)に使用するDataSetの列名を設定します。
   探索条件(WHERE句)に設定するOdbcParameter.SourceVersionプロパティにDataRowVersion.Originalを設定します。
9. FileStreamオブジェクトを使用してBLOB型データをファイルよりバイト配列へ取得します。
10. データの編集をします。
    OdbcDataAdapter.Tables.Rows.Countメソッドを使用してDataSetに取り出した行数を取得します。
    行がある場合は1行目を更新対象行としてDataRowオブジェクトへ取り出します。
    行がない場合はDataAdapter.Tables.NewRowメソッドで新しい行を作成しDataRowオブジェクトへ取り出します。
    DataRow.BeginEditメソッドで編集モードにしてデータの編集をします。
    DataRow.EndEditメソッドで編集を終了します。
    NewRowメソッドにより作成したDataRowオブジェクトをOdbcDataAdapter.Tables.Rows.Addメソッドで追加します。
11. OdbcDataAdapter.UpdateメソッドでDataSetの編集操作の内容により設定したINSERT文かUPDATE文を実行します。
12. DataSet.AcceptChangesメソッドでDataSetに対して行った編集内容をコミットします。
13. コネクションを切断します。
14. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim adp As System.Data.Odbc.OdbcDataAdapter
        Dim dst As DataSet
        Dim myRow As DataRow

        Dim fs As System.IO.FileStream
        Dim MyData() As Byte
        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcDataAdapterオブジェクトの生成
            adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM SAMPLE1.IMGTBL", con)
            '3. DataSetオブジェクトの生成
            dst = New DataSet()
            '4. DataSet オブジェクトへデータを取得
            adp.Fill(dst, "IMGTBL")
            '5. OdbcDataAdapter.InsertCommandプロパティの設定
            adp.InsertCommand = New System.Data.Odbc.OdbcCommand(
"INSERT INTO IMGTBL VALUES(?,?)", con)
            '6. OdbcParameterオブジェクトの生成と設定
            adp.InsertCommand.Parameters.Add("KEY_C", System.Data.Odbc.OdbcType.Int)
            adp.InsertCommand.Parameters("KEY_C").SourceColumn = "KEY_C"
            adp.InsertCommand.Parameters.Add("BLB_C", System.Data.Odbc.OdbcType.VarBinary)
            adp.InsertCommand.Parameters("BLB_C").SourceColumn = "BLB_C"
```

```
        '7. OdbcDataAdapter.UpdateCommandプロパティの設定
        adp.UpdateCommand = New System.Data.Odbc.OdbcCommand(
"UPDATE IMGTBL SET KEY_C=?, BLB_C=? WHERE KEY_C=?", con)
        '8. OdbcParameterオブジェクトの生成と設定
        adp.UpdateCommand.Parameters.Add("KEY_C", System.Data.Odbc.OdbcType.Int)
        adp.UpdateCommand.Parameters("KEY_C").SourceColumn = "KEY_C"
        adp.UpdateCommand.Parameters.Add("BLB_C", System.Data.Odbc.OdbcType.VarBinary)
        adp.UpdateCommand.Parameters("BLB_C").SourceColumn = "BLB_C"
        adp.UpdateCommand.Parameters.Add("KEY", System.Data.Odbc.OdbcType.Int)
        adp.UpdateCommand.Parameters("KEY").SourceColumn = "KEY_C"
        adp.UpdateCommand.Parameters("KEY").SourceVersion = DataRowVersion.Original

        '9. BLOB型データをファイルより取得する
        fs = New System.IO.FileStream _
 ("C:¥yamasaki¥test.bmp", System.IO.FileMode.OpenOrCreate,
  System.IO.FileAccess.Read)
        ReDim MyData(fs.Length)
        fs.Read(MyData, 0, fs.Length)
        fs.Close()
        fs = Nothing

        '10. データの更新(DataSetに1行もデータがない場合は行を追加する)
        If Not dst.Tables("IMGTBL").Rows.Count = 0 Then
            myRow = dst.Tables("IMGTBL").Rows(0)
        Else
            myRow = dst.Tables("IMGTBL").NewRow
        End If

        myRow.BeginEdit()
        myRow("KEY_C") = 1
        myRow("BLB_C") = MyData
        myRow.EndEdit()

        If dst.Tables("IMGTBL").Rows.Count = 0 Then
            dst.Tables("IMGTBL").Rows.Add(myRow)
        End If

        '11. INSERT文/UPDATE文の実行
        adp.Update(dst, "IMGTBL")
        '12. DataSet オブジェクトに対して行われたすべての編集をコミットする
        dst.AcceptChanges()

        MessageBox.Show("データベースにBLOB型データが保存されました",
                 "Normal End", MessageBoxButtons.OK, MessageBoxIcon.None)

        '13. コネクション切断
        con.Close()

        '14. オブジェクトの破棄
        con.Dispose()
        dst.Dispose()
        adp.Dispose()

        'エラー処理
    Catch ex As System.Data.Odbc.OdbcException

        'エラー処理ルーチンを記述
        msgstr = "▲▼  COMMUNICATION OFF LINE  ▲▼"
        MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                        MessageBoxIcon.None)
```

```
        End Try


    End Sub
End Class
```

B.5.8 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をメッセージボックスで表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。
2. OdbcCommandオブジェクトを生成します。(CommandTextプロパティへCALL文を設定します)
3. CommandTextプロパティへ設定したパラメタマーカ('?')に対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcCommand.Parameters.AddメソッドでOdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名を任意で設定します。
   OdbcParameter.Directionプロパティへ入力(Input)、出力(Output)、入出力(InputOutput)を設定します。
   OdbcParameter.Valueプロパティへパラメタの値を設定します。
   文字列型の出力パラメタおよび入出力パラメタのOdbcParameter.Sizeプロパティには、最大サイズをバイト数で設定します。
4. OdbcCommand.ExecuteNonQueryメソッドでCALL文を実行します。
5. 出力および入出力パラメタの結果をメッセージボックスで表示します。
   OdbcParameter.Valueプロパティで結果が参照できます。
   このサンプルコードでは、出力パラメタごとに改行し表示します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load


        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand

        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcCommandオブジェクトの生成
            com = New System.Data.Odbc.OdbcCommand("CALL SAMPLE1.COUNTPRC(?,?,?,?)", con)

            '3. OdbcParameterオブジェクトの生成と設定
            com.Parameters.Add("DAT_P", System.Data.Odbc.OdbcType.Date)
            com.Parameters("DAT_P").Direction = ParameterDirection.Input
            com.Parameters("DAT_P").Value = "2007 - 4 - 10"
            com.Parameters.Add("OUT_P", System.Data.Odbc.OdbcType.Int)
```

```
            com.Parameters("OUT_P").Direction = ParameterDirection.Output
            com.Parameters.Add("PRCSTATE", System.Data.Odbc.OdbcType.Char)
            com.Parameters("PRCSTATE").Direction = ParameterDirection.Output
            com.Parameters("PRCSTATE").Size = 5
            com.Parameters.Add("PRCMSG", System.Data.Odbc.OdbcType.Char)
            com.Parameters("PRCMSG").Direction = ParameterDirection.Output
            com.Parameters("PRCMSG").Size = 255

            '4. CALL文実行
            com.ExecuteNonQuery()

            '5. 結果の表示
            msgstr = "COUNT: " & com.Parameters("OUT_P").Value & ControlChars.Cr
            msgstr &= "PRCSTATE: " & com.Parameters("PRCSTATE").Value & ControlChars.Cr
            msgstr &= "PRCMSG: " & com.Parameters("PRCMSG").Value
            MessageBox.Show(msgstr, "PROCEDURE DATA",
                    MessageBoxButtons.OK, MessageBoxIcon.None)

            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            con.Dispose()
            com.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            msgstr = "▲▼  COMMUNICATION OFF LINE  ▲▼"
            MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                        MessageBoxIcon.None)

        End Try


    End Sub
End Class
```

B.5.9 トランザクション制御

ADO.NET連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

OdbcConnection.BeginTransactionメソッド、OdbcTransaction.CommitメソッドまたはOdbcTransaction.Rollbackメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードの設定または変更は、SET TRANSACTION文で行います。

独立性水準の設定または変更は、OdbcConnection.BeginTransactionメソッドの引数IsolationLevelで指定するまたはSET TRANSACTION文で行います。SET TRANSACTION文で設定または変更を行う場合は、OdbcConnection.BeginTransactionメソッド実行時に引数IsolationLevelの設定値に変更されるため、OdbcConnection.BeginTransactionメソッド実行後に行う必要があります。このため、引数IsolationLevelによる設定または変更を推奨します。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定後、INSERT文を実行します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“B.5.1 接続および切断”を参照してください。

2. OdbcCommand.ExecuteNonQueryメソッドでSET TRANSACTION文を実行しアクセスモードを設定します。
3. OdbcConnection.BeginTransactionメソッドでトランザクションを開始します。
   引数IsolationLevelに独立性水準(IsolationLevel.ReadCommitted)を設定します。
   OdbcTransactionオブジェクトが作成されます。
4. 作成されたOdbcTransactionオブジェクトを、データ操作を行うオブジェクト(OdbcCommand)のTransactionプロパティに設定し実行します。
5. データ操作が正常終了した場合、OdbcTransaction.Commitメソッドを実行します。
   データ操作が異常終了した場合、OdbcTransaction.Rollbackメソッドを実行します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“B.5.10 エラー処理”を参照してください。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand
        Dim trn As System.Data.Odbc.OdbcTransaction
        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01;")

        Try
            '1. コネクション接続
            con.Open()

            '2. アクセスモードの設定
            com = New System.Data.Odbc.OdbcCommand("SET TRANSACTION READ WRITE", con)
            com.ExecuteNonQuery()

            '3. トランザクション開始(独立性水準の設定)
            'OdbcTransactionオブジェクトの生成
            trn = con.BeginTransaction(IsolationLevel.ReadCommitted)

            '4. データ操作を行う処理を記述
            com.CommandText = "INSERT INTO SAMPLE1.TESTTBL(KEY_C) VALUES(1)"

            'OdbcCommand.Transactionプロパティに生成したOdbcTransactionオブジェクトを設定
            com.Transaction = trn
            com.ExecuteNonQuery()

            '5. コミット
            trn.Commit()

            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            com.Dispose()
            con.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException
            '5. ロールバック
            trn.Rollback()

            msgstr = "▲▼  COMMUNICATION OFF LINE  ▲▼"
```

```
            MessageBox.Show(msgstr, "！ERROR！", MessageBoxButtons.OK,
                                                MessageBoxIcon.None)


        End Try

    End Sub
End Class
```

B.5.10 エラー処理

エラー処理を行うサンプルコードについて説明します。

Tryステートメントからcatchステートメントの範囲内にエラーが発生する可能性のあるコードを記述し、Catchステートメントでエラー発生時のエラー処理を記述します。ADO.NETを使用してODBCエラーが発生した場合、OdbcExceptionクラスをキャッチすることで、SQLSTATEやエラーメッセージなどが確認できます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

ODBC以外のエラーは、Exceptionクラスをキャッチすることでエラー情報が確認できます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio .NET ドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。発生したエラーのメッセージとSQLSTATEをメッセージボックスに表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトを生成します。(ConnectionStringプロパティに接続文字列を間違えて設定します。)
2. TryステートメントからCatchステートメントの範囲内にエラーが発生する可能性のあるコードを記述します。
3. OdbcConnection.Openメソッドでコネクションを接続します。(パスワードを間違えて指定しエラーを発生させます)
4. Catchステートメントに処理したいエラーのクラスを指定します。
   OdbcException.Errors.CountプロパティでODBCのエラー数を取得します。
   OdbcException.Errors.MessageプロパティでODBCのエラーメッセージを取得します。
   OdbcException.Errors.SQLStateプロパティにでSQLSTATEを取得します。
   Exception.MessageプロパティでODBC以外のエラーメッセージを取得します。
5. 処理がすべて行われた後に行う処理を、Finallyステートメント以降に記述します。

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection

        Dim i As Integer
        Dim msgstr As String

        '1. OdbcConnectionオブジェクトの生成(コネクション文字列を間違えて設定)
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=XXXXXX ")

        '2. Try ～ Catch の範囲内にエラーが発生する可能性のあるコードを記述

        Try

            '3. コネクション接続(エラー発生)
            con.Open()
```

```
            'コネクション切断
            con.Close()

            '4. OdbcExceptionクラスのキャッチ
        Catch ex As System.Data.Odbc.OdbcException
            For i = 0 To ex.Errors.Count - 1
                msgstr &= ex.Errors(i).Message & ControlChars.Cr
                msgstr &= "SQLSTATE: " & ex.Errors(i).SQLState & ControlChars.Cr
            Next
            MessageBox.Show(msgstr, "ODBC Error Message",
                    MessageBoxButtons.OK, MessageBoxIcon.Error)

            '4. Exceptionクラスのキャッチ
        Catch ex As Exception
            msgstr = ex.Message
            MessageBox.Show(msgstr, "Error Message",
                    MessageBoxButtons.OK, MessageBoxIcon.Error)

            '5. Tryステートメント終了時の処理
        Finally

            'オブジェクトの破棄
            con.Dispose()

        End Try

    End Sub
End Class
```

付録C　Webアプリケーションのサンプル

本付録では、Webアプリケーションのサンプルコードおよび留意事項について説明します。

C.1　サンプル実行前の準備

本付録に記載されているサンプルコードは、以下のデータベース定義環境への接続を想定しています。

参照

データベースの作成についての詳細は、“RDB運用ガイド(データベース定義編)”を参照してください。

また、データベース簡単運用でのデータベース作成については、“データベース簡単運用ガイド”を参照してください。

データベース定義環境

サンプルコードを実行する場合は、ODBCデータソースのデフォルトスキーマ名に以下のスキーマ名を指定してください。

- スキーマ名：SAMPLE1

表の定義

表名：TESTTBL

列名	データ型	列制約定義
KEY_C	INTEGER	NOT NULL PRIMARY KEY
DEC_C	DECIMAL(10,3)	
DAT_C	DATE	
CHA_C	CHAR(250)	

