以下は、wxWidgets ODBC クラス - wxDb, wxDBTable, 及び関連する関数の使用方法の詳細な概要である。これらは Remstar International により提供された ODBC クラスであり、ここではまとめて wxODBC と呼ばれる。

・wxDb/wxDbTable wxODBC 概要
・wxODBC のはじめ方
・wxODBC - ODBC を使用するための設定
・wxODBC - コンパイル
・wxODBC - 基本ガイド
・wxODBC - 既知の不具合
・wxODBC - サンプルコード
・wxODBC - SQL コマンドの選択

# wxDb/wxDbTable wxODBC 概要

wxODBC クラスは、データベース非依存になるように設計された。SQL も ODBC もどちらも、仕様に従ってサポートすべき最小限の要求を定義した標準規格を持っているが、別のデータベースベンダはこれらを少し異なって実装しているかもしれない。この 1 つの例は、Oracle はデータソースのユーザ名を大文字で提供している。このような状況において、データベース固有のシンタックスを要求する関数を使用する際、wxODBC クラスはこれをプログラマに対して透過的になるよう書かれている。

今現在いくつかの有名なデータベースは、他の広く使用されるデータベースと共に wxODBC クラスを介してテストされ、サポートされている。多くのユーザがこれらのクラスを備えたソフトウェアを実装し始めているため、サポートされたデータベースのリストは、確実に増えてきている。しかし、このドキュメントを記述している時点では、ユーザは首尾よく以下のデータソースを備えたクラスを使用している。:

・DB2
・DBase (IV, V)**
・Firebird
・INFORMIX
・Interbase
・MS SQL Server (v7 - minimal testing)
・MS Access (97, 2000, 2002, and 2003)
・MySQL (2.x and 3.5 - use the 2.5x drivers though)
・Oracle (v7, v8, v8i)
・Pervasive SQL
・PostgreSQL
・Sybase (ASA and ASE)
・XBase Sequiter
・VIRTUOSO

最新のリストは、db.cpp の関数 wxDb::Dbms のコメントか、db.h の wxDBMS の定義 (enum 型 ) を参照して得ることができる。

dBase は厳密には ODBC データソースではない。しかし、dBase テーブルへの ODBC 接続の多くの

機能性がエミュレートされるドライバが存在する。「wxODBC - 既知の不具合」の項を参照のこと。

## wxODBC のはじめ方

はじめに、もしも SQL や ODBC について詳しくないのであれば、近くの書店でそれぞれの良い書籍を買うことを薦める。この主題に熱中することにより幾つか習得することがあるかもしれないが、このドキュメントは、SQL や ODBC について多くの詳細を教えるつもりではない。

もし、SQL/ODBC ではないデータソースを使用して仕事をしたことがあるなら、習っていない状態にする必要がある事柄があるかも知れない。まず、以下のような専門用語が頻繁に使用される。

| 名称 | 説明 |
|---|---|
| データソース | ( 大抵はデータベース ) wxODBC クラスによってアクセスされるデータを含む。 |
| データテーブル | データソースのセクションはデータの列とカラムのデータを含む。 |
| ODBC ドライバ | アプリケーションから送られた ODBC コマンドを解釈し、それらを対象となるデータソースによって期待される SQL 構文に変換するミドルウェア。 |
| データソースコネクション (Datasource connection) | アプリケーションと、対象となるデータソースに対してコネクションを行っている ODBC ドライバの間の ( 開いた ) パイプ。データソースコネクションは、同じコネクション（ODBC ドライバによって決まる）を使用して仮想的に無限個の wxDbTable インスタンスを持つことが可能である。個別のコネクションはテーブル毎に必要ない（the exception is for isolating commits/rollbacks on different tables from affecting more than the desired table. wxDb::CommitTrans と wxDb::RollbackTrans のドキュメントを参照のこと）。 |
| 行 (Rows) | 旧関係データベースで言うレコードと同様、列は、互いに関係しているデータテーブルのカラム毎の１つのインスタンスの集合である。 |
| カラム ( 列、Columns) | データテーブルの行毎に結び付けられた特定のフィールドである。 |

| | |
|---|---|
| クエリ (Query) | クライアントからデータソースに、ユーザが要求した特定の要求に合致するデータを問い合わせるための要求。クエリが実行されると、データソースはクエリを満たす行を検索し、結果集合を作成する。 |
| 結果集合 (Result set) | データソースに送られたクエリに定められた要求に合致したデータ。ドライバに依存し、クライアントが ODBC ドライバに対して結果集合を取り出すように実際に指示するまで、結果集合は通常データソースに残っている（ODBC ドライバにはデータは送信されない）。 |
| カーソル (Cursor) | クエリが生成した結果集合への論理的なポインタであり、次のレコードへの要求が作られると、次のレコードを指し示す。 |
| カーソルのスクロール (Scrolling cursors) | スクロールは結果集合の最初から最後までカーソルの移動を参照する。カーソルは常に結果集合（前方のみのスクロールカーソル）を連続にスクロールすることができる。前方スクロールカーソルを使用した場合、結果集合の中の行が一旦クライアントの ODBC ドライバに戻ると、カーソルは結果集合の中を戻ることができなくなる。ODBC ドライバとデータソースの両方で後方スクロールカーソルがサポートされている場合、後方スクロールカーソル関数が使用できる（wxDbTable::GetPrev、wxDbTable::GetFirst、及び wxDbTable::GetLast）。データソースまたは ODBC ドライバが前方スクロールカーソルのみをサポートしている場合、アプリケーションはこれを取り込まなくてはならない。 |
| コミット／ロールバック (Commit/Rollback) | コミットは、挿入／削除／更新を物理的に保存し、一方、ロールバックは基本的に、既にコミット済みのデータソースコネクションをアンドゥー (undo) する。コミットとロールバックはコネクション上で行われるものであり、特定のテーブルに対して行われるものではない。データソースに対して共通のコネクションを使用する全てのテーブルは wxDb::CommitTrans または wxDb::RollbackTrans が呼ばれたときに同時にコミット／ロールバックされる。 |

| | |
|---|---|
| インデックス (Index) | インデックスはデータソースにより管理される検索の仕組みであり、データソースが特定のカラムの値に基づいたデータ行を素早く検索できるようにする。インデックスが無い場合、データソースはクエリ要求がある度に順次検索を行わなければならない。真にユニークキーインデックスな構成はデータソースのクエリを殆ど瞬間に行うことができる。 |