表の定義

表名：IMGTBL

列名	データ型	列制約定義
KEY_C	INTEGER	NOT NULL PRIMARY KEY
BLB_C	BINARY LARGE OBJECT (1M)	

プロシジャルーチンの定義

ルーチン名：COUNTPRC

パラメタモード	パラメタ名	データ型
IN	DAT_P	DATE
OUT	OUT_P	INTEGER
OUT	PRCSTATE	CHAR(5)
OUT	PRCMSG	CHAR(255)
複合文		
DECLARE SQLSTATE CHAR(5); DECLARE SQLMSG CHAR(255); SELECT COUNT(*) INTO OUT_P FROM SAMPLE1.TESTTBL WHERE DAT_C > DAT_P;		

パラメタモード	パラメタ名	データ型
SET PRCSTATE = SQLSTATE; SET PRCMSG = SQLMSG;		

C.2 Visual Basic .NETでのADOのサンプル

Visual Basic .NETでのADOのサンプルコードおよび留意事項について説明します。

なお、このサンプルコードを実行する場合は、Visual Studio 2017を使用してください。

C.2.1 接続および切断

接続および切断をするサンプルコードについて説明します。

本サンプルコードは、Webページ上に接続できたことを表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   Connection.ConnectionStringプロパティへ接続文字列を設定します。
2. Connection.Openメソッドでコネクションを接続します。
3. Connection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default

    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection

        '1. ADODB.Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection()
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '2. コネクション接続
        Con.Open()

        'メッセージの表示
        Label1.Text = "Connect<br>接続できました"

        '3. コネクション切断
        Con.Close()

        '4. オブジェクトの破棄
        Con = Nothing

    Exit Sub

        'エラー処理
 ErrorProc:

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"
```

```
    End Sub
End Class
```

C.2.2 データの参照

ADOとの連携でデータの参照を行う方法について説明します。

本サンプルコードは、取得データをWebページ上に表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. Recordsetオブジェクトよりデータを取得します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Fields.Countプロパティで列数を確認します。
   Field.Valueプロパティでデータを取得します。
   Recordset.MoveNextメソッドでカレント行を次の行へと移動します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

注意

Recordset.EOFプロパティでカレント行がEOFかどうか確認後は、以下のエラーがアプリケーション側で認識できなくなります。

- JYP2099E デッドロックが発生しました.
- JYP5014E スキーマ“@1@”の表“@2@”は占有中です.

これらのエラーをアプリケーション側で認識するタイミングは、Recordset.MoveFirstメソッドを実行するか、データを参照した時です。上記エラーを認識するためには、EOFプロパティを確認せずにMoveFirstメソッドの実行またはデータを参照してください。

ただし、SELECT文の検索結果が0件であった場合、Visual Basicの実行時エラー“3021”が発生します。エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default

    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        Dim i As Integer
        Dim msgstr As String

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc
```

```
        '1. コネクション接続
        Con.Open()

        '2. Recordsetオブジェクトの生成
        Rst = New ADODB.Recordset

        '3. Recordsetオブジェクトを開く
        Rst.Open("SELECT * FROM TESTTBL", Con,
         ADODB.CursorTypeEnum.adOpenForwardOnly,
         ADODB.LockTypeEnum.adLockReadOnly,
         ADODB.CommandTypeEnum.adCmdText)

        '4. データの取得
        'EOFまで繰り返し
        Do Until Rst.EOF
            '列数の取得
            For i = 0 To Rst.Fields.Count - 1
                'データの取得
                msgstr &= Rst.Fields(i).Value() & " "
            Next
            msgstr &= "<br>"
            '行の位置づけ
            Rst.MoveNext()
        Loop

        '取得データの表示
        Label1.Text = "Recordset<br>" & msgstr

        '5. Recordsetオブジェクトを閉じる
        Rst.Close()

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Rst = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

    End Sub
End Class
```

C.2.3 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

本サンプルコードは、1行目のBLOB型データをGetChunkメソッドで16384バイトずつ取得します。取得したBLOB型データは、FileStreamオブジェクトを使用してファイルに出力します。出力したデータをイメージコントロールで表示します。事前にフォームへイメージコントロール“Image1”を追加してください。なお、本サンプルで表示する、BLOB型データは予めVisual Basicなどを利用して登録してください。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。

2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. Recordsetオブジェクトよりデータを取得します。
   Recordset.BOFプロパティでカレント行がBOFかを確認します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Field.ActualSizeプロパティで取得データのバイト数を確認します。
   Field.GetChunkメソッドで指定バイト数分データを取得します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        Dim fs As System.IO.FileStream
        Dim Chunks As Short
        Dim Fl As Integer
        Dim Fragment As Short
        Dim Chunk() As Byte
        Const ChunkSize As Short = 16384
        Dim k As Integer


        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Recordsetオブジェクトの生成
        Rst = New ADODB.Recordset

        '3. Recordsetオブジェクトを開く
        Rst.Open("SELECT * FROM IMGTBL", Con,
         ADODB.CursorTypeEnum.adOpenForwardOnly,
         ADODB.LockTypeEnum.adLockReadOnly,
         ADODB.CommandTypeEnum.adCmdText)

        '4. データの取得
        '結果列が存在するか
        If Not (Rst.BOF And Rst.EOF) Then

            'BLOB型データのサイズを取得
            Fl = Rst.Fields("BLB_C").ActualSize

            'BLB_C列にデータがあるか
            If Fl > 0 Then
                'BLOB型データが存在する場合
```

```
                '規定量にデータを分けて取得するため、分割数と残量を求める
                Chunks = Fl ¥ ChunkSize
                Fragment = Fl Mod ChunkSize
                '残量分のデータ取得
                ReDim Chunk(Fragment - 1)
                Chunk = Rst.Fields("BLB_C").GetChunk(Fragment)
                'FileStreamオブジェクトの生成(ファイルを開く)
                fs = New System.IO.FileStream _
          (Server.MapPath("C.BMP"), System.IO.FileMode.OpenOrCreate,
           System.IO.FileAccess.Write)

                'BLOB型データをファイルに出力する
                fs.Write(Chunk, 0, Fragment)

                ReDim Chunk(ChunkSize - 1)
                '分割数分繰り返し
                For k = 1 To Chunks
                    '規定量分のデータ取得
                    Chunk = Rst.Fields("BLB_C").GetChunk(ChunkSize)
                    'BLOB型データをファイルに出力する
                    fs.Write(Chunk, 0, ChunkSize)
                Next k

                'FileStreamオブジェクトを閉じて破棄する
                fs.Close()
                fs = Nothing

                '出力したファイルをImageコントロールで表示する
                Image1.ImageUrl = "C.BMP"

            End If

        End If


        '5. Recordsetオブジェクトを閉じる
        Rst.Close()

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Rst = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述

    End Sub
End Class
```

C.2.4 データの挿入

ADOとの連携でデータの挿入を行う方法について説明します。

本サンプルコードは、INSERT文実行後、Webページ上で完了を知らせます。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにINSERT文を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdTextを設定します。
3. Command.ExecuteメソッドでINSERT文を実行します。
4. コネクションを切断します。
5. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command

        'Connectionオブジェクトの生成
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Commandオブジェクトの生成と設定
        Com = New ADODB.Command
        Com.ActiveConnection = Con
        Com.CommandText =
 "INSERT INTO TESTTBL VALUES(100,1000.025,DATE'2018-08-07','INSERT DATA')"
        Com.CommandType = ADODB.CommandTypeEnum.adCmdText


        '3. INSERT文実行
        Com.Execute()

        'メッセージの表示
        Label1.Text = "Normal End<br>行を挿入しました"

        '4. コネクション切断
        Con.Close()

        '5. オブジェクトの破棄
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

    End Sub
End Class
```

C.2.5　パラメタマーカを使用したSQL文での更新

パラメタマーカ('?')を使用したSQL文でのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、UPDATE文実行後、影響を受けた行数をWebページ上に表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにUPDATE文を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdTextを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名前を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、パラメタが入力パラメタ(adParamInput)であることを設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
   Parameter.Valueプロパティには、値を設定します。
   Parameter.Precisionプロパティには、精度を設定します。
   Parameter.NumericScaleプロパティには、値の小数点以下の桁数を設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。
5. Command.ExecuteメソッドでUPDATE文を実行します。
   引数RecordsAffectedを指定しExecuteメソッドで影響を受けた行数を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command
        Dim Par1, Par2, Par3, Par4 As ADODB.Parameter

        Dim ra As Long

        'Connectionオブジェクトの生成
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Commandオブジェクトの生成と設定
        Com = New ADODB.Command
        Com.ActiveConnection = Con
        Com.CommandText = "UPDATE TESTTBL SET DEC_C=?,DAT_C=?,CHA_C=? WHERE KEY_C=?"
        Com.CommandType = ADODB.CommandTypeEnum.adCmdText

        '3. Parameterオブジェクトの生成と設定
        Par1 = Com.CreateParameter("DEC_C", ADODB.DataTypeEnum.adDecimal,
```

```
         ADODB.ParameterDirectionEnum.adParamInput, , 2000.222)
        Par1.Precision = 10
        Par1.NumericScale = 3
        Par2 = Com.CreateParameter("DAT_C", ADODB.DataTypeEnum.adDate,
         ADODB.ParameterDirectionEnum.adParamInput, , "2018-08-07")
        Par3 = Com.CreateParameter("CHA_C", ADODB.DataTypeEnum.adChar,
         ADODB.ParameterDirectionEnum.adParamInput, 250, "UPDATE DATA3")
        Par4 = Com.CreateParameter("KEY_C", ADODB.DataTypeEnum.adInteger,
         ADODB.ParameterDirectionEnum.adParamInput, , 100)

        '4. ParametersコレクションにParameterオブジェクトを追加
        Com.Parameters.Append(Par1)
        Com.Parameters.Append(Par2)
        Com.Parameters.Append(Par3)
        Com.Parameters.Append(Par4)

        '5. UPDATE文実行
        Com.Execute(ra)

        'メッセージボックスの表示
        Label1.Text = "Normal End<br>" & ra & "行更新されました"

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Par1 = Nothing
        Par2 = Nothing
        Par3 = Nothing
        Par4 = Nothing
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

    End Sub
End Class
```

C.2.6 カーソル位置づけでのデータ更新

カーソル位置づけでのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、Recordsetオブジェクトを使用してデータ更新を行います。Recordsetオブジェクトに行が1行もない場合には行を挿入し、1行以上ある場合には1行目を更新します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. データを編集します。(行がない場合は行を挿入します)
   Recordset.BOFプロパティでカレント行がBOFかを確認します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。

Recordset.AddNewメソッドでRecordsetオブジェクトに新しい行を作成します。
Field.Valueプロパティへ値を設定します。

5. Recordset.UpdateメソッドでRecordsetオブジェクトの編集内容によりINSERT文かUPDATE文を実行します。
6. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
7. コネクションを切断します。
8. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Rst As ADODB.Recordset

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Recordsetオブジェクトの生成
        Rst = New ADODB.Recordset

        '3. Recordsetオブジェクトを開く
        Rst.Open("SELECT * FROM TESTTBL", Con,
         ADODB.CursorTypeEnum.adOpenStatic,
         ADODB.LockTypeEnum.adLockPessimistic,
         ADODB.CommandTypeEnum.adCmdText)

        '4. データの編集(rdoResultsetに1行もデータがない場合は行を追加する)
        If Rst.BOF And Rst.EOF Then
            Rst.AddNew()
        End If

        Rst.Fields("KEY_C").Value = 300
        Rst.Fields("DEC_C").Value = 11111.111
        Rst.Fields("DAT_C").Value = "2018-08-08"
        Rst.Fields("CHA_C").Value = "UPDATE DATA5"

        '5. INSERT文/UPDATE文の実行
        Rst.Update()

        'メッセージの表示

        Label1.Text = "Normal End<br>行を更新しました"

        '6. Recordsetオブジェクトを閉じる
        Rst.Close()

        '7. コネクション切断
        Con.Close()

        '8. オブジェクトの破棄
        Rst = Nothing
```

```
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

    End Sub
End Class
```

C.2.7 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をWebページ上に表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにストアドプロシジャ名を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdStoredProcを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名前を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、入力パラメタ(adParamInput)、出力パラメタ(adParamOutput)のいずれかであることを設定します。
   入力パラメタのParameter.Valueプロパティには値を設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。
5. Command . Executeメソッドでストアドプロシジャを実行します。
   ストアドプロシジャの結果を、出力パラメタのParameter.Valueプロパティで結果を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection
        Dim Com As ADODB.Command
        Dim Par1, Par2, Par3, Par4 As ADODB.Parameter

        Dim msgstr As String

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01;"
```

```
        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. Commandオブジェクトの生成と設定
        Com = New ADODB.Command
        Com.ActiveConnection = Con
        Com.CommandText = "COUNTPRC"
        Com.CommandType = ADODB.CommandTypeEnum.adCmdStoredProc

        '3. Parameteオブジェクトの生成と設定
        Par1 = Com.CreateParameter("DAT_P", ADODB.DataTypeEnum.adDate,
         ADODB.ParameterDirectionEnum.adParamInput, , "2018-08-07")
        Par2 = Com.CreateParameter("OUT_P", ADODB.DataTypeEnum.adInteger,
         ADODB.ParameterDirectionEnum.adParamOutput)
        Par3 = Com.CreateParameter("PRCSTATE", ADODB.DataTypeEnum.adChar,
         ADODB.ParameterDirectionEnum.adParamOutput, 5)
        Par4 = Com.CreateParameter("PRCMSG", ADODB.DataTypeEnum.adChar,
         ADODB.ParameterDirectionEnum.adParamOutput, 255)

        '4. ParametersコレクションにParameterオブジェクトを追加
        Com.Parameters.Append(Par1)
        Com.Parameters.Append(Par2)
        Com.Parameters.Append(Par3)
        Com.Parameters.Append(Par4)

        '5. ストアドプロシジャの実行
        Com.Execute()

        '結果を表示
        msgstr = "COUNT: " & Com.Parameters("OUT_P").Value & "<br>"
        msgstr &= "PRCSTATE: " & Com.Parameters("PRCSTATE").Value & "<br>"
        msgstr &= "PRCMSG: " & Com.Parameters("PRCMSG").Value
        Label1.Text = "PROCEDURE DATA<br>" & msgstr

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Par1 = Nothing
        Par2 = Nothing
        Par3 = Nothing
        Par4 = Nothing
        Com = Nothing
        Con = Nothing

        Exit Sub

        'エラー処理

ErrorProc:

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

    End Sub
End Class
```

C.2.8 トランザクション制御

ADOとの連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

Connection.BeginTransメソッド、Connection.CommitTransメソッドまたは Connection.RollbackTransメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードの設定または変更は、SET TRANSACTION文または Connection.Modeプロパティで行います。Connection.Modeプロパティはコネクションが切れている時だけ設定または変更することができます。コネクション接続中に変更が可能なSET TRANSACTION文による設定または変更を推奨します。

独立性水準の設定または変更は、Connection.IsolationLevelプロパティまたはSET TRANSACTION文で行います。SET TRANSACTION文で設定または変更を行う場合は、Connection.BeginTransメソッド実行時にConnection.IsolationLevelプロパティ設定値に変更されるため、Connection.BeginTransメソッド実行後に行う必要があります。このため、Connection.IsolationLevelプロパティによる設定または変更を推奨します。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.2.1 接続および切断”を参照してください。
2. Connection.ExecuteメソッドでSET TRANSACTION文を実行しアクセスモードを設定します。
3. Connection.IsolationLevelプロパティ独立性水準
   (IsolationLevelEnum.adXactReadCommitted)を設定します。
4. Connection.BeginTransメソッドでトランザクションを開始します。
5. データ操作が正常終了した場合、Connection.CommitTransメソッドを実行します。
   データ操作が異常終了した場合、Connection.RollbackTransメソッドを実行します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.2.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection

        'Connectionオブジェクトの生成と設定
        Con = New ADODB.Connection
        Con.Mode = ADODB.ConnectModeEnum.adModeRead
        Con.ConnectionString = "DSN=checkRDB;UID=ADMINISTRATOR;PWD=symfo000@admin"

        On Error GoTo ErrorProc

        '1. コネクション接続
        Con.Open()

        '2. アクセスモードの設定
        Con.Execute("SET TRANSACTION READ WRITE", , ADODB.CommandTypeEnum.adCmdText)

        '3. IsolationLevelの設定
        Con.IsolationLevel = ADODB.IsolationLevelEnum.adXactReadCommitted

        '4. トランザクション開始
        Con.BeginTrans()

        'データ操作を行う処理を記述
```

```
        '5. コミット
        Con.CommitTrans()

        '6. コネクション切断
        Con.Close()

        '7. オブジェクトの破棄
        Con = Nothing

        Exit Sub

        'エラー処理
ErrorProc:

        '5. ロールバック
        Con.RollbackTrans()

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

    End Sub
End Class
```

C.2.9 エラー処理

エラー処理を行うサンプルコードについて説明します。

On Errorステートメントで、エラー発生時のエラー処理ルーチンを指定します。ADOを使用してODBCエラーが発生した場合、ErrorオブジェクトにSQLSTATEやエラーメッセージなどが格納されます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

ODBC以外のエラーは、Errオブジェクトにエラー情報が格納されます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio .NETドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。発生したエラーのメッセージとSQLSTATEをWebページ上に表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   (Connection. ConnectionStringプロパティに接続文字列を間違えて設定する)
2. On Errorステートメントを使用して、エラー処理ルーチンを設定します。
3. Connection.Openメソッドでコネクションを接続します。
   (パスワードを間違えて指定しエラーを発生させます)
4. エラー処理ルーチンを記述します。
   Errors.Countプロパティで格納されているErrorオブジェクト数を取得します。
   Error.DescriptionプロパティでODBCのエラーメッセージを取得します。
   Error.SQLStateプロパティでSQLSTATEを取得します。
   Error.ClearメソッドでErrorオブジェクトを削除します。
   Err.Descriptionプロパティでエラーメッセージを取得します。
   Err.ClearメソッドでErrオブジェクトを削除します。

```
Public Class _Default
    Inherits Page
```

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim Con As ADODB.Connection

        Dim i As Integer
        Dim Err_Count As Integer
        Dim msgstr As String

        '1. Connectionオブジェクトの生成と設定(接続文字列を間違えて設定)
        Con = New ADODB.Connection
        Con.ConnectionString = " DSN=DSN01;UID=USER01;PWD=XXXXXX;"

        '2. エラー処理ルーチンの設定
        On Error GoTo ErrorProc

        '3. コネクション接続(エラー発生)
        Con.Open()

        'コネクション切断
        Con.Close()

        'オブジェクトの破棄
        Con = Nothing

        Exit Sub

        '4. エラー処理ルーチン
ErrorProc:
        Err_Count = Con.Errors.Count
        If Err_Count > 0 Then
            For i = 0 To Err_Count - 1
                msgstr &= Con.Errors(i).Description & "<br>"
                msgstr &= "SQLSTATE: " & Con.Errors(i).SQLState & "<br>"
            Next
            Con.Errors.Clear()
        Else
            msgstr &= Err.Description
            Err.Clear()
        End If
        Label1.Text = "Error<br>" & msgstr
        'オブジェクトの破棄
        Con = Nothing

    End Sub
End Class
```

C.3 Visual Basic .NETでのADO.NETのサンプル

Visual Basic .NETでのADO.NETのサンプルコードおよび留意事項について説明します。

なお、このサンプルコードを実行する場合は、Visual Studio 2017を使用してください。

C.3.1 接続および切断

接続および切断をするサンプルコードについて説明します。

本サンプルコードは、接続後、Webページ上に接続できたことを表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトを生成します。
   ConnectionStringプロパティへ接続文字列を設定します。
2. OdbcConnection.Openメソッドでコネクションを接続します。
3. OdbcConnection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection

        '1. OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try

            '2. コネクション接続
            con.Open()

            'メッセージ表示
            Label1.Text = "接続できました"

            '3. コネクション切断
            con.Close()

            '4. オブジェクトの破棄
            con.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"

        End Try

    End Sub
End Class
```

C.3.2 前方向読み取り専用での参照

OdbcDataReaderオブジェクトを使用して、前方向スクロール読み取り専用でデータ参照するサンプルコードについて説明します。

OdbcDataReaderオブジェクトでの参照方法では、参照レコードの取り出し方向はNEXTだけでカーソル位置づけによるデータ更新はできません。

本サンプルコードは、取得データをWebページ上に表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcCommandオブジェクトを生成します。(CommandTextプロパティへSELECT文を設定します)

3. OdbcCommand.ExecuteReaderメソッドでOdbcDataReaderオブジェクトを生成します。
4. OdbcDataReaderオブジェクトよりデータを取得します。
   OdbcDataReader.Readメソッドで次の行へ位置づけます。
   OdbcDataReader.FieldCountプロパティで列数を取得します。
   OdbcDataReader.GetValueメソッドでデータを取得します。
5. OdbcDataReader.CloseメソッドでOdbcDataReaderオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand
        Dim drd As System.Data.Odbc.OdbcDataReader

        Dim i As Integer
        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try

            '1. コネクション接続
            con.Open()

            '2. OdbcCommandオブジェクトの生成
            com = New System.Data.Odbc.OdbcCommand("SELECT * FROM TESTTBL", con)

            '3. OdbcDataReaderオブジェクトの生成
            drd = com.ExecuteReader

            '4. データの取得
            '行の位置づけ
            While drd.Read()
                '列数の取得
                For i = 0 To drd.FieldCount - 1
                    'データの取得
                    msgstr &= drd.GetValue(i) & " "
                Next
                msgstr &= "<br>"
            End While
            'メッセージ出力
            Label1.Text = "DataReader<br>" & msgstr

            '5. OdbcDataReaderオブジェクトを閉じる
            drd.Close()

            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            con.Dispose()
            com.Dispose()

            'エラー処理
```

```
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"
        End Try

    End Sub
End Class
```

C.3.3 任意の方向で更新可能な参照

DataSetオブジェクトを使用して、任意の方向で更新可能なデータ参照をするサンプルコードについて説明します。

カーソルとは異なりデータをVisual Basic .NETの内部にコレクション(DataSet)として取り込んで参照します。

更新方法については“C.3.6 データのメモリ内キャッシュを使用した更新”を参照してください。

本サンプルコードでは、取得データをデータグリッドコントロールで表示します。事前にフォームへデータグリッドコントロール(DataGrid1)を追加してください。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandTextプロパティへSELECT文を設定する)
3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. DataSet のデータをデータグリッドコントロールにて参照します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim adp As System.Data.Odbc.OdbcDataAdapter
        Dim dst As DataSet

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcDataAdapterオブジェクトの生成
            adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM TESTTBL", con)
            '3. DataSetオブジェクトの生成
            dst = New DataSet
            '4. DataSet オブジェクトへデータを取得
            adp.Fill(dst, "TESTTBL")

            '5. DataSet のデータを DataGrid コントロールで参照する
            DataGrid1.DataSource = dst
            DataGrid1.DataMember = "TESTTBL"
            DataGrid1.DataBind()
```

```
            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            con.Dispose()
            adp.Dispose()
            dst.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述

        End Try

    End Sub
End Class
```

C.3.4 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

本サンプルコードは、1行目のBLOB型データをバイト配列に取得します。取得したBLOB型データはFileStreamオブジェクトを使用してファイルに出力します。出力したデータをイメージコントロールで表示します。事前にフォームへイメージコントロール“PictureBox1”を追加してください。なお、本サンプルで表示するBLOB型データは、あらかじめVisual Basicなどを利用して登録してください。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandTextプロパティへSELECT文を設定する)
3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. DataSetTables.Rows.Countプロパティで行数を取得します。
6. DataRowオブジェクトへ1行分のデータを取り出します。
7. IsDBNull関数でBLOB型データの列“BLB_C”にデータが存在するかを確認します。
8. DataRowオブジェクトからBLOB型データの列を指定してバイト配列へデータを取り出します。
9. コネクションを切断します。
10. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim adp As System.Data.Odbc.OdbcDataAdapter
        Dim dst As DataSet

        Dim myRow As DataRow
        Dim MyData() As Byte
        Dim fs As System.IO.FileStream
        Dim K As Long
```

```
        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcDataAdapterオブジェクトの生成
            adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM IMGTBL", con)
            '3. DataSetオブジェクトの生成
            dst = New DataSet
            '4. DataSet オブジェクトへデータを取得
            adp.Fill(dst, "IMGTBL")

            '5. 結果列が存在するか
            If dst.Tables("IMGTBL").Rows.Count > 0 Then

                '6. 1行分のデータをDataRowオブジェクトへ取り出す
                myRow = dst.Tables("IMGTBL").Rows(0)

                '7. BLB_C列にデータがあるか
                If Not IsDBNull(myRow("BLB_C")) Then
                    'BLB_C列にデータが存在する場合
                    '8. BLOB型データの取り出し
                    MyData = myRow("BLB_C")

                    'FileStreamオブジェクトの生成(ファイルを開く)
                    fs = New System.IO.FileStream _
                (Server.MapPath("C.BMP"), System.IO.FileMode.OpenOrCreate,
                 System.IO.FileAccess.Write)
                    'BLOB型データのサイズを取得
                    K = UBound(MyData)
                    'BLOB型データをファイルへ書き込む
                    fs.Write(MyData, 0, K)
                    'FileStreamオブジェクトを閉じて破棄する
                    fs.Close()
                    fs = Nothing

                    '出力したファイルをImageコントロールで表示する
                    Image1.ImageUrl = "C.BMP"

                End If

            End If

            '9. コネクション切断
            con.Close()
            '10. オブジェクトの破棄
            con.Dispose()
            dst.Dispose()
            adp.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述

        End Try

    End Sub
End Class
```

C.3.5 探索条件付きSQL文での更新

ADO.NETとの連携で、探索条件付きSQL文で更新を行うサンプルコードについて説明します。

本サンプルコードは、INSERT文実行後、影響を受けた行数をWebページ上に表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcCommandオブジェクトを生成します。(CommandTextプロパティへUPDATE文を設定する)
3. OdbcCommand.Parameters.AddメソッドでOdbcParameterオブジェクトの生成と設定を行います。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名前を任意で設定します。
   OdbcParameter.Valueプロパティへパラメタの値を設定します。
   日時型のパラメタにはOdbcParameter.OdbcTypeプロパティを設定します。
4. OdbcCommand.ExecuteNonQueryメソッドでUPDATE文を実行します。
   ExecuteNonQueryメソッドより更新された行数が返されます。
5. コネクションを切断します。
6. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default

    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand

        Dim i As Integer

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcCommandオブジェクトの生成
            com = New System.Data.Odbc.OdbcCommand(
             "UPDATE TESTTBL SET DEC_C=?,DAT_C=?,CHA_C = ? WHERE KEY_C = ?", con)

            '3. OdbcParameterオブジェクトの生成と設定
            com.Parameters.AddWithValue("DEC_C", 888888.888)
            com.Parameters.AddWithValue("DAT_C", "2018-08-08")
            com.Parameters("DAT_C").OdbcType = System.Data.Odbc.OdbcType.Date
            com.Parameters.AddWithValue("CHA_C", "更新データ2018/08/08")
            com.Parameters.AddWithValue("KEY_C", 300)

            '4. UPDATE文の実行
            i = com.ExecuteNonQuery
            '更新行数を表示する
            Label1.Text = "RowCount<br>" & i & "行更新されました"

            '5. コネクション切断
            con.Close()

            '6. オブジェクトの破棄
```

```
            con.Dispose()
            com.Dispose()

            'エラー-処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"
        End Try

    End Sub
End Class
```