データソースの中のデータテーブルからデータを読む前に、データソースへのコネクションを作らなければならない。各々のデータソースコネクションは同じコネクション上で複数のテーブルを開くために使用されるかも知れない（開くことができるテーブルの数は、ドライバ、データベース構成、クライアントワークステーションのメモリ量に依存する）。複数のコネクションにより同じクライアントで同じデータソースを開くことができる（同時に開くことができるコネクション (concurrent connections) の数はドライバとデータソース構成に依存する）。

クエリが実行されると、クライアントはクエリを ODBC ドライバに渡し、その時ドライバはクエリをデータソースに渡す。データベースを提供するマシン上で実行しているデータソースエンジン（in most cases - exceptions are text and dBase files）は、要求されたデータを検索するための全ての処理を実行する。クライアントはデータソースから ODBC ドライバを介して状態が戻るのを単に待つだけである。

ODBC ドライバに依るが、結果集合がデータベースサーバ上に " 待ち " が残るか、ドライバが待たされているマシンに転送される。クライアントはこのデータを受信しない。クライアントは、データ行がクライアントアプリケーションに返される前に、幾つか、または全ての結果集合が返されるを要求しなければならない。(The client must request some or all of the result set to be returned before any data rows are returned to the client application.)

結果集合にはクエリに合致した各行の全てのカラムを含んでいる必要は無い。事実、結果集合は２個以上のデータテーブルから得られたカラムの連結になりえる。(In fact, result sets can actually be joinings of columns from two or more data tables, may have derived column values, or calculated values returned.)

各々の結果集合に対し、カーソルは、ユーザが現在アクセスしている結果集合の位置がどこにあるのかを追跡するよう（通常、データベースにより）保持されている。データベースと ODBC ドライバ、setup.h の設定によりどのように wxWidgets ODBC を設定するかに依るが（「wxODBC - コンパイル」を参照）、カーソルは前方または後方スクロールのどちらかになりうる。最低でもカーソルは前方にスクロールする。例えば、クエリが１００行の結果集合を返せば、クライアントアプリケーションによりデータがリードされる時、行１，２，３のように読まれる。前方のみのカーソルの場合には、一旦カーソルが次の行に移動すると、再度クエリを実行しない限り、前の行に再びアクセスすることができない。後方スクロールカーソルは結果集合から前の行を要求することが可能であり、実際にはカーソルを後方にスクロールさせる。

後方スクロールカーソルは、全てのデータベース／ドライバの組み合わせでサポートされている

訳ではない。前方のみのカーソルは wxODBC クラスでデフォルトである。もしデータソースが後方スクロールカーソルをサポートしており、それを使用したい場合、setup.h に適切な修正をしなければならない(「wxODBC - コンパイル」を参照)。データソース間で優れた移植性を持たせるためには、前方スクロールカーソルのみを使用してプログラムを書くことが最善の方法である。一方、後方スクロールカーソルをサポートしたデータソースのみを対象としているのであれば、潜在的に良いパフォーマンスを得ることができる。

データソースへの各コネクションを開くことができるカーソルの数、通常、データソース自身のためのカーソルの最大値には限界がある。これは全てデータベースに依存する。開かれた(wxDb の各インスタンス) 各コネクションは、更新／削除／ロールバック／クエリなどのために、作成時に最低５つのカーソルを開く。カーソルは有限なリソースであるため、ユーザはカーソルを多く作成するときに注意しなければならない。

追加のカーソルは必要に応じて wxDbTable::GetNewCursor を使用して作成することができる。追加のカーソルを使用する１つの例は、結果集合を複数のスクロールポインタで巡回する場合である。新しいカーソルを作成することにより、１つ目の結果集合で元のカーソルの場所を保持する間、プログラムはデータソースから２つ目の結果集合を作成することもありうる。

SQL/ODBC ではないデータソースと違い、プログラムが ODBC を介して挿入、削除、更新を実行するとき(あるいは、修正テーブル (altering table) などの他の SQL 機能)、プログラムはデータソースに永続的に記録されるように問い合わせるため、データソースに " コミット " を発行する必要がある。コミットが実行されるまで、データソースにクエリを発行した他のどのプログラムも、その変更を知ることができない (しかし、自動コミットされるようなデータベースが存在する)。大部分のデータソースでは、コミットが実行されるまで、同じデータソースコネクション上に開かれたカーソルはアンコミットされた変更を見ることが可能である。これを検証するために、使用するデータベースのドキュメント／設定を参照のこと。

ロールバックは、基本的にはデータソースコネクション上において、アンドゥー (UNDO) コマンドである。ロールバックが発行されると、データソースは最後にコミットが実行された時点まで、全てのコマンドを消去する。

" コミット／ロールバックはデータソースコネクション上 (wxDb インスタンス) で行われるのであって、wxDbTable インスタンスに対して行われるものである。これが意味するところは、１つ以上のテーブルが同じコネクションを共有しており、コミットまたはロールバックがそのコネクションに対して実行された場合、そのコネクションを使用している全てのテーブルに対する全ての未解決なままの変更は、コミット／ロールバックされる。"

## wxODBC - ODBC を使用するための設定

データソースにアクセスできるようになる前に、ODBC ドライバのインストールと設定が必要である。これはシステム固有であるため、ここでは詳細に説明しない。しかし、ここでは、幾つかの事柄を説明する。

多くのデータベースベンダは、ベンダのデータベース製品とともに、最低限の ODBC ドライバを

提供している。実際のところ、これらのドライバの多くは低速で不完全なものである。ベンダは彼らの製品に ODBC インターフェースを使用したがらないとの噂がある。ベンダはデータにアクセスするために彼らのアプリケーションを使用してもらいたいのである。

理由が何であれ、データベースアプリケーションのために、サードパーティの ODBC ドライバを使用しなければならない。良くテストされ、良く利用されるサードパーティ製の ODBC ドライバの一例は、Rogue Wave のドライバである。Rogue Wave は多くの異なるプラットフォームやデータベースに利用可能なドライバを持っている。Microsoft Windows 環境では、使用しようと計画している ODBC ドライバをインストールすること。その時、所望するデータソースに対するドライバのインスタンスを設定するために、コントロールパネルで ODBC アドミニストレータを使用する。NT 系で使用する場合には、この設定はシステムまた DSN はユーザ DSN（データソース名）として設定しなければならない。システムリソースとして設定するということは、全てのユーザに対して利用可能にすることを意味し（アドミニストレータとしてログインした場合）、そうでなければ、そのデータソースは DSN を設定したユーザでのみ利用可能になる。

Unix 環境では、iODBC は ODBC API の実装に利用される。wxODBC クラスをコンパイルするために、まず http://www.iodbc.org から iODBC を入手し、インストールしなければならない（Note: wxWidgets currently includes a version of iODBC.）。そして、"~/.odbc.ini" ファイル（あるいは状況に応じて、システム上の前ユーザがアクセスできるよう、"/etc/odbc.ini"）を作成しなければならない。このファイルはシステムとデータソースのための設定を含んでいる。以下は MySQL を使用して "samples/db" を使用するための、odbc.ini ファイルの一部分の例である。

```
[contacts]
Trace    = Off
TraceFile= stderr
Driver   = /usr/local/lib/libmyodbc.so
DSN      = contacts
SERVER   = 192.168.1.13
USER     = qet
PASSWORD =
PORT     = 3306
```

## wxODBC - コンパイル

wxWidgets の setup.h ファイルは、wxODBC クラスのコンパイルに関連した設定がある。

| マクロ名 | 説明 |
|---|---|
| wxUSE_ODBC | wxODBC クラスをコンパイルするためには１でなければならない。１でなければ、wxODBC クラスを使用することができない。デフォルトは０である。 |

| | |
|---|---|
| wxODBC_FWD_ONLY_CURSORS | 新しいデータベースコネクションが要求された場合、この設定は、前方スクロールカーソルのみの使用を許可するか、前方と後方スクロールカーソルの使用を許可するかどうかのデフォルト設定を制御する（カーソルに関する更なる情報は「wxODBC のはじめ方」を参照のこと）。このデフォルトは wxDbGetConnection または wxDb コンストラクタの第２引数で変更することが可能である。デフォルトは１である。 |
| wxODBC_BACKWARD_COMPATABILITY | Between v2.0 and 2.2、 massive renaming efforts were done to the ODBC classes to get naming conventions similar to those used throughout wxWidgets、 as well as to preface all wxODBC classes names and functions with a wxDb preface. Because this renaming would affect applications written using the v2.0 names、 this compile-time directive was added to allow those programs written for v2.0 to still compile using the old naming conventions.（<-- 要は、v2.0 から 2.2 まで準備されていた関数は性能が悪く、それ以降のバージョンで新しい関数名として改善されたが、互換性のために、旧バージョンの関数が残されている　... ということかな？）これらの非推奨な名称は、db.cpp/dbtable.cpp の最後で、新しい関数名に対応するように全て #define で定義された。これらの非推奨なクラス名／関数名は、将来削除される可能性があるため、今後の開発で使用すべきではない。デフォルトは０である。 |

MS Windows 環境

コンパイラベンダから提供されている "odbc32.lib" をリンクしなければならない。wxWidgets で提供されている名句ファイルを使用する場合には、makefile.b32、makefile.vc、及び makefile.g95 の中で、このライブラリを使用するように既にインクルードされている。

MORE TO COME

Unix 環境 --with-odbc flag for configure

MORE TO COME

## wxODBC - **基本ガイド**

wxODBC クラスをアプリケーションで使用するために、８つの基本的な手順がある。

・データソースコネクション情報の定義
・データソースコネクションの取得
・テーブル定義の作成
・テーブルの開く
・テーブルを使用する
・テーブルを閉じる
・データソースコネクションを閉じる
・ODBC 環境ハンドルを開放する

以下に各手順について詳細に説明し、ビギナーがクラスを使用し始めるときに陥る、多くの落とし穴について説明する。各手順で、コードの断片はその手順を実行するためのシンタックスを示すために準備されている。完全なコードはこの概要の最後に準備されており、これら全ての手順の完全なワークフローを示している（「wxODBC - サンプルコード」を参照）。

## データソースコネクション情報の定義

ODBC ドライバを介してデータソースに接続できるためには、プログラムは最低限３つの情報（データソース名、ユーザ ID、そして認証文字列（パスワード））を提供しなければならない。４つ目の情報として、データファイルが保存される場所を示すデフォルトディレクトリが Text ドライバ、dBase ドライバで要求される。

wxWidgets データクラス wxDbConnectInf はこれらの値と、要求によってはさらに幾つかの情報を保持するために存在する。

'Henv' メンバは ODBC ドライバが使用するメモリにアクセスするために使用される環境ハンドルである。このメンバの使用方法は以下にある「データソースコネクションの取得」で記述されている。

ODBC データソース（ODBC アドミニストレータ（MSW のみ）または .odbc.ini ファイルに存在する）を構築するために、'Dsn' はデータソース名に厳密に一致していなければならない。

'Uid' はデータソースにログインする際に使用されるユーザ ID である。このユーザ ID は既に作成されており、接続しようとしているデータソースへの権利が与えられていなければならない。コネクション情報が確立されたことのあるコネクションを使用するときには、接続が確立されたユーザは、何が正しいか、また、プログラムが保持することを許されたデータソースコネクションに何が特権を与えるか、決定するであろう。（The user that the connection is establish by will determine what rights and privileges the datasource connection will allow the program to have when using the connection that this connection information was used to establish.）データソースによっては、ユーザ ID の大文字･小文字を区別する。しかし、wxODBC クラスは、データソースの要求に受け入れられたデータを管理することにより、このこと（大文字・小文字を区別すること）をユーザから隠蔽しようとする。データソースが要求するときに 'Uid' を使用することが常に良い。（Some datasources are case sensitive for User IDs, and though the wxODBC classes attempt to hide this from you by manipulating whatever data you pass in to match the datasource's needs, it is always best to pass the 'Uid' in the case that the datasource requires.）

'AuthStr' は、'Uid' メンバで指定されたユーザ ID に対するパスワードである。'Uid' と同様、データ

ソースによっては大文字・小文字が区別される（事実、多くがそうである）。wxODBC クラスは、'AuthStr' の大文字・小文字を管理することは一切無い。パスワードは、そのままの文字でデータソースに渡されるため、ユーザはデータソースが期待した文字を使用しなければならない。