C.3.6 データのメモリ内キャッシュを使用した更新

DataSetオブジェクトを使用して更新を行うサンプルコードについて説明します。

OdbcCommandBuilderオブジェクトを使用した自動生成コマンドは使用できません。明示的にコマンドを設定してください。

本サンプルコードは、結果列がある場合は1行目を更新し、結果列がない場合は行を挿入します。UPDATE文またはINSERT文実行後、影響を受けた行数をWebページ上に表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcDataAdapterオブジェクトを生成します。(SelectCommandTextプロパティへSELECT文を設定します)
3. DataSetオブジェクトを生成します。
4. OdbcDataAdapter.Fillメソッドでデータを DataSetへ取得します。
5. OdbcDataAdapter.InsertCommandプロパティへINSERT文を設定します。
6. InsertCommandプロパティへ設定したパラメタに対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcDataAdapter.InsertCommand.Parameters.AddメソッドでOdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名前を任意で設定します。
   OdbcParameter.OdbcTypeプロパティを設定します。
   OdbcParameter.SourceColumnプロパティへパラメタの値(Valueプロパティ)に使用するDataSetの列名を設定します。
7. OdbcDataAdapter.UpdateCommandプロパティへUPDATE文を設定します。
8. UpdateCommandプロパティへ設定したパラメタに対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcDataAdapter.UpdateCommand.Parameters.AddメソッドでOdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名前を任意で設定します。
   OdbcParameter.OdbcTypeプロパティを設定します。
   OdbcParameter.SourceColumnプロパティへパラメタの値(Valueプロパティ)に使用するDataSetの列名を設定します。
   探索条件(WHERE句)に設定するOdbcParameter.SourceVersionプロパティにDataRowVersion.Originalを設定します。
9. データの編集をします。
   OdbcDataAdapter.Tables.Rows.Countメソッドを使用してDataSetに取り出した行数を取得します。
   行がある場合は1行目を更新対象行としてDataRowオブジェクトへ取り出します。
   行がない場合はDataSet.Tables.NewRowメソッドで新しい行を作成しDataRowオブジェクトへ取り出します。
   DataRow.BeginEditメソッドで編集モードにしてデータの編集をします
   DataRow.EndEditメソッドで編集を終了します。
   NewRowメソッドにより作成したDataRowオブジェクトをDataSet.Tables.Rows.Addメソッドで追加します。
10. OdbcDataAdapter.UpdateメソッドでDataSetの編集操作の内容により設定したINSERT文かUPDATE文を実行します。
11. DataSet.AcceptChangesメソッドでDataSetに対して行った編集内容をコミットします。
12. コネクションを切断します。

13. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim adp As System.Data.Odbc.OdbcDataAdapter
        Dim dst As DataSet
        Dim myRow As DataRow

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcDataAdapterオブジェクトの生成
            adp = New System.Data.Odbc.OdbcDataAdapter("SELECT * FROM TESTTBL", con)
            '3. DataSetオブジェクトの生成
            dst = New DataSet
            '4. DataSet オブジェクトへデータを取得
            adp.Fill(dst, "TESTTBL")

            '5. OdbcDataAdapter.InsertCommandプロパティの設定
            adp.InsertCommand = New System.Data.Odbc.OdbcCommand(
              "INSERT INTO TESTTBL VALUES(?,?,?,?)", con)
            '6. OdbcParameterオブジェクトの生成と設定
            adp.InsertCommand.Parameters.Add("KEY_C", System.Data.Odbc.OdbcType.Int)
            adp.InsertCommand.Parameters("KEY_C").SourceColumn = "KEY_C"
            adp.InsertCommand.Parameters.Add("DEC_C",
              System.Data.Odbc.OdbcType.Decimal)
            adp.InsertCommand.Parameters("DEC_C").SourceColumn = "DEC_C"
            adp.InsertCommand.Parameters.Add("DAT_C", System.Data.Odbc.OdbcType.Date)
            adp.InsertCommand.Parameters("DAT_C").SourceColumn = "DAT_C"
            adp.InsertCommand.Parameters.Add("CHA_C", System.Data.Odbc.OdbcType.Char)
            adp.InsertCommand.Parameters("CHA_C").SourceColumn = "CHA_C"

            '7. OdbcDataAdapter.UpdateCommandプロパティの設定
            adp.UpdateCommand = New System.Data.Odbc.OdbcCommand(
             "UPDATE TESTTBL SET KEY_C=?, DEC_C=?, DAT_C=?, CHA_C=? WHERE KEY_C=?", con)
            '8. OdbcParameterオブジェクトの生成と設定
            adp.UpdateCommand.Parameters.Add("KEY_C", System.Data.Odbc.OdbcType.Int)
            adp.UpdateCommand.Parameters("KEY_C").SourceColumn = "KEY_C"
            adp.UpdateCommand.Parameters.Add("DEC_C",System.Data.Odbc.OdbcType.Decimal)
            adp.UpdateCommand.Parameters("DEC_C").SourceColumn = "DEC_C"
            adp.UpdateCommand.Parameters.Add("DAT_C", System.Data.Odbc.OdbcType.Date)
            adp.UpdateCommand.Parameters("DAT_C").SourceColumn = "DAT_C"
            adp.UpdateCommand.Parameters.Add("CHA_C", System.Data.Odbc.OdbcType.Char)
            adp.UpdateCommand.Parameters("CHA_C").SourceColumn = "CHA_C"
            adp.UpdateCommand.Parameters.Add("KEY", System.Data.Odbc.OdbcType.Int)
            adp.UpdateCommand.Parameters("KEY").SourceColumn = "KEY_C"
            adp.UpdateCommand.Parameters("KEY").SourceVersion = DataRowVersion.Original

            '9. データの更新(DataSetに1行もデータがない場合は行を追加する)
            If Not dst.Tables("TESTTBL").Rows.Count = 0 Then
                myRow = dst.Tables("TESTTBL").Rows(0)
            Else
                myRow = dst.Tables("TESTTBL").NewRow
            End If
```

```
            myRow.BeginEdit()
            myRow("KEY_C") = 200
            myRow("DEC_C") = 9999999.999
            myRow("DAT_C") = "2018-08-08"
            myRow("CHA_C") = "TEST DATA"
            myRow.EndEdit()

            If dst.Tables("TESTTBL").Rows.Count = 0 Then
                dst.Tables("TESTTBL").Rows.Add(myRow)
            End If

            '10. INSERT文/UPDATE文の実行
            adp.Update(dst, "TESTTBL")
            '11. DataSet オブジェクトに対して行われたすべての編集をコミットする
            dst.AcceptChanges()

            'メッセージ表示
            Label1.Text = "Normal End<br>データベースにデータが保存されました"

            '12. コネクション切断
            con.Close()

            '13. オブジェクトの破棄
            con.Dispose()
            dst.Dispose()
            adp.Dispose()

            'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"
        End Try

    End Sub
End Class
```

C.3.7 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をWebページ上に表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトでコネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcCommandオブジェクトを生成します。(CommandTextプロパティへCALL文を設定します)
3. CommandTextプロパティへ設定したパラメタマーカ('?')に対応するOdbcParameterオブジェクトの生成と設定をします。
   OdbcCommand.Parameters.AddメソッドでOdbcParameterオブジェクトを生成します。SQL文中のパラメタマーカ('?')の出現順に生成してください。
   OdbcParameter.ParameterNameプロパティには、オブジェクト名前を任意で設定します。
   OdbcParameter.Directionプロパティへ入力(Input)、出力(Output)、入出力(InputOutput)を設定します。
   OdbcParameter.Valueプロパティへパラメタの値を設定します。
   文字列型の出力パラメタおよび入出力パラメタのOdbcParameter.Sizeプロパティには、最大サイズをバイト数で設定します。
4. OdbcCommand.ExecuteNonQueryメソッドでCALL文を実行します。

5. 出力および入出力パラメタの結果をメッセージボックスで表示します。
   OdbcParameter.Valueプロパティで結果が参照できます。
   このサンプルコードでは、出力パラメタごとに改行し表示します。

6. コネクションを切断します。

7. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand

        Dim msgstr As String

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. OdbcCommandオブジェクトの生成
            com = New System.Data.Odbc.OdbcCommand("CALL COUNTPRC(?,?,?,?)", con)

            '3. OdbcParameterオブジェクトの生成と設定
            com.Parameters.Add("DAT_P", System.Data.Odbc.OdbcType.Date)
            com.Parameters("DAT_P").Direction = ParameterDirection.Input
            com.Parameters("DAT_P").Value = "2018-08-08"
            com.Parameters.Add("OUT_P", System.Data.Odbc.OdbcType.Int)
            com.Parameters("OUT_P").Direction = ParameterDirection.Output
            com.Parameters.Add("PRCSTATE", System.Data.Odbc.OdbcType.Char)
            com.Parameters("PRCSTATE").Direction = ParameterDirection.Output
            com.Parameters("PRCSTATE").Size = 5
            com.Parameters.Add("PRCMSG", System.Data.Odbc.OdbcType.Char)
            com.Parameters("PRCMSG").Direction = ParameterDirection.Output
            com.Parameters("PRCMSG").Size = 255

            '4. CALL文実行
            com.ExecuteNonQuery()

            '5. 結果の表示
            msgstr = "COUNT: " & com.Parameters("OUT_P").Value & "<br>"
            msgstr &= "PRCSTATE: " & com.Parameters("PRCSTATE").Value & "<br>"
            msgstr &= "PRCMSG: " & com.Parameters("PRCMSG").Value
            Label1.Text = "PROCEDURE DATA<br>" & msgstr

            '6. コネクション切断
            con.Close()

            '7. オブジェクトの破棄
            con.Dispose()
            com.Dispose()

        'エラー処理
        Catch ex As System.Data.Odbc.OdbcException

            'エラー処理ルーチンを記述
            Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"
```

```
        End Try

    End Sub
End Class
```

C.3.8 トランザクション制御

ADO.NETとの連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

OdbcConnection.BeginTransactionメソッド、OdbcTransaction.CommitメソッドまたはOdbcTransaction.Rollbackメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードの設定または変更は、SET TRANSACTION文で行います。

独立性水準の設定または変更は、OdbcConnection.BeginTransactionメソッドの引数IsolationLevelで指定する、またはSET TRANSACTION文で行います。SET TRANSACTION文で設定または変更を行う場合は、OdbcConnection.BeginTransactionメソッド実行時に引数IsolationLevelの設定値に変更されるため、OdbcConnection.BeginTransactionメソッド実行後に行う必要があります。このため、引数IsolationLevelによる設定または変更を推奨します。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定後、INSERT文を実行します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.3.1 接続および切断”を参照してください。
2. OdbcCommand.ExecuteNonQueryメソッドでSET TRANSACTION文を実行しアクセスモードを設定します。
3. OdbcConnection.BeginTransactionメソッドでトランザクションを開始します。
   引数IsolationLevelに独立性水準(IsolationLevel.ReadCommitted)を設定します。
   OdbcTransactionオブジェクトが作成されます。
4. 作成されたOdbcTransactionオブジェクトを、データ操作を行うオブジェクト(OdbcCommand)のTransactionプロパティに設定し実行します。
5. データ操作が正常終了した場合、OdbcTransaction.Commitメソッドを実行します。
   データ操作が異常終了した場合、OdbcTransaction.Rollbackメソッドを実行します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.3.9 エラー処理”を参照してください。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection
        Dim com As System.Data.Odbc.OdbcCommand
        Dim trn As System.Data.Odbc.OdbcTransaction

        'OdbcConnectionオブジェクトの生成
        con = New System.Data.Odbc.OdbcConnection("DSN=DSN01;UID=USER01;PWD=PASS01")

        Try
            '1. コネクション接続
            con.Open()

            '2. アクセスモードの設定
            com = New System.Data.Odbc.OdbcCommand("SET TRANSACTION READ WRITE", con)
```

```
            com.ExecuteNonQuery()

            '3. トランザクション開始(独立性水準の設定)
            trn = con.BeginTransaction(IsolationLevel.ReadCommitted)

            '4. データ操作を行う処理を記述
            com.CommandText = "INSERT INTO TESTTBL(KEY_C,DEC_C,DAT_C,CHA_C)
              VALUES(100,100.111,DATE '2018-08-08','TRAN DATA 20180808-1')"
            'OdbcCommand.Transactionプロパティに生成したOdbcTransactionオブジェクトを設定
            com.Transaction = trn
            com.ExecuteNonQuery()

            '5. コミット
            trn.Commit()

            '6. コネクション切断
            con.Close()
            '7. オブジェクトの破棄
            con.Dispose()

        'エラー処理
    Catch ex As System.Data.Odbc.OdbcException
        '5. ロールバック
        trn.Rollback()

        'エラー処理ルーチンを記述
        Label1.Text = "No Connect<br>▲▼  COMMUNICATION OFF LINE  ▲▼"
    End Try

End Sub
End Class
```

C.3.9 エラー処理

エラー処理を行うサンプルコードについて説明します。

TryステートメントからCatchステートメントの範囲内にエラーが発生する可能性のあるコードを記述し、Catchステートメントでエラー発生時のエラー処理を記述します。ADO.NETを使用してODBCエラーが発生した場合、OdbcExceptionクラスをキャッチすることで、SQLSTATEやエラーメッセージなどが確認できます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

ODBC以外のエラーは、Exceptionクラスをキャッチすることでエラー情報が確認できます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio .NET ドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。発生したエラーのメッセージとSQLSTATEをWebページ上に表示します。

アプリケーションの手順

1. OdbcConnectionオブジェクトを生成します。(ConnectionStringプロパティに接続文字列を間違えて設定する)
2. TryステートメントからCatchステートメントの範囲内にエラーが発生する可能性のあるコードを記述します。
3. OdbcConnection.Openメソッドでコネクションを接続します。(パスワードを間違えて指定しエラーを発生させます)
4. Catchステートメントに処理したいエラーのクラスを指定します。
   OdbcException.Errors.CountプロパティでODBCのエラー数を取得します。
   OdbcException.Errors.MessageプロパティでODBCのエラーメッセージを取得します。

OdbcException.Errors.SQLStateプロパティにでSQLSTATEを取得します。
Exception.Messageプロパティ でODBC以外のエラーメッセージを取得します。

5. 処理がすべて行われた後に行う処理を、Finallyステートメント以降に記述します。

```
Public Class _Default
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        'オブジェクト宣言
        Dim con As System.Data.Odbc.OdbcConnection

        Dim i As Integer
        Dim msgstr As String

        '1. OdbcConnectionオブジェクトの生成(コネクション文字列を間違えて設定)
        con = New System.Data.Odbc.OdbcConnection("DSN=checkRDB;UID=ADMINISTRATOR;PWD=XXXXXX")

        '2. Try ～ Catch の範囲内にエラーが発生する可能性のあるコードを記述
        Try


            '3. コネクション接続(エラー発生)
            con.Open()

            'データ処理を記述

            'コネクション切断
            con.Close()

            '4. OdbcExceptionクラスのキャッチ
        Catch ex As System.Data.Odbc.OdbcException
            For i = 0 To ex.Errors.Count - 1
                msgstr &= ex.Errors(i).Message & "<br>"
                msgstr &= "SQLSTATE: " & ex.Errors(i).SQLState & "<br>"
            Next
            Label1.Text = "ODBC Error Message<br>" & msgstr

            '4. Exceptionクラスのキャッチ
        Catch ex As Exception
            msgstr = ex.Message
            Label1.Text = "Error Message<br>" & msgstr

            '5. Tryステートメント終了時の処理
        Finally

            'オブジェクトの破棄
            con.Dispose()

        End Try

    End Sub
End Class
```

C.4 ASPでのADOのサンプル

ASPでのADOのサンプルコードおよび留意事項について説明します。

C.4.1 接続および切断

接続および切断をするサンプルコードについて説明します。

本サンプルコードは、接続後Webページ上に接続できたことを表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   Connection.ConnectionStringプロパティへ接続文字列を設定します。
2. Connection.Openメソッドでコネクションを接続します。
3. Connection.Closeメソッドでコネクションを切断します。
4. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<html>
<head>
<title>接続/切断</title>
</head>
<body bgcolor="White">

<b>接続/切断</b>
<hr>

<%
'宣言
Dim OBJdbConnection

'1. Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'2. コネクション接続
OBJdbConnection.Open

Response.Write "Connect<br>接続できました"

'3. コネクション切断
OBJdbConnection.Close

'4. オブジェクトの破棄
Set OBJdbConnection = Nothing

%>

<hr>

</body>
</html>
```

C.4.2 データの参照

ADOでの連携でデータの参照を行う方法について説明します。

本サンプルコードは、取得データをWebページ上にて表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.4.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。

4. Recordsetオブジェクトよりデータを取得します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Fields.Countプロパティで列数を確認します。
   Field.Valueプロパティでデータを取得します。
   Recordset.MoveNextメソッドでカレント行を次の行へと移動します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

注意

Recordset.EOFプロパティでカレント行がEOFかどうか確認後は、以下のエラーがアプリケーション側で認識できなくなります。

- JYP2099E デッドロックが発生しました.
- JYP5014E スキーマ“@1@”の表“@2@”は占有中です.

これらのエラーをアプリケーション側で認識するタイミングは、Recordset.MoveFirstメソッドを実行するか、データを参照した時です。上記エラーを認識するためには、EOFプロパティを確認せずにMoveFirstメソッドの実行またはデータを参照してください。

ただし、SELECT文の検索結果が0件であった場合、Visual Basicの実行時エラー“3021”が発生します。エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>データの参照</title>
</head>
<body bgcolor="White">

<b>データの参照</b>

<hr>

<%
'宣言
Dim OBJdbConnection
Dim RecordSetObj

Dim msgstr

'Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'1. コネクション接続
OBJdbConnection.Open

'2. Recordsetオブジェクトの生成
Set RecordSetObj = Server.CreateObject("ADODB.Recordset")

'3. Recordsetオブジェクトを開く
RecordSetObj.Open "SELECT * FROM TESTTBL", OBJdbConnection, _
                  adOpenForwardOnly, _
                  adLockReadOnly, _
                  adCmdText
```

```
'4. データの取得'最終行まで繰り返し
Do Until RecordSetObj.EOF
    '列数の取得
    For i = 0 To RecordSetObj.Fields.Count - 1
        'データの取得
        msgstr = msgstr & RecordSetObj.Fields(i).Value & " "
    Next
    msgstr = msgstr & "<br>"
    '行の位置づけ
    RecordSetObj.MoveNext
Loop

'取得データの表示
Response.Write "Recordset<br>" & msgstr

'5. Recordsetオブジェクトを閉じる
RecordSetObj.Close

'6. コネクション切断
OBJdbConnection.Close

'7. オブジェクトの破棄
Set RecordSetObj = Nothing
Set OBJdbConnection = Nothing

%>

<hr>

</body>
</html>
```

C.4.3 BLOB型データの参照

BLOB型データを参照するサンプルコードについて説明します。

本サンプルコードは、GetChunkメソッドで16384バイトずつBLOB型データを取得します。取得したBLOB型データはResponse.BinaryWriteメソッドを使用してWebページに表示します。なお、本サンプルではGIFを表示するため、BLOB型データは予めVisual Basicなどを利用して登録してください。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.4.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. Recordsetオブジェクトよりデータを取得します。
   Recordset.BOFプロパティでカレント行がBOFかを確認します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Field.ActualSizeプロパティで取得データのバイト数を確認します。
   Field.GetChunkメソッドで指定バイト数分データを取得します。
5. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
```

```
<%
'宣言
Dim OBJdbConnection
Dim RecordSetObj

Dim Chunks
Dim Fragment
Const ChunkSize = 16384
Dim I

'Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'1. コネクション接続
OBJdbConnection.Open

'2. Recordsetオブジェクトの生成
Set RecordSetObj = Server.CreateObject("ADODB.Recordset")

'3. Recordsetオブジェクトを開く
RecordSetObj.Open "SELECT * FROM IMGTBL", OBJdbConnection, _
                  adOpenForwardOnly, _
                  adLockReadOnly, _
                  adCmdText

If NOT (RecordSetObj.BOF AND RecordSetObj.EOF) Then

    '4. データの取得
    'BLOB型データのサイズを取得
    Fl = RecordSetObj.Fields("BLB_C").ActualSize

    'BLB_C列にデータがあるか
    If Fl > 0 Then
        '規定量にデータを分けて取得するため、分割数と残量を求める
        Chunks = Fl ¥ ChunkSize
        Fragment = Fl Mod ChunkSize

        'GIFデータであることを指定する
        'Response.ContentType = "image/gif"

        '残量分のデータを取得
        'Response.BinaryWrite RecordSetObj.Fields("BLB_C").GetChunk(Fragment)

        '分割数分繰り返し
        For i = 1 To Chunks
            '規定量分のデータ取得
            'Response.BinaryWrite RecordSetObj.Fields("BLB_C").GetChunk(ChunkSize)
        Next

    End If

End If

'5. Recordsetオブジェクトを閉じる
RecordSetObj.Close

'6. コネクション切断
OBJdbConnection.Close

'7. オブジェクトの破棄
Set RecordSetObj = Nothing
Set OBJdbConnection = Nothing
```

```
%>
```

C.4.4 データの挿入

ADOとの連携でデータの挿入を行う方法について説明します。

本サンプルコードは、INSERT文実行後、Webページ上で完了を知らせます。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.4.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにINSERT文を設定します。
   Command.CommandTypeプロパティにadCmdTextを設定します。
3. Command.ExecuteメソッドでINSERT文を実行します。
4. コネクションを切断します。
5. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>データの挿入</title>
</head>

<body bgcolor="White">

<b>データの挿入</b>
<hr>

<%
'宣言
Dim OBJdbConnection
Dim OBJdbCommand

'Connectionオブジェクトの生成
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"
'1. コネクション接続
OBJdbConnection.Open

'2. Commandオブジェクトの生成と設定
Set OBJdbCommand = Server.CreateObject("ADODB.Command")
OBJdbCommand.ActiveConnection = OBJdbConnection
OBJdbCommand.CommandText = _
"INSERT INTO TESTTBL VALUES(401,1000.025,DATE'2007/04/10','INSERT DATA')"
OBJdbCommand.CommandType = adCmdText

'3. INSERT文実行
OBJdbCommand.Execute

'メッセージの表示
Response.Write "行を挿入しました<br>"

'4. コネクション切断
OBJdbConnection.Close
```

```
'5. オブジェクトの破棄
Set OBJdbCommand = Nothing
Set OBJdbConnection = Nothing

%>

<hr>
</body>
</html>
```

C.4.5 パラメタマーカを使用したSQL文での更新

パラメタマーカ('?')を使用したSQL文でのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、UPDATE文実行後、影響を受けた行数をWebページ上に表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、"C.4.1 接続および切断"を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにUPDATE文を設定します。
   Command.CommandTypeプロパティにCommandTypeEnum.adCmdTextを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名前を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、パラメタが入力パラメタ(adParamInput)であることを設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
   Parameter.Valueプロパティには、値を設定します。
   Parameter.Precisionプロパティには、精度を設定します。
   Parameter.NumericScaleプロパティには、値の小数点以下の桁数を設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。
5. Command.ExecuteメソッドでUPDATE文を実行します。
   引数RecordsAffectedを指定しExecuteメソッドで影響を受けた行数を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、"C.4.10 エラー処理"を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>パラメタマーカ('?')を使用したSQL文での更新</title>
</head>
<body bgcolor="White">

<b>パラメタマーカ('?')を使用したSQL文での更新</b>
<hr>

<%
'宣言
Dim OBJdbConnection
Dim OBJdbCommand
Dim OBJdbParameter1,OBJdbParameter2,OBJdbParameter3,OBJdbParameter4
Dim RecordsAffected
```

```
'Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'1. コネクション接続
OBJdbConnection.Open

'2. Commandオブジェクトの生成と設定
Set OBJdbCommand = Server.CreateObject("ADODB.Command")
OBJdbCommand.ActiveConnection = OBJdbConnection
OBJdbCommand.CommandText = "UPDATE TESTTBL SET DEC_C=?,DAT_C=?,CHA_C=? WHERE KEY_C=?"
OBJdbCommand.CommandType = adCmdText

'3. Parameterオブジェクトの生成と設定
Set OBJdbParameter1 = OBJdbCommand.CreateParameter
OBJdbParameter1.Name = "Pm1"
OBJdbParameter1.Type = adNumeric
OBJdbParameter1.Value = "2000.222"
OBJdbParameter1.Precision = 10
OBJdbParameter1.NumericScale = 3

Set OBJdbParameter2 = OBJdbCommand.CreateParameter
OBJdbParameter2.Name = "Pm2"
OBJdbParameter2.Type = adDate
OBJdbParameter2.Value = "2007/04/10"

Set OBJdbParameter3 = OBJdbCommand.CreateParameter
OBJdbParameter3.Name = "Pm3"
OBJdbParameter3.Type = adChar
OBJdbParameter3.Value = "UPDATE DATA"
OBJdbParameter3.Size = 250

Set OBJdbParameter4 = OBJdbCommand.CreateParameter
OBJdbParameter4.Name = "Pm4"
OBJdbParameter4.Type = adInteger
OBJdbParameter4.Value = "401"

'4. ParametersコレクションにParameterオブジェクトを追加
OBJdbCommand.Parameters.Append OBJdbParameter1
OBJdbCommand.Parameters.Append OBJdbParameter2
OBJdbCommand.Parameters.Append OBJdbParameter3
OBJdbCommand.Parameters.Append OBJdbParameter4

'5. UPDATE文実行
OBJdbCommand.Execute RecordsAffected

'メッセージの表示
Response.Write RecordsAffected
Response.Write "行更新されました<br>"

'6. コネクション切断
OBJdbConnection.Close

'7. オブジェクトの破棄
Set OBJdbParameter1 = Nothing
Set OBJdbParameter2 = Nothing
Set OBJdbParameter3 = Nothing
Set OBJdbParameter4 = Nothing
Set OBJdbCommand = Nothing
Set OBJdbConnection = Nothing

%>
```

```
<hr>

</body>
</html>
```

C.4.6 カーソル位置づけでのデータ更新

カーソル位置づけでのデータ更新を行うサンプルコードについて説明します。

本サンプルコードは、Recordsetオブジェクトを使用してデータ更新を行います。Recordsetオブジェクトに行が1行もない場合には行を挿入し、1行以上ある場合には1行目を更新します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.4.1 接続および切断”を参照してください。
2. Recordsetオブジェクトを生成します。
3. Recordset.OpenメソッドでRecordsetオブジェクトを開きます。
4. データを編集します。(行がない場合は行を挿入します)
   Recordset.BOFプロパティでカレント行がBOFかを確認します。
   Recordset.EOFプロパティでカレント行がEOFかを確認します。
   Recordset.AddNewメソッドでRecordsetオブジェクトに新しい行を作成します。
   Field.Valueプロパティへ値を設定します。
5. Recordset.UpdateメソッドでRecordsetオブジェクトの編集内容によりINSERT文かUPDATE文を実行します。
6. Recordset.CloseメソッドでRecordsetオブジェクトを閉じます。
7. コネクションを切断します。
8. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>カーソル位置づけでのデータ更新</title>
</head>
<body bgcolor="White">
<b>カーソル位置づけでのデータ更新</b>

<hr>

<%
'宣言
Dim OBJdbConnection
Dim RecordSetObj

'Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'1. コネクション接続
OBJdbConnection.Open