'defaultDir' メンバは、ファイルベースのデータソース（即ち、dBase、FoxPro、テキストファイル）で使用される。それは、データテーブル、またはデータファイルが存在する古パスを含んでいる。この値を設定する際、バックスラッシュよりもスラッシュ '/' を使用することにより、ODBC ドライバ間の互換性の差異を避けることができる。

他のフィールドは今のところ使用されない。これらフィールドは、データソースに関わらず、MSW と Unix の両方で動作する我々の ODBC アドミニストレータプログラムを作成するときに使用されることを意図している。この作業は今まで殆ど行われていない。

## データソースコネクションの取得

データソースへのコネクションを確立する方法は２種類ある。ユーザ自身で wxDb インスタンスを作成しコネクションを開く方法か、またはコネクションの作成／メンテナンス／削除を行うために wxODBC クラスで準備されたキャッシング機能を使用する方法である。

どちらの方法を使用するにしても、まず wxDbConnectInf オブジェクトを使用する必要がある。wxDbConnectInf インスタンスには、有効な Dns、Uid、及び AuthStr（必要であれば、'defaultDir' も）が準備されている。しかし、これを使用する前に、環境ハンドルを 'Henv' メンバに割り当てなくてはならない。

```
wxDbConnectInf DbConnectInf;
DbConnectInf.SetDsn("MyDSN");
DbConnectInf.SetUserID("MyUserName");
DbConnectInf.SetPassword("MyPassword");
DbConnectInf.SetDefaultDir("");
```

ODBC コネクション用に環境ハンドルを割り当てるために、wxDbConnectInf クラスには必要なハンドルを作成するためのデータソースに依存しない方法がある。:

```
if (DbConnectInf.AllocHenv())
{
    wxMessageBox("Unable to allocate an ODBC environment handle",
                 "DB CONNECTION ERROR", wxOK | wxICON_EXCLAMATION);
    return;
}
```

wxDbConnectInf::AllocHenv() が成功すると、true が返る。false は割り当てに失敗したことを意味し、ハンドルは未定義になる。

上記の方法をより簡潔に行うための書式が、wxDbConnectInf コンストラクタでカプセル化されている。

```
wxDbConnectInf *DbConnectInf;

DbConnectInf = new wxDbConnectInf(NULL, "MyDSN", "MyUserName",
                                  "MyPassword", "");
```

コンストラクタでの初期化の際、SQL 環境ハンドルに NULL を渡すと、コンストラクタでの初期化の際にハンドルを割り当てる事ができる。このハンドルは wxDbConnectInf の生存期間の間 管理され、インスタンスが破棄されるときに自動的に破棄される。

一旦 wxDbConnectInf インスタンスが初期化されれば、データソースに接続する準備はできている。

データソースコネクションを手動で作成するには、wxDb インスタンスを作成し、オープンしなければならない。

```
wxDb *db = new wxDb(DbConnectInf->GetHenv());
opened = db->Open(DbConnectInf);
```

１行目は、wxDb クラスの全てのメンバを初期化するために状態監視をする。２行目は、wxDb::Open を呼ぶときに渡される引数を使用するデータソースに関連付けられたコネクションを開くために ODBC ドライバに要求を送る。

コネクションを開くためのより進んだ方法は、wxODBC クラスに含まれるコネクションキャッシング機能を使用することである。キャッシングの仕組みは、手動でコネクションを開くのと同じ関数が使用される。しかし、それらは作成された各々のコネクションで管理され、コーディングしていなくても、コネクションを閉じるときには破棄と再利用される。

データソースへのコネクションを取得のためにキャッシング関数 wxDbGetConnection を使用するために、wxDbConnectInf 型の引数を１つ渡す方法がある。:

```
db = wxDbGetConnection(DbConnectInf);
```

戻り値の wxDb へのポインタは、初期化され、オープンされている。コネクションを開くときに何らかの問題があれば、wxDbGetConnection の戻り値は NULL になる。

得られたコネクションは、新しいコネクションか、もう使われていないクラスが管理するコネクションのキャッシュから得られた " フリーな " コネクションである。wxDbGetConnection をコールして作られる wxDb インスタンスは、確立したコネクションのリンクリストに記録される。プログラムがコネクションを終了するとき、wxDbFreeConnection の呼び出しが行われ、そのとき、データソースコネクションが " フリーになった " と認識され、同じ情報（Dsn、Uid、AuthStr）を使用するコネクションを要求する次の wxDbGetConnection 呼び出しが利用可能になる。キャッシュされた関数は、wxDbCloseConnections が呼ばれるまで、キャッシュされた状態になる。そのとき、キャッシュされた関数はクローズされ、削除される。（The cached connections remain cached until a call to wxDbCloseConnections is made, at which time all cached connections are closed and deleted.）

データソースコネクションを取得するための１つのキャッシングルーチンを使用することの明白な長所に加え、キャッシュされた関数を使用することはパフォーマンスを素晴らしく向上することもできる。新しくコネクションを作成するたびに（フリーなコネクションのキャッシュから取得するのではなく）、wxODBC クラスは、データソースのデータ型及び他の基本的な振る舞いを決定するためにデータソースに対して多くのクエリを実行する。ハードウェア、ネットワークの

バンド幅、データソースの速度によっては、新しいコネクションを確立するのに数秒で終えられるケースもある（良くバランスの取れたシステムでは、それはほんの一瞬であるべきである）。データソースコネクションを作成／削除するよりは、既に確立されたデータソースコネクションを再利用するほうが、コネクションの作成／削除に関して時間を節約することができる。

他の時間節約の方法は、wxDb::Open と wxDbGetConnection の両方を使用した " コネクションのコピー " である。もしも、主導で wxDb インスタンスを作成しオープンした場合、既存のコネクション設定をコピーすることによって性能面での利便性を得るために、wxDb::Open 関数に既存のコネクションを渡す必要がある。wxDbGetConnection 関数は、Dsn、Uid、AuthStr が同じ設定であるような既存のコネクションにコネクションを要求するとき、Dsn、Uid、AuthStr パラメータを調べることで、これを自動的に行う。もし１つでも見つかれば、wxDbGetConnection は、全ての同じ設定でデータソースに再度クエリするよりも、既にクエリされたデータ型と他のデータソース仕様情報のためにデータソース設定をコピーする。

コネクション作成時の最後の注意事項である。コネクションが作成されるときには、前方スクロールのみか、または前方と後方スクロールの両方がデフォルトで設定される。デフォルトの動作は、wxWidgets をコンパイルするときに、setup.h の wxODBC_FWD_ONLY_CURSORS の設定により決定される。ライブラリは前方スクロールカーソルのみをサポートするようにデフォルトで設定されるが、wxDb コンストラクタまたは wxDbGetConnection 関数の引数で切り替えることが可能である。全てのデータソースと ODBC ドライバは、前方スクロールカーソルをサポートしていなければならない。多くのデータソース及び多くの ODBC ドライバは、後方スクロールカーソルをサポートしている。後方スクロールカーソルを使用する計画を立てる前に、使用するデータソースと ODBC ドライバが後方スクロールカーソルをサポートしているか確認すべきである。この概要の冒頭で記述された「スクロールカーソル」、または、wxDb クラスのドキュメントで記載されたカーソル動作の詳細を参照のこと。

## テーブル定義の作成

wxDb クラスの様々な関数（wxDb::GetData を参照のこと）を使用して、データソーステーブルに直接アクセスすることが可能である。しかし、シンプルにするために、wxDbTable クラスは必要とされる全ての SQL 仕様の API コールを隠蔽している。

wxDbTable クラスを介してデータソーステーブルにアクセスするための始めの手順は、wxDbTable インスタンスを作成することである。

```
table = new wxDbTable(db, tableName, numTableColumns, "",
                      !wxDB_QUERY_ONLY, "");
```

インスタンスを作成する際、以下の情報が必要になる。テーブルにアクセスできるように確立されたデータソースコネクション、データソーステーブルを使用してアクセスされる始めのテーブル名、返される行数、実際にクエリされるテーブルのビュー名（Oracle 使用時のみ）、返ったデータがクエリされるだけかどうか、データソースに接続するときに指定されたテーブルへのパスと異なる場合には、そのパス。

上記引数の詳細な説明は wxDbTable クラスで記述されているが、ここでは５個目の引数（クエリ

のみの設定）について説明しておく。wxDbTable 作成時に wxDB_QUERY_ONLY を指定された場合、wxDbTable インスタンスを使用した挿入／削除／更新は許可されない。この wxDbTable インスタンスを使用して wxDb::CommitTrans または wxDb::RollbackTrans をコールしても、このインスタンスに対しては無視される。上に示されたように !wxDB_QUERY_ONLY を指定して wxDbTable インスタンスが作成された場合、テーブルへの挿入／削除／更新のためのカーソルやオーバヘッド（overhead）が作成され、それによって、これらの操作はこの wxDbTable インスタンスを介して行うことができる。

テーブルが wxDbTable インスタンスを介して関連付けられている場合、テーブルは読み出しを許可されるだけであり、書き出しは許可されない。しかし、パフォーマンスの向上し（メンテナンス／更新用のカーソルのために必要となる多くのカーソルが不要になり、アクセス時間が向上する）、wxDbTable インスタンスを作成するのに少しのカーソルだけを準備すればよくなるため、リソースを減らすことができる。また、複数のデータソースを使用するとき、同時に使用されるカーソルの数が限定される。

wxDbTable インスタンスによって取り出すことが可能なカラムを定義するとき、テーブルの中で１カラム目から全てのカラムに対して、どこにでも指示することができる。

```
table->SetColDefs(0, "FIRST_NAME", DB_DATA_TYPE_VARCHAR, FirstName,
                  SQL_C_WXCHAR, sizeof(FirstName), true, true);
table->SetColDefs(1, "LAST_NAME", DB_DATA_TYPE_VARCHAR, LastName,
                  SQL_C_WXCHAR, sizeof(LastName), true, true);
```

カラムの定義は、インデックス０から開始し、１ずつ増加し、wxDbTable インスタンスを作成するときに指定されたカラムの数未満で無ければならない（この例では、２カラムであり、１つはインデックス０、１つはインデックス１である。）。

上記コードは、クライアントアプリケーションでメモリ変数に割り付けられたデータソースカラムに " バインド " する（結び付ける）。したがって、アプリケーションから wxDbTable::GetNext（または、結果集合からデータを取り出す他の関数）の呼び出しがあると、カラムにバインドされた変数は、変数にストアされたカラム値を持つ。この関数の全ての引数に関する詳細は、wxDbTable::SetColDefs クラスのドキュメントを参照のこと。

バインドされたメモリ変数は、作成された結果集合からデータを取り出す関数（例えば、wxDbTable::GetNext、wxDbTable::GetPrev など）が呼び出されるまで、未定義値を持つ。その変数は、wxODBC クラスによっては初期化されず、wxDbTable::Query の呼出し後も未定義値を含んでいる。::GetXxxx() という関数のうち、何らかの関数呼び出しが成功したときだけ、変数が有効な値を持つ。

クライアントに返されることがないデータに対するカラム定義は必要ない。例えば、データソースに対してファーストネームが 'GEORGE' である全てのユーザをクエリし、これらの行に関連付けられたラストネームのリストだけが必要であった場合（ファーストネームが 'GEORGE' であることを既に知っている場合には、FIRST_NAME カラムは常に 'GEORGE' を返す）、上記１つのカラムだけを定義すればよい。

要求した同じ wxDb インスタンスを使用する同じテーブルにアクセスするために、多くの

wxDbTable インスタンスを持つかもしれない。クラスを使用する上で制限は無い。さらに、サポートされている全てのデータソースは、これに対する制限は無い。

## テーブルのオープン

テーブルを開くことは、データソース自身に対して技術的には何もしなくて良い。
Calling wxDbTable::Open simply does all the housekeeping of checking that the specified table exists, that the current connected user has at least SELECT privileges for accessing the table, setting up the requisite cursors, binding columns and cursors, and constructing the default INSERT statement that is used when a new row is inserted into the table (non-wxDB_QUERY_ONLY tables only).

```
if (!table->Open())
{
    // An error occurred opening (setting up) the table
}
```

The only reason that a call to wxDbTable::Open is likely to fail is if the user has insufficient privileges to even SELECT the table. Other problems could occur, such as being unable to bind columns, but these other reason point to some lack of resource (like memory). Any errors generated internally in the wxDbTable::Open function are logged to the error log if SQL logging is turned on for the classes.