'1. Connectionオブジェクトの生成と設定

'2. Recordsetオブジェクトの生成
Set RecordSetObj = Server.CreateObject("ADODB.Recordset")
```

```
'3. Recordsetオブジェクトを開く
RecordSetObj.Open "SELECT * FROM TESTTBL",OBJdbConnection, _
                  adOpenStatic, _
                  adLockPessimistic, _
                  adCmdText

'4. データの編集(rdoResultsetに1行もデータがない場合は行を追加する)
If RecordSetObj.BOF And RecordSetObj.EOF Then
    RecordSetObj.AddNew
End If

RecordSetObj.Fields(0).Value = 500
RecordSetObj.Fields(1).Value = 11111.111
RecordSetObj.Fields(2).Value = "2007/04/10"
RecordSetObj.Fields(3).Value = "UPDATE DATA"

'5. INSERT文/UPDATE文の実行
RecordSetObj.Update

'メッセージの表示
Response.Write "行を更新しました<br>"

'6. Recordsetオブジェクトを閉じる
RecordSetObj.Close

'7. コネクション切断
OBJdbConnection.Close

'8. オブジェクトの破棄
Set RecordSetObj = Nothing
Set OBJdbConnection = Nothing

%>

<hr>
</body>
</html>
```

C.4.7 ストアドプロシジャの実行

ストアドプロシジャを実行するサンプルコードについて説明します。

本サンプルコードは、ストアドプロシジャを実行し結果をWebページ上に表示します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.4.1 接続および切断”を参照してください。
2. Commandオブジェクトを生成します。
   Command.ActiveConnectionプロパティに手順1で生成したConnectionオブジェクトを設定します。
   Command.CommandTextプロパティにストアドプロシジャ名を設定します。
   Command.CommandTypeプロパティにadCmdStoredProcを設定します。
3. Parameterオブジェクトの生成と設定をします。
   Parameter.Nameプロパティには、オブジェクト名前を任意で設定します。
   Parameter.Typeプロパティには、データ型を設定します。
   Parameter.Directionプロパティには、入力パラメタ(adParamInput)、出力パラメタ(adParamOutput)のいずれかであることを設定します。
   入力パラメタのParameter.Valueプロパティには値を設定します。
   文字列型のParameter.Sizeプロパティには、データの最大サイズをバイト数で設定します。
4. ParametersコレクションにParameterオブジェクトを追加します。SQL文中のパラメタマーカ('?')の出現順に追加してください。

5. Command.Executeメソッドでストアドプロシジャを実行します。
   ストアドプロシジャの結果を、出力パラメタのParameter.Valueプロパティで結果を取得します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>ストアドプロシジャの実行</title>
</head>
<body bgcolor="White">
<b>ストアドプロシジャの実行</b>

<hr>

<%
'宣言
Dim OBJdbConnection
Dim OBJdbCommand
Dim OBJdbParameter1,OBJdbParameter2,OBJdbParameter3,OBJdbParameter4

'Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'1. コネクション接続
OBJdbConnection.Open

'2. Commandオブジェクトの生成と設定

Set OBJdbCommand = Server.CreateObject("ADODB.Command")
OBJdbCommand.ActiveConnection = OBJdbConnection
OBJdbCommand.CommandText = "COUNTPRC"
OBJdbCommand.CommandType = adCmdStoredProc

'3. Parameteオブジェクトの生成と設定
Set OBJdbParameter1 = OBJdbCommand.CreateParameter
OBJdbParameter1.Name = "DAT_P"
OBJdbParameter1.Type = adDate
OBJdbParameter1.Value = "2007/04/10"
OBJdbParameter1.Direction = adParamInput

Set OBJdbParameter2 = OBJdbCommand.CreateParameter
OBJdbParameter2.Name = "OUT_P"
OBJdbParameter2.Type = adInteger
OBJdbParameter2.Direction = adParamOutput

Set OBJdbParameter3 = OBJdbCommand.CreateParameter
OBJdbParameter3.Name = "PRCSTATE"
OBJdbParameter3.Type = adChar
OBJdbParameter3.Size = 5
OBJdbParameter3.Direction = adParamOutput

Set OBJdbParameter4 = OBJdbCommand.CreateParameter
OBJdbParameter4.Name = "PRCMSG"
OBJdbParameter4.Type = adChar
OBJdbParameter4.Size = 255
OBJdbParameter4.Direction = adParamOutput
```

```
'4. Parametersコレクションにパラメータオブジェクトを追加
OBJdbCommand.Parameters.Append OBJdbParameter1
OBJdbCommand.Parameters.Append OBJdbParameter2
OBJdbCommand.Parameters.Append OBJdbParameter3
OBJdbCommand.Parameters.Append OBJdbParameter4

'5. ストアドプロシジャの実行
OBJdbCommand.Execute

'結果を表示
Response.Write "PROCEDURE DATA<br>"
Response.Write "COUNT: " & OBJdbCommand.Parameters(1).Value & "<br>"
Response.Write "PRCSTATE: " & OBJdbCommand.Parameters(2).Value & "<br>"
Response.Write "PRCMSG: " & OBJdbCommand.Parameters(3).Value & "<br>"

'6. コネクション切断
OBJdbConnection.Close

'7. オブジェクトの破棄
Set OBJdbParameter1 = Nothing
Set OBJdbParameter2 = Nothing
Set OBJdbParameter3 = Nothing
Set OBJdbParameter4 = Nothing
Set OBJdbCommand = Nothing
Set OBJdbConnection = Nothing

%>

<hr>

</body>
</html>
```

C.4.8 トランザクション制御

ADOとの連携でトランザクション制御を行うサンプルコードについて説明します。

トランザクション制御は、明示的に行うことができます。

Connection.BeginTransメソッド、Connection.CommitTransメソッドまたは Connection.RollbackTransメソッドで明示的に行うことができます。これらのメソッドでトランザクションを制御していない場合は、トランザクションはSQL文を実行することで開始され、そのSQL文の処理完了後すぐにコミットされて終了します。

アクセスモードの設定または変更は、SET TRANSACTION文または Connection.Modeプロパティで行います。Connection.Modeプロパティはコネクションが切れている時だけ設定または変更することができます。コネクション接続中に変更が可能なSET TRANSACTION文による設定または変更を推奨します。

独立性水準の設定または変更は、Connection.IsolationLevelプロパティまたはSET TRANSACTION文で行います。SET TRANSACTION文で設定または変更を行う場合は、Connection.BeginTransメソッド実行時にConnection.IsolationLevelプロパティ設定値に変更されるため、Connection.BeginTransメソッド実行後に行う必要があります。このため、Connection.IsolationLevelプロパティによる設定または変更を推奨します。

本サンプルコードは、アクセスモードをREAD WRITE、独立性水準をREAD COMMITTEDに設定します。

アプリケーションの手順

1. コネクションを接続します。詳細は、“C.4.1 接続および切断”を参照してください。
2. Connection.ExecuteメソッドでSET TRANSACTION文を実行しアクセスモードを設定します。
3. Connection.IsolationLevelプロパティ独立性水準(adXactReadCommitted)を設定します。
4. Connection.BeginTransメソッドでトランザクションを開始します。

5. データ操作が正常終了した場合、Connection.CommitTransメソッドを実行します。
   データ操作が異常終了した場合、Connection.RollbackTransメソッドを実行します。
6. コネクションを切断します。
7. オブジェクトを破棄します。

エラー処理については、“C.4.10 エラー処理”を参照してください。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>トランザクション制御</title>
</head>
<body bgcolor="White">
<b>トランザクション制御</b>

<hr>

<%
'宣言
Dim OBJdbConnection

'エラーを取得するためエラー発生時に処理を止めないようにする
On Error Resume Next

'Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=PASS01"

'1. コネクション接続
OBJdbConnection.Open

'2. アクセスモードの設定
OBJdbConnection.Execute "SET TRANSACTION READ WRITE", , adCmdText

'3. IsolationLevelの設定
OBJdbConnection.IsolationLevel = adXactReadCommitted

'4. トランザクション開始
OBJdbConnection.BeginTrans

'データ操作を行う処理を記述

If OBJdbConnection.Error.Count = 0 Then
    '5. コミット(正常終了時)
    OBJdbConnection.CommitTrans()
Else
    '5. ロールバック(異常終了時)
    OBJdbConnection.RollbackTrans()
End If

'6. コネクション切断
OBJdbConnection.Close

'7. オブジェクトの破棄
Set OBJdbConnection = Nothing

%>

<hr>
</body>
</html>
```

C.4.9 コネクションプーリング

ODBCドライバマネージャのプーリング機能を利用するパラメタを接続文字列に記載する必要があります。

以下に、ASPのADOからプーリング機能を利用する場合のサンプルプログラムを示します。

```
<html>
<head></head>
<body>
<p>サーバの現在の日付は<%= Date%>時刻は<%= Time %>です。</p>
<%
Server.ScriptTimeOut = 3600

str = "DSN=DSN01;UID=USER01;PWD=PASS01; "
strOLE = "OLE DB Services=-2; "          'ODBCコネクションプーリングを使用します。
str = str + strOLE

For i=1 To 10
    Set c = Server.CreateObject("ADODB.Connection")
    c.Open str
    '切断要求されたコネクションは、プールに保存されます。
    c.close
    Set c = Nothing
Next
%>
<p>サーバの現在の日付は<%= Date%>時刻は<%= Time %>です。</p>
</body>
</html>
```

C.4.10 エラー処理

エラー処理を行うサンプルコードについて説明します。

On Errorステートメントで、エラー発生時のエラー処理ルーチンを指定します。ADOを使用してODBCエラーが発生した場合、ErrorオブジェクトにSQLSTATEやエラーメッセージなどが格納されます。SQLSTATEやエラーメッセージをキーにしてエラー処理を切り分け、その後の振る舞いを決めることが可能です。

参照

エラーメッセージの対処方法は“メッセージ集”を参照してください。

Symfoware/RDBで利用者の認証情報を管理している場合(SET SYSTEM PARAMETER文で“USER_CONTROL=YES”を指定)、正常終了に“[Symfoware ODBC Driver][Symfoware Server]JYP2605I 正常終了しました. 接続日時: 前回接続日時:”を返却します。

利用者は、この接続日時を記憶しておき、次回の接続時に参照し不正なアクセスがないかを確認する必要があります。

なお、上記メッセージが通知されれることで、ODBCドライバマネージャが各種ODBCドライバのAPIサポートレベルをチェックする際に発生するメッセージ“[Microsoft][ODBCDriver Manager]ドライバの SQLSetConnectAttr は失敗しました”を、正常終了ですが返却します。

このODBCドライバマネージャから返却される不要なメッセージを表示したくない場合には、SQLSTATE(IM006)で判断して表示しない処理を行ってください。

ODBC以外のエラーは、Errオブジェクトにエラー情報が格納されます。ODBC以外のエラーの対処方法は、Microsoft Visual Studio .NET ドキュメントを参照してください。

本サンプルコードでは、接続文字列を間違えて入力しエラーを発生させます。発生したエラーのメッセージとSQLSTATEをメッセージボックスに表示します。

アプリケーションの手順

1. Connectionオブジェクトを生成します。
   (Connection. ConnectionStringプロパティに接続文字列を間違えて設定する)
2. On Errorステートメントを使用して、エラー処理ルーチンを設定します。
3. Connection.Openメソッドでコネクションを接続します。
   (パスワードを間違えて指定しエラーを発生させます)
4. エラー処理ルーチンを記述します。
   Errors.Countプロパティで格納されているErrorオブジェクト数を取得します。
   Error.DescriptionプロパティでODBCのエラーメッセージを取得します。
   Error.SQLStateプロパティでSQLSTATEを取得します。
   Error.ClearメソッドでErrorオブジェクトを削除します。

```
<%@ language="vbscript" %>
<!-- #include file = "adovbs.inc" -->
<html>
<head>
<title>エラー処理</title>
</head>
<body bgcolor="White">
<b>エラー処理</b>

<hr>

<%
'宣言
Dim OBJdbConnection

'エラーを取得するためエラー発生時に処理を止めないようにする
On Error Resume Next

'1. Connectionオブジェクトの生成と設定
Set OBJdbConnection = Server.CreateObject("ADODB.Connection")
OBJdbConnection.ConnectionString = "DSN=DSN01;UID=USER01;PWD=XXXXXX"

'3. コネクション接続
OBJdbConnection.Open

'4. エラー処理
Err_Count = OBJdbConnection.Errors.Count
If Err_Count > 0 Then
    For i = 0 To Err_Count - 1
        msgstr = msgstr & OBJdbConnection.Errors(i).Description & "<br>"
        msgstr = msgstr & "SQLSTATE: " & OBJdbConnection.Errors(i).SQLState & "<br>"
    Next
    Response.Write "Error<br>" & msgstr
    OBJdbConnection.Errors.Clear
Else
    '5. コネクション切断
    OBJdbConnection.Close
End If