## テーブルを使用する

セットアップされたテーブルと定義を使用するために、まず、データソースに対して結果集合に集めたいデータが何であるか、どこからデータを得るか、どのような手順でデータを返してもらいたいかを定義する必要がある。

```
// WHERE 節は、テーブル内のどの行を結果集合に返してほしいかを限定・指定する。
table->SetWhereClause("FIRST_NAME = 'GEORGE'");

// 結果集合は 'LAST_NAME' カラムのデータをアルファベット順にソートされている。
// ２つの行に同じラストネームがある場合、'AGE' カラムで従属ソート（sub-sort）される。
table->SetOrderByClause("LAST_NAME, AGE");

// 他のテーブル（連結）がこのクエリに対して使用されない。
table->SetFromClause("");
```

上記は､テーブル内のカラム "FIRST_NAME" に名前 'GEORGE' を含む行の結果集合を返すためのデータソースへの問い合わせである（シングルクォーテーションで囲まれた文字列の使用が要求される)。また、結果集合はラストネームの昇順にソートされる（昇順はデフォルトであり、"DESC" キーワードを付加して切り替えることができる。 - "LAST_NAME DESC")。

空白の WHERE 節は、データソース内の全ての行を結果集合に返す。

空の ORDERBY 節を指定することは､データソースが選定条件にマッチした行を全て順番に結果集合に返すことを意味する。この順番がどうなるのか､決定するのは難しい。一般的に､WHERE 条件にマッチした行を探すときにデータソースが使用するインデックスに依存する。注意 - ORDERBY 節を使用しない場合に確実な順番でデータソースが結果を返すことを期待するのは、結局のところ、プログラムに問題を引き起こすことになる。データベースは、結果集合を得る際に、コスト重視、速度重視、または、その他の基準に従う。要するに、指定した順番で結果集合

を得る必要があれば、ORDERBY 節を使用しなければならない。

ORDERBY 節を使用すると、データベースはクライアントに結果集合を返す前にソートしなければならないため、パフォーマンスは悪くなる（performance hit）。的確な ORDERBY の順番でデータを見つけるようにするための効果的なインデックスを作成することは、パフォーマンス向上につながる。多くの場合、データベースは、アプリケーションで（ソートされていない）全てのレコードを読み、それらをソートする場合よりも、もっと早くレコードをソートすることができる。データベースは良く動作する。

上記の例で特筆すべきは、境界データカラム（'AGE'）に含まれないカラムは、結果集合を従属ソートするのに使用される。

この例では、簡単なクエリを使用しテーブルの結合が行われなかったため、FROM 節は空である。FROM 節が空の場合、wxDbTable インスタンスに対し、デフォルトテーブルから、参照される全てのカラムが取得されると仮定される。

選定条件が指定された後、プログラムは取得される結果集合の検索と作成をデータソースに問い合わせることができるようになる。：

```
// where/orderBy/from 節で指定された条件に基づいてクエリを実行するよう、
// データソースに指示する
if (!table->Query())
{
    // クエリ実行時にエラーが発生した
}
```

一般的に、wxDbTable::Query を呼んだときにエラーが発生するのは、指定された WHERE 節のシンタックスのエラーが原因である。wxDbTable::Query（と、データソースに対する全ての操作）の失敗を引き起こした正しい SQL の（データソース固有の）理由は、テーブルのデータソースコネクションの "errorList[]" 配列メンバに格納されたエラー文字列を解析することで判明する。

wxDbTable::Query が true を返すと、そのデータベースは与えられた規準を使用して要求したクエリを完了できた。このことは、結果集合の中に何らかの行が存在することを意味するのではなく、クエリが成功したことだけを意味する。

---

IMPORTANT: The result created by the call to wxDbTable::Query can take one of two forms. It is either a snapshot of the data at the exact moment that the database determined the record matched the search criteria, or it is a pointer to the row that matched the selection criteria. Which form of behavior is datasource dependent. If it is a snapshot, the data may have changed since the result set was constructed, so beware if your datasource uses snapshots and call wxDbTable::Refresh. Most larger brand databases do not use snapshots, but it is important to mention so that your application can handle it properly if your datasource does.

---

To retrieve the data, one of the data fetching routines must be used to request a row from the result set, and to store the data from the result set into the bound memory variables. After wxDbTable::Query has

completed successfully, the default/current cursor is placed so it is pointing just before the first record in the result set. If the result set is empty (no rows matched the criteria), then any calls to retrieve data from the result set will return false.

```
wxString msg;

while (table->GetNext())
{
    msg.Printf("Row #%lu -- First Name : %s  Last Name is %s",
               table->GetRowNum(), FirstName, LastName);
    wxMessageBox(msg, "Data", wxOK | wxICON_INFORMATION, NULL);
}
```

上記サンプルコードは、結果集合の最後に到達するまで、結果集合の中から次のレコードを繰り返し読み出し続ける。wxDbTable::Query 呼び出しが成功した後に、wxDbTable::GetNext が正しく呼び出された初回は、実際には、結果集合の始めのレコードが返される。

wxDbTable::GetNext が呼ばれて、結果集合の現在のカーソルの位置以降に行が残っていない場合、wxDbTable::GetNext は（他の wxDbTable::GetXxxxx() 関数も同様に）false を返す。

## テーブルを閉じる

wxDbTable インスタンスの使用を終了するとき、単にテーブルポインタを delete するだけでよい（あるいは、wxDbTable を静的に宣言した場合、変数がスコープ外に出ればよい）。一般的に、デフォルトデストラクタは、wxDbTable インスタンスを破棄するために要求される全てのものを管理する。

```
if (table)
{
    delete table;
    table = NULL;
}
```

wxDbTable インスタンスを削除するとき、全てのカーソルを開放し、カラム定義を削除し、テーブルによって使用された SQL 環境ハンドルを開放する（しかし、wxDbTable インスタンスが使用していたデータソースコネクションによって使用された環境ハンドルを除く）。

## データソースコネクションを閉じる

データソースを使用していた全てのテーブルがクローズされた後（このことは、wxDb::GetTableCount を呼び出し、戻り値が 0 であることを確認する）、データソースコネクションをクローズする。その方法は、データソースコネクションを取得するときに非キャッシュを使用したか、キャッシュを使用したかに依存する。

データソースコネクションが手動で（キャッシュではなく）作成された場合、コネクションのクローズは下記のように行われる。：

```
if (db)
{
    db->Close();
    delete db;
    db = NULL;
}
```

wxDbConnection 関数がデータソースコネクションを取得するときに使用された場合、以下のコードがコネクションを開放するために使用されなければならない。：

```
if (db)
{
    wxDbFreeConnection(db);
    db = NULL;
}
```

次の wxDbGetConnection 呼び出しで再利用ができるよう、上記のコードはコネクションを開放するのみであることに注意しなければならない。実際にコネクションを整理するために、（環境ハンドル以外の）全てのリソースを開放するために、下記を実行する必要がある。：

```
wxDbCloseConnections();
```

### ODBC 環境ハンドルを開放する

ODBC 環境ハンドル（この例では、"DbConnectInf.Henv" に格納されている）を使用した全てのコネクションを１度クローズすれば、環境ハンドルは安全に開放される。：

```
DbConnectInf->FreeHenv();
```

あるいは、長い書式のコンストラクタが使用され、コンストラクタが自身の SQL 環境ハンドルを確保することが認められれば、スコープ外になるか wxDbConnectInf デストラクタによって自動的にハンドルが開放される。

```
delete DbConnectInf;
```

コネクションがまだハンドルを使用している場合には、環境ハンドルをリリースしてはならない。

## wxODBC - 既知の不具合

As with creating wxWidgets, writing the wxODBC classes was not the simple task of writing an application to run on a single type of computer system. The classes need to be cross-platform for different operating systems, and they also needed to take in to account different database manufacturers and different ODBC driver manufacturers. Because of all the possible combinations of OS/database/drivers, it is impossible to say that these classes will work perfectly with datasource ABC, ODBC driver XYZ, on platform LMN. You may run into some incompatibilities or unsupported features when moving your application from one environment to another. But that is what makes cross-platform programming fun. It also pinpoints one of the great things about open source software. It can evolve!

The most common difference between different database/ODBC driver manufacturers in regards to these wxODBC classes is the lack of standard error codes being returned to the calling program. Sometimes manufacturers have even changed the error codes between versions of their databases/drivers.

In all the tested databases, every effort has been made to determine the correct error codes and handle them in the class members that need to check for specific error codes (such as TABLE DOES NOT EXIST when you try to open a table that has not been created yet). Adding support for additional databases in the future requires adding an entry for the database in the wxDb::Dbms function, and then handling any error codes returned by the datasource that do not match the expected values.

## Databases

Following is a list of known issues and incompatibilities that the wxODBC classes have between different datasources. An up to date listing of known issues can be seen in the comments of the source for wxDb::Dbms.

ORACLE

- Currently the only database supported by the wxODBC classes to support VIEWS

DBASE

NOTE: dBase is not a true ODBC datasource. You only have access to as much functionality as the driver can emulate.

- Does not support the SQL_TIMESTAMP structure
- Supports only one cursor and one connect (apparently? with Microsoft driver only?)
- Does not automatically create the primary index if the 'keyField' param of SetColDef is true. The user must create ALL indexes from their program with calls to wxDbTable::CreateIndex
- Table names can only be 8 characters long
- Column names can only be 10 characters long
- Currently cannot CREATE a dBase table - bug or limitation of the drivers used??
- Currently cannot insert rows that have integer columns - bug??

SYBASE (all)

- To lock a record during QUERY functions, the reserved word 'HOLDLOCK' must be added after every table name involved in the query/join if that table's matching record(s) are to be locked
- Ignores the keywords 'FOR UPDATE'. Use the HOLDLOCK functionality described above

SYBASE (Enterprise)

- If a column is part of the Primary Key, the column cannot be NULL
- Maximum row size is somewhere in the neighborhood of 1920 bytes

mySQL

- If a column is part of the Primary Key, the column cannot be NULL.
- Cannot support selecting for update [wxDbTable::CanSelectForUpdate]. Always returns false.
- Columns that are part of primary or secondary keys must be defined as being NOT NULL when they are created. Some code is added in wxDbTable::CreateIndex to try to adjust the column definition if it is not defined correctly, but it is experimental (as of wxWidgets v2.2.1)
- Does not support sub-queries in SQL statements

POSTGRES

- Does not support the keywords 'ASC' or 'DESC' as of release v6.5.0
- Does not support sub-queries in SQL statements

DB2

- Columns which are part of a primary key must be declared as NOT NULL

## UNICODE with wxODBC classes

As of v2.6 of wxWidgets, the wxODBC classes now fully support the compilation and use of the classes in a Unicode build of wxWidgets, assuming the compiler and OS on which the program will be compiled/run is Unicode capable.

The one major difference in writing code that can be compiled in either unicode or non-unicode builds that is specific to the wxODBC classes is to use the SQL_C_WXCHAR datatype for string columns rather than SQL_C_CHAR or SQL_C_WCHAR.

# wxODBC - サンプルコード

Simplest example of establishing/opening a connection to an ODBC datasource, binding variables to the columns for read/write usage, opening an existing table in the datasource, inserting a record, setting query parameters (where/orderBy/from), querying the datasource, reading each row of the result set, deleting a record, releasing the connection, then cleaning up.