'オブジェクトの破棄
Set OBJdbConnection = Nothing

%>

<hr>
</body>
</html>
```

付録D　Symfoware/RDBのデータベースの資源と命名規約

パソコンよりデータ定義文あるいはデータ操作文を実行する場合、またはSymfoware/RDBのデータベースの運用操作コマンドを実行する場合、Symfoware/RDBが自動的に生成する命名規約を意識する必要があります。

以下に、ODOSを利用して定義するSymfoware/RDBのデータベースの各種資源に対する命名規約について説明します。

データベーススペース名に対する命名規約

パソコンから表定義またはインデックス定義を行う場合は、データベース名と同じ名前の空きのあるデータベーススペースを用意しておく必要があります。なお、データベース簡単運用の場合は、データベーススペースを考慮する必要がありません。

スキーマ名および表名に対する命名規約

パソコンから実行するスキーマ定義文などで指定するスキーマ名および表名は、通常はそれぞれ8文字以内で指定します。

なお、スキーマ名および表名の文字数を拡張したい場合は、システム用の動作環境ファイルの“DEFAULT_DSI_NAME”を指定してください。データベース簡単運用の場合は、システム用の動作環境ファイルによる指定はできません。

参照

“DEFAULT_DSI_NAME”の詳細は、“セットアップガイド”を参照してください。

スキーマ名およびインデックス名に対する命名規約

パソコンから実行するスキーマ定義文などで指定するスキーマ名およびインデックス名は、通常はそれぞれ8文字以内で指定します。

なお、スキーマ名およびインデックス名の文字数を拡張したい場合は、システム用の動作環境ファイルの“DEFAULT_DSI_NAME”を指定してください。データベース簡単運用の場合は、システム用の動作環境ファイルによる指定はできません。

参照

“DEFAULT_DSI_NAME”の詳細は、“セットアップガイド”を参照してください。

表のDSO名およびDSI名に対する命名規約

参照

表のDSO名およびDSI名に対する命名規約については、“セットアップガイド”を参照してください。

インデックスのDSO名およびDSI名に対する命名規約

参照

インデックスのDSO名およびDSI名に対する命名規約については、“セットアップガイド”を参照してください。

付録E　セットアップAPI

ODBCデータソースの登録をコマンドから行う方法を以下に説明します。

セットアップAPIコマンド(F3CWDELA.EXE)を実行するバッチファイルを作成し、各クライアント環境に配布することによりODBCデータソース登録作業を軽減することができます。

作成したバッチファイルを“Symfoware Server クライアント機能のインストールフォルダ(ドライブ名:¥SFWCLNT)¥ODOS¥BIN”配下以外に格納した場合は、バッチファイルのPath環境変数に“Symfoware Server クライアント機能のインストールフォルダ(ドライブ名:¥SFWCLNT)¥ODOS¥BIN”を指定します。

本コマンドを利用すると、32ビットアプリケーション用のデータソースおよび64ビットアプリケーション用のデータソースが同時に登録されます。

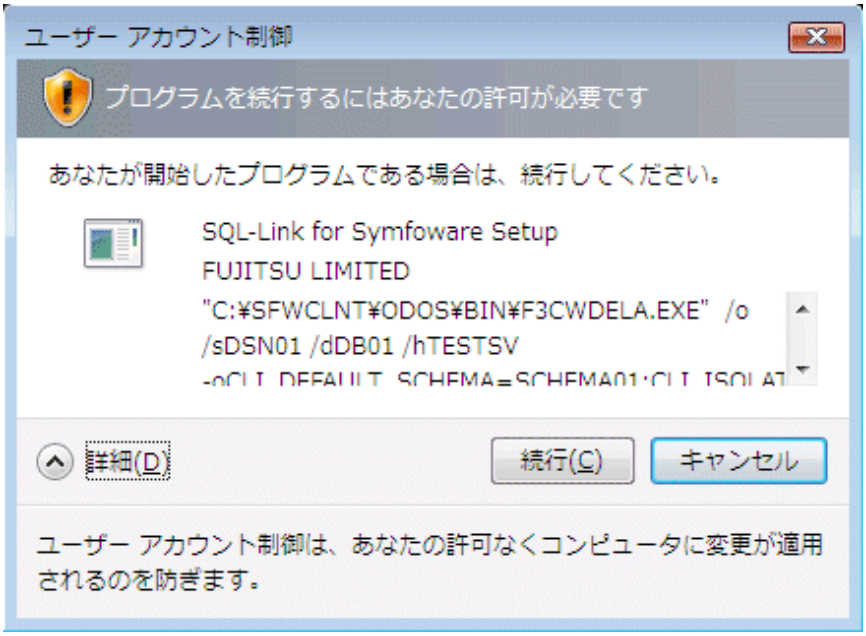
指定形式

```
F3CWDELA.EXE /o /sデータソース名 /dデータリソース名 /hホスト名 [/pポート番号]
[-c{RDB2_TCP|RDB2_TCPS}] [-l] [-u]
[-oオプション名=値[;オプション名=値…]|-oFile=ファイル名] [-y] [-n]
```

例

```
F3CWDELA.EXE /o /sDSN01 /dDB01 /hTESTSV -cRDB2_TCPS oCLI_DEFAULT_SCHEMA=SCHEMA01;
CLI_ISOLATION_WAIT=(REJECT)
```

以下の画面と同じ[ユーザー アカウント制御]ダイアログが表示された場合、[続行]ボタンをクリックして、処理を継続してください。



パラメタ・オプション

No	セットアップ情報	APIパラメタ	内容
1	ODOSセットアップ	/o	本パラメタの指定によりODOSのODBCデータソースを作成します。 本パラメタは省略できません。
2	データソース名	/s	任意のデータソース名を指定します。 本パラメタは省略できません。

No	セットアップ情報	APIパラメタ	内容
3	データリソース名 (注1)	/d	接続先データベース名を指定します。 本パラメタは省略できません。
4	ホスト名 (注2)	/h	接続先ホスト名を18バイト以内で指定します。 本パラメタは省略できません。
5	ポート番号	/p	リモートアクセスで使用するポート番号を指定します。 省略した場合は、初期値の2050となります。 Symfoware Serverのバージョンレベルや対象のプラットフォームによって、デフォルトのポート番号が異なります。サーバ側で定義されているポート番号を確認の上、指定してください。ポート番号の詳細については、“3.1 ODBCデータソースの登録”を参照してください。
6	通信方法	-c	リモートアクセスの通信方法として、以下のどちらかを指定します。 RDB2_TCP:通信データを暗号化しない場合に指定します。 RDB2_TCPS:通信データを暗号化する場合に指定します。 -lオプションとは同時に指定できません。また、-lオプションと-cオプションの両方を省略した場合は、RDB2_TCPが指定されたものとみなします。
7	ローカルデータベース	-l	ローカルデータベースアクセス指定のオプションを指定します。 省略した場合は、リモートアクセスとなります。 本オプションは、Windows Server(R)の場合のみ指定可能です。
8	UNICODE	-u	UNICODEドライバ指定のオプションを指定します。 省略した場合は、シフトJISドライバとなります。
9	ODOSオプション (注3)	-o	ODOS用チューニング情報のオプションを指定します。省略した場合は、チューニングなしとなります。
10	データソース種類	-y	システムDSNとして登録する際に指定するオプションを指定します。 省略した場合は、ユーザーDSNとして登録されます。
11	パスワード変更機能 (注4)	-n	パスワード変更機能のオプションを指定します。 省略した場合は、パスワード変更機能は利用しません。

注1) /dで指定するデータリソース名の文字列の仕様詳細は下記のとおりです。

- -lを指定しない場合、データリソース名にデータベース名が指定可能です。
  データリソース名は、全角文字および半角文字共通で、合計36バイト以内の文字列を指定してください。
  データリソース名が0バイトの場合、または37バイト以上の文字列を指定した場合、データリソース名の指定に誤りがある旨のエラーが発生します。
- -lを指定する場合、データリソース名をRDBシステム名.データベース名の形式で指定可能です。
  RDBシステム名は、8バイト以内の文字列を指定してください。9バイト以上の文字列を指定した場合、データリソース名の

指定に誤りがある旨のエラーが発生します。
なお、RDBシステム名の指定を省略した場合、RDBシステム名がない状態で動作を継続します。

RDBシステム名とデータベース名の指定組み合わせについて、以下の表で説明します。

表E.1 データリソース名の指定組合せの仕様詳細

No	オプション「/d」の文字列の指定例	指定内容	登録結果
1	/dds.db1	RDBシステム名がds、データベース名がdb1	○
2	/dds	RDBシステム名がds、データベース名を省略	×(*1)
3	/dds.	RDBシステム名がds、データベース名を省略	×(*1)
4	/d.	RDBシステム名およびデータベース名を省略	×(*1)
5	/d.db1	RDBシステム名を省略、データベース名がdb1	○
6	/d	RDBシステム名およびデータベース名を省略	×(*1)

*1)「データリソース名の指定に誤りがあります」というエラーが返却されます。

注2) -lを指定した場合、/hは省略可能となります。またこの場合は、/hおよび/pで指定した情報は無視されます。

注3) ODOSオプションに同一のオプション名が複数指定された場合には、より後ろに記述された値が有効になります。

注4) パスワード変更機能の詳細については、“付録F パスワード変更機能”を参照してください。

ODOSオプションの指定形式

ODOSオプションは“オプション名=値”の形式で列挙します。オプションの区切りは“;”です。
またファイルでの指定も可能です。その場合は“File=ファイル名”の形式で指定します。ファイル名は絶対パスで指定する必要があります。指定するファイルはTXT形式で以下のフォーマットです。

```
オプション名=値
オプション名=値
   .
   .
```

行の先頭が“;”の場合はコメントとみなします。

CLI_DEFAULT_SCHEMA

SQL文でスキーマ名を省略した場合の初期値のスキーマ名を指定します。

指定形式

```
CLI_DEFAULT_SCHEMA=(スキーマ名)
```

パラメタの意味

スキーマ名

初期値のスキーマ名を指定します。

その他のオプションについては“4.5 アプリケーションのチューニング”を参照してください。

ポイント

コマンドにオプションを列挙した場合、コマンド長の制限を超える可能性があります。また、同一のオプションを指定する場合でも、ODBCデータソースごとに指定する必要があります。このため、ファイルによる指定を推奨します。

復帰値

- 処理の復帰値は、EXEの終了コード“ERRORLEVEL”で返却します。
- 正常終了時は0、異常終了時は1以上のエラーコードを返却します。

エラーコード

返却する主なエラーコードとその内容は以下のとおりです。

エラーコード	内容
24	省略できないパラメタを省略したかまたは、不当なパラメタが指定されました。
25	データソース名の指定に誤りがあります。
26	データリソース名の指定に誤りがあります。
27	ホスト名の指定に誤りがあります。
28	ポート番号の指定に誤りがあります。
29	不当なパラメタが指定されました。
31	ODBCデータソースの登録に失敗しました。
35	ローカルデータベースオプションの指定に誤りがあります。
36	不当なパラメタが指定されました。
40	ODOSチューニング情報に誤りがあります。
41	ODOSチューニング情報の登録に失敗しました。
42	ODOSチューニング情報のファイル名に誤りがあります。
43	このプラットホームでは実行できません。
45	ODOSがインストールされていません。
46	ODOSのODBCデータソースを作成する場合には /oを指定してください。
52	パラメタの組み合わせが正しくありません。
53	Symfoware ODOSがインストールされていません。
54	Symfoware ODOS(Unicode)がインストールされていません。
55	F3CWDELP.EXEの起動に失敗しました。
254	内部エラー

ERRORLEVELの取得方法

ERRORLEVELの取得方法について説明します。

以下の例をバッチファイルとしてコピーし、実行することによりERRORLEVELが取得できます。

例

```
echo off
rem **** start /wait はWIN9x使用時にERRORLEVELを取得する際に必要です ****
rem **** ERRORLEVEL は数字の大きいものから順番に判定しなければなりません ****

start /wait f3cwdela.exe /o /sDSN01 /dDB01 /hTESTSV -oCLI_DEFAULT_SCHEMA=SCHEMA01

if ERRORLEVEL 254 goto END254
if ERRORLEVEL 55  goto END55
if ERRORLEVEL 54  goto END54
if ERRORLEVEL 53  goto END53
if ERRORLEVEL 52  goto END52
```

```
if ERRORLEVEL 46  goto END46
if ERRORLEVEL 45  goto END45
if ERRORLEVEL 43  goto END43
if ERRORLEVEL 42  goto END42
if ERRORLEVEL 41  goto END41
if ERRORLEVEL 40  goto END40
if ERRORLEVEL 36  goto END36
if ERRORLEVEL 35  goto END35
if ERRORLEVEL 31  goto END31
if ERRORLEVEL 29  goto END29
if ERRORLEVEL 28  goto END28
if ERRORLEVEL 27  goto END27
if ERRORLEVEL 26  goto END26
if ERRORLEVEL 25  goto END25
if ERRORLEVEL 24  goto END24
if ERRORLEVEL  0  goto END0
goto END_OTHER

:END0
echo 正常に終了しました
goto END

:END24
echo 省略できないパラメタを省略したかまたは、不当なパラメタが指定されました
goto END

:END25
echo ODBCのデータソース名の指定に誤りがあります
goto END

:END26
echo データリソース名の指定に誤りがあります
goto END

:END27
echo ホスト名の指定に誤りがあります
goto END

:END28
echo ポート番号の指定に誤りがあります
goto END

:END29
echo 不当なパラメタが指定されました
goto END

:END31
echo ODBCデータソースの登録に失敗しました
goto END

:END35
echo ローカルデータベースオプションの指定に誤りがあります
goto END

:END36
echo 不当なパラメタが指定されました
goto END

:END40
echo ODOSチューニング情報に誤りがあります
goto END

:END41
```

```
echo ODOSチューニング情報の登録に失敗しました
goto END

:END42
echo ODOSチューニング情報のファイル名に誤りがあります
goto END

:END43
echo このプラットホームでは実行できません
goto END

:END45
echo ODOSがインストールされていません
goto END

:END46
echo ODOSのODBCデータソースを作成する場合には /oを指定してください
goto END

:END52
echo パラメタの組み合わせが正しくありません
goto END
‘
:END53
echo Symfoware ODOSドライバがインストールされていません
goto END

:END54
echo Symfoware ODOS(Unicode)ドライバがインストールされていません
goto END

:END55
echo F3CWDELP.EXEの起動に失敗しました
goto END

:END254
echo 内部エラー
goto END

:END_OTHER
echo その他のエラー
goto END

:END
pause
```

付録F　パスワード変更機能

パスワード変更機能についての詳細を説明します。

この機能は、Symfoware/RDBのデータベースの機密保護としてSymfoware/RDBで利用者の認証情報を管理している場合に利用できます。

参照

Symfoware/RDBで利用者の認証情報を管理する方法については“RDB運用ガイド”を参照してください。

データベース簡単運用の場合に、Symfoware/RDBで利用者の認証情報を管理する方法については、“データベース簡単運用ガイド”を参照してください。

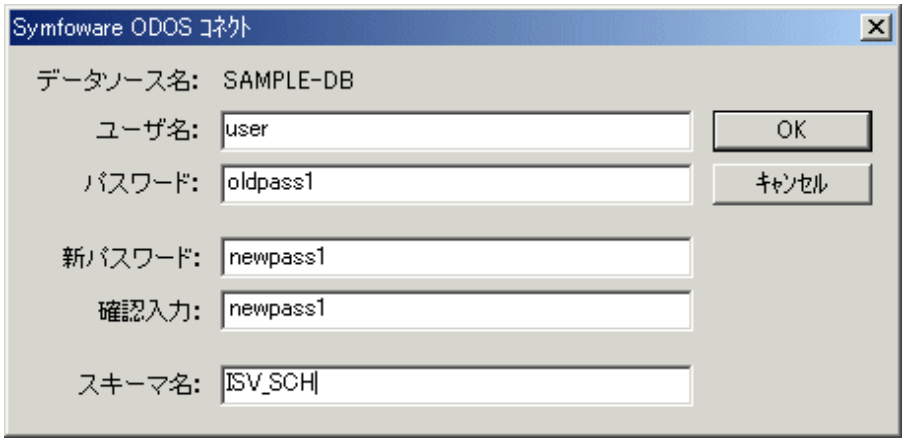
Symfoware ODOSセットアップ画面で“Symfoware利用者認証を使用する”を選択した場合、またはセットアップAPIでAPIパラメタ“-n”を指定した場合

Symfoware/RDBのデータベースへの接続時、以下の[Symfoware ODOS コネクト]画面が表示されます。

この場合、パスワードを接続時に変更することができます。

パスワードを変更して接続する場合

1. ユーザ名、パスワード、新パスワード、確認入力およびスキーマ名の各項目を入力します。

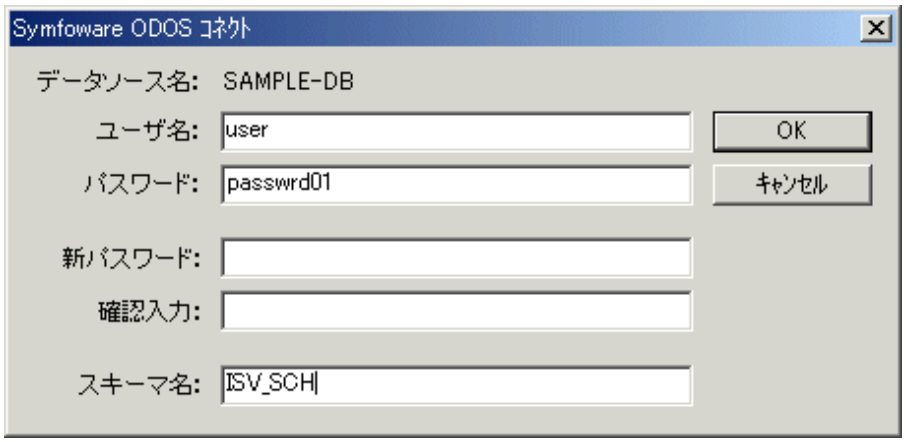


   実際には、[パスワード]、[新パスワード]および[確認入力]は*で表示されます。

2. [OK]ボタンをクリックして接続します。

パスワードを変更せずに接続する場合

1. ユーザ名、パスワードおよびスキーマ名の各項目を入力します。



   実際には、[パスワード]は*で表示されます。

2. [OK]ボタンをクリックして接続します。

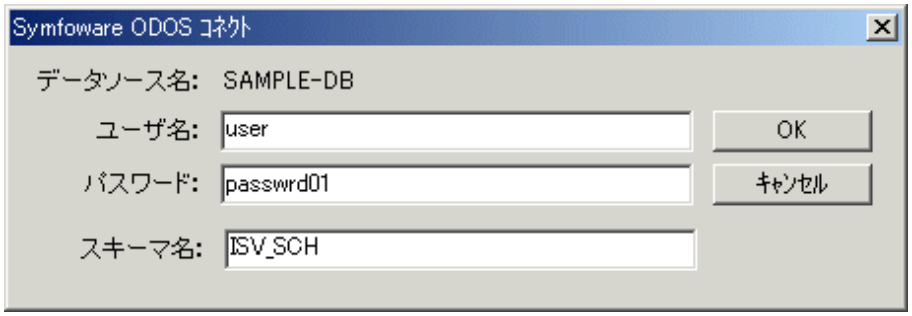
Symfoware ODOSセットアップ画面で“Symfoware利用者認証を使用する”を選択していない場合、またはセットアップAPIでAPIパラメタ“-n”を指定していない場合

Symfoware/RDBのデータベースへの接続時、以下のような画面が表示されます。

1. ユーザ名、パスワードおよびスキーマ名の各項目を入力します。

   ただし、以下の場合、ユーザ名およびパスワードの指定は不要です。

   - ローカルアクセスの場合
   - 接続先ホスト名に自端末のIPアドレス、自端末のホスト名、“localhost”またはループバックアドレスを指定したリモートアクセスの場合



   実際には、[パスワード]は*で表示されます。

2. [OK]ボタンをクリックして接続します。

付録G　使用可能SQL文一覧

ODOSを使用したアプリケーションで指定可能なSQL文について説明します。

表G.1 アプリケーションに指定可能なSQL文

分類	SQL文
システム制御文	SET SYSTEM PARAMETER文
利用者制御文	CREATE USER文 (利用者定義文)
	ALTER USER文 (利用者変更文)
	DROP USER文 (利用者削除文)
スキーマ定義文	CREATE SCHEMA文 (スキーマ定義)
	CREATE TABLE文 (注1) (表定義)
	CREATE VIEW文 (ビュー定義)
	CREATE PROCEDURE文 (プロシジャルーチン定義)
	CREATE FUNCTION文 (ファンクションルーチン定義)
	CREATE INDEX文 (注1) (インデックス定義)
	CREATE TRIGGER文 (トリガ定義)
	CREATE SEQUENCE文 (順序定義)
スキーマ操作文	DROP SCHEMA文 (スキーマ削除文)
	DROP TABLE文 (表削除文)
	ALTER TABLE文 (表定義変更文)
	DROP VIEW文 (ビュー削除文)
	DROP PROCEDURE文 (プロシジャルーチン削除文)
	DROP INDEX文 (インデックス削除文)
	DROP TRIGGER文 (トリガ削除文)
	SWAP TABLE文 (表交換文)
	DROP SEQUENCE文 (順序削除文)

分類	SQL文
	DROP FUNCTION文 (ファンクションルーチン削除文)
格納構造定義文	CREATE SCOPE文 (スコープ定義文)
格納構造操作文	DROP SCOPE文 (スコープ削除文)
	APPLY SCOPE文 (スコープ適用文)
	RELEASE SCOPE文 (スコープ解除文)
	SET STATISTICS文 (最適化情報設定文)
	PRINT STATISTICS文 (最適化情報表示文)
データ操作文	問合せ式
	INSERT文
	DELETE文: 探索
	UPDATE文: 探索
トランザクション管理文	SET TRANSACTION文
セション管理文	SET SCHEMA文
アクセス制御文	CREATE ROLE文 (ロール定義文)
	DROP ROLE文 (ロール削除文)
	GRANT文
	REVOKE文
プロシジャ実行	CALL文

注1) ODOSを利用したアプリケーションで、表定義文およびインデックス定義文を実行する場合、データベース名と同じ名前の空きのあるデータベーススペースを用意しておく必要があります。

付録H　APIリファレンス

Symfoware ServerのODBCドライバがサポートするAPIについて説明します。

H.1　サポートAPI一覧

APIのサポート状況を以下に示します。

関数名	サポート状況
SQLAllocConnect	○
SQLAllocEnv	○
SQLAllocHandle	○
SQLAllocStmt	○
SQLBindCol	○
SQLBindParameter	○
SQLBrowseConnect	×
SQLBulkOperations	×
SQLCancel	○
SQLCloseCursor	○
SQLColAttribute	○
SQLColAttributes	○
SQLColumnPrivileges	○
SQLColumns	○
SQLConnect	○
SQLCopyDesc	○
SQLDescribeCol	○
SQLDescribeParam	○
SQLDisconnect	○
SQLDriverConnect	○
SQLEndTran	○
SQLError	○
SQLExecDirect	○
SQLExecute	○
SQLExtendedFetch	○
SQLFetch	○
SQLFetchScroll	△
SQLForeignKeys	○
SQLFreeConnect	○
SQLFreeEnv	○
SQLFreeHandle	○
SQLFreeStmt	○
SQLGetConnectAttr	△

関数名	サポート状況
SQLGetConnectOption	○
SQLGetCursorName	○
SQLGetData	○
SQLGetDescField	○
SQLGetDescRec	○
SQLGetDiagField	○
SQLGetDiagRec	○
SQLGetEnvAttr	○
SQLGetFunctions	○
SQLGetInfo	○
SQLGetStmtAttr	○
SQLGetStmtOption	○
SQLGetTypeInfo	○
SQLMoreResults	○
SQLNativeSql	×
SQLNumParams	○
SQLNumResultCols	○
SQLParamData	○
SQLParamOptions	△
SQLPrepare	○
SQLPrimaryKeys	○
SQLProcedureColumns	○
SQLProcedures	○
SQLPutData	○
SQLRowCount	○
SQLSetConnectAttr	△
SQLSetConnectOption	○
SQLSetCursorName	○
SQLSetDescField	○
SQLSetDescRec	○
SQLSetEnvAttr	○
SQLSetParam	×
SQLSetPos	△
SQLSetScrollOptions	○
SQLSetStmtAttr	△
SQLSetStmtOption	△
SQLSpecialColumns	○
SQLStatistics	○
SQLTablePrivileges	○

関数名	サポート状況
SQLTables	○
SQLTransact	○

○: サポート。

△: サポート。ただし、留意事項あり。

×: 未サポート。

H.2 留意事項

サポートAPIの使用上の留意事項は以下のとおりです。

関数名	留意事項
SQLFetchScroll	第2引数で指定可能なフェッチの種類として、以下は未サポートです。 ・ SQL_FETCH_BOOKMARK
SQLGetConnectAttr	第2引数で指定可能な属性として、以下は未サポートです。 ・ SQL_ATTR_ODBC_CURSORS ・ SQL_ATTR_TRACE ・ SQL_ATTR_TRACEFILE
SQLParamOptions	第2引数で以下の属性が指定され、第3引数に2以上の値が指定された場合、1が指定されたものとして動作します。その際、アプリケーションに、JYP9506Wの警告を返却します。 ・ SQL_ATTR_PARAMSET_SIZE 第2引数で以下の属性が指定され、第3引数にNULLではない値が指定された場合、NULLが指定されたものとして動作します。その際、アプリケーションに、JYP9506Wの警告を返却します。 ・ SQL_ATTR_PARAMS_PROCESSED_PTR 第2引数で指定可能なパラメタの入力配列数に1より大きい値を指定した場合、32キロバイト以上のBLOB型は未サポートです。
SQLSetConnectAttr	第2引数で指定可能な属性として、以下は未サポートです。 ・ SQL_ATTR_ODBC_CURSOR ・ SQL_ATTR_TRACE
SQLSetPos	第2引数で指定可能な値は0または1です。
SQLSetStmtAttr	第2引数で指定可能な属性として、以下は未サポートです。 ・ SQL_ATTR_KEYSET_SIZE ・ SQL_ATTR_FETCH_BOOKMARK_PTR ・ SQL_ATTR_PARAM_BIND_OFFSET_PTR ・ SQL_ATTR_PARAM_OPERATION_PTR ・ SQL_ATTR_ROW_BIND_OFFSET_PTR ・ SQL_ATTR_ROW_OPERATION_PTR 第2引数で以下の属性を指定する場合、32Kバイト以上のBLOB型は未サポートです。

関数名	留意事項
	・ SQL_ATTR_PARAMSET_SIZE ・ SQL_ATTR_ROW_ARRAY_SIZE 第2引数で以下の属性が指定され、第3引数に2以上の値が指定された場合、1が指定されたものとして動作します。その際、アプリケーションに、JYP9618Wの警告を返却します。 ・ SQL_ATTR_ROW_ARRAY_SIZE ・ SQL_ATTR_PARAMSET_SIZE 第2引数で以下の属性が指定され、第3引数に1以上の値が指定された場合、0が指定されたものとして動作します。その際、アプリケーションに、JYP9618Wの警告を返却します。 ・ SQL_ATTR_ROW_BIND_TYPE ・ SQL_ATTR_PARAM_BIND_TYPE 第2引数で以下の属性が指定され、第3引数にNULLではない値が指定された場合、NULLが指定されたものとして動作します。その際、アプリケーションに、JYP9618Wの警告を返却します。 ・ SQL_ATTR_ROW_STATUS_PTR ・ SQL_ATTR_PARAM_STATUS_PTR ・ SQL_ATTR_ROWS_FETCHED_PTR ・ SQL_ATTR_PARAMS_PROCESSED_PTR
SQLSetStmtOption	第2引数で指定可能な属性として、以下は未サポートです。 ・ SQL_KEYSET_SIZE 第2引数で以下の属性を指定する場合、32Kバイト以上のBLOB型は未サポートです。 ・ SQL_ROWSET_SIZE 第2引数で以下の属性が指定され、第3引数に2以上の値が指定された場合、1が指定されたものとして動作します。その際、アプリケーションに、JYP9506Wの警告を返却します。 ・ SQL_ROWSET_SIZE

付録I　DPCライブラリからの移行について

DPCライブラリからODBCドライバに移行する際には、DPCライブラリを使用して構築した業務が問題なく動作することを検証後、移行を行ってください。DPCライブラリの環境定義時に、クライアント用の動作環境ファイルを指定していた場合、その内容をODOSのセットアップ時に反映してください。

索　引

[な]

[は]

[ま]

[ら]