NOTE: Very basic error handling is shown here, to reduce the size of the code and to make it more easily readable. The HandleError() function uses the wxDbLogExtendedErrorMsg() function for retrieving database error messages.

```
// ----------------------------------------------------------------------------
// HEADERS
// ----------------------------------------------------------------------------
#include "wx/log.h"         // #included to enable output of messages only
#include "wx/dbtable.h"

// ----------------------------------------------------------------------------
// FUNCTION USED FOR HANDLING/DISPLAYING ERRORS
// ----------------------------------------------------------------------------
// Very generic error handling function.
// If a connection to the database is passed in, then we retrieve all the
// database errors for the connection and add them to the displayed message
int HandleError(wxString errmsg, wxDb *pDb=NULL)
{
    // Retrieve all the error message for the errors that occurred
    wxString allErrors;
    if (!pDb == NULL)
        // Get the database errors and append them to the error message
        allErrors = wxDbLogExtendedErrorMsg(errmsg.c_str(), pDb, 0, 0);
    else
        allErrors = errmsg;

    // Do whatever you wish with the error message here
    // wxLogDebug() is called inside wxDbLogExtendedErrorMsg() so this
    // console program will show the errors in the console window,
    // but these lines will show the errors in RELEASE builds also
```

```
    wxFprintf(stderr, wxT("\n%s\n"), allErrors.c_str());
    fflush(stderr);

    return 1;
}


// ----------------------------------------------------------------------
// entry point
// ----------------------------------------------------------------------
int main(int argc, char **argv)
{
wxDbConnectInf  *DbConnectInf    = NULL;    // DB connection information

wxDb            *db              = NULL;    // Database connection

wxDbTable       *table           = NULL;    // Data table to access
const wxChar     tableName[]     = wxT("USERS"); // Name of database table
const UWORD      numTableColumns = 2;       // Number table columns
wxChar           FirstName[50+1];           // column data: "FIRST_NAME"
wxChar           LastName[50+1];            // column data: "LAST_NAME"

wxString         msg;                       // Used for display messages

// ----------------------------------------------------------------------
// DEFINE THE CONNECTION HANDLE FOR THE DATABASE
// ----------------------------------------------------------------------
DbConnectInf = new wxDbConnectInf(NULL,
                                  wxT("CONTACTS-SqlServer"),
                                  wxT("sa"),
                                  wxT("abk"));

// Error checking....
if (!DbConnectInf || !DbConnectInf->GetHenv())
{
    return HandleError(wxT("DB ENV ERROR: Cannot allocate ODBC env handle"));
}


// ----------------------------------------------------------------------
// GET A DATABASE CONNECTION
// ----------------------------------------------------------------------
db = wxDbGetConnection(DbConnectInf);

if (!db)
{
    return HandleError(wxT("CONNECTION ERROR - Cannot get DB connection"));
}


// ----------------------------------------------------------------------
// DEFINE THE TABLE, AND THE COLUMNS THAT WILL BE ACCESSED
// ----------------------------------------------------------------------
table = new wxDbTable(db, tableName, numTableColumns, wxT(""),
                  !wxDB_QUERY_ONLY, wxT(""));
//
// Bind the columns that you wish to retrieve. Note that there must be
// 'numTableColumns' calls to SetColDefs(), to match the wxDbTable def
//
// Not all columns need to be bound, only columns whose values are to be
// returned back to the client.
//
table->SetColDefs(0, wxT("FIRST_NAME"), DB_DATA_TYPE_VARCHAR, FirstName,
               SQL_C_WXCHAR, sizeof(FirstName), true, true);
table->SetColDefs(1, wxT("LAST_NAME"), DB_DATA_TYPE_VARCHAR, LastName,
               SQL_C_WXCHAR, sizeof(LastName), true, true);


// ----------------------------------------------------------------------
// CREATE (or RECREATE) THE TABLE IN THE DATABASE
// ----------------------------------------------------------------------
if (!table->CreateTable(true))  //NOTE: No CommitTrans is required
{
    return HandleError(wxT("TABLE CREATION ERROR: "), table->GetDb());
}


// ----------------------------------------------------------------------
// OPEN THE TABLE FOR ACCESS
// ----------------------------------------------------------------------
if (!table->Open())
{
```

```
    return HandleError(wxT("TABLE OPEN ERROR: "), table->GetDb());
}


// ---------------------------------------------------------------------
// INSERT A NEW ROW INTO THE TABLE
// ---------------------------------------------------------------------
wxStrcpy(FirstName, wxT("JULIAN"));
wxStrcpy(LastName, wxT("SMART"));
if (!table->Insert())
{
    return HandleError(wxT("INSERTION ERROR: "), table->GetDb());
}

// Must commit the insert to write the data to the DB
table->GetDb()->CommitTrans();


// ---------------------------------------------------------------------
// RETRIEVE ROWS FROM THE TABLE BASED ON SUPPLIED CRITERIA
// ---------------------------------------------------------------------
// Set the WHERE clause to limit the result set to return
// all rows that have a value of 'JULIAN' in the FIRST_NAME
// column of the table.
table->SetWhereClause(wxT("FIRST_NAME = 'JULIAN'"));

// Result set will be sorted in ascending alphabetical
// order on the data in the 'LAST_NAME' column of each row
table->SetOrderByClause(wxT("LAST_NAME"));

// No other tables (joins) are used for this query
table->SetFromClause(wxT(""));

// Instruct the datasource to perform a query based on the
// criteria specified above in the where/orderBy/from clauses.
if (!table->Query())
{
    return HandleError(wxT("QUERY ERROR: "), table->GetDb());
}

// Loop through all rows matching the query criteria until
// there are no more records to read
while (table->GetNext())
{
    msg.Printf(wxT("Row #%lu -- First Name : %s  Last Name is %s"),
            table->GetRowNum(), FirstName, LastName);

    // Code to display 'msg' here
    wxLogMessage(wxT("\n%s\n"), msg.c_str());
}


// ---------------------------------------------------------------------
// DELETE A ROW FROM THE TABLE
// ---------------------------------------------------------------------
// Select the row which has FIRST_NAME of 'JULIAN' and LAST_NAME
// of 'SMART', then delete the retrieved row
//
if (!table->DeleteWhere(wxT("FIRST_NAME = 'JULIAN' and LAST_NAME = 'SMART'")))
{
    return HandleError(wxT("DELETION ERROR: "), table->GetDb());
}

// Must commit the deletion to the database
table->GetDb()->CommitTrans();


// ---------------------------------------------------------------------
// TAKE CARE OF THE ODBC CLASS INSTANCES THAT WERE BEING USED
// ---------------------------------------------------------------------
// If the wxDbTable instance was successfully created
// then delete it as we are done with it now.
wxDELETE(table);

// Free the cached connection
// (meaning release it back in to the cache of datasource
// connections) for the next time a call to wxDbGetConnection()
// is made.
wxDbFreeConnection(db);
db = NULL;
```

```
// ----------------------------------------------------------------------
// CLEANUP BEFORE EXITING APP
// ----------------------------------------------------------------------
// The program is now ending, so we need to close
// any cached connections that are still being
// maintained.
wxDbCloseConnections();

// Release the environment handle that was created
// for use with the ODBC datasource connections
wxDELETE(DbConnectInf);

wxUnusedVar(argc);  // Here just to prevent compiler warnings
wxUnusedVar(argv);  // Here just to prevent compiler warnings

return 0;
}
```

# wxODBC - SQL コマンドの選択

The following is a very brief description of some common SQL commands, with examples.

See also

Database classes overview

## Create

Creates a table.

Example:

```
CREATE TABLE Book
 (BookNumber     INTEGER     PRIMARY KEY
 , CategoryCode  CHAR(2)     DEFAULT 'RO' NOT NULL
 , Title         VARCHAR(100) UNIQUE
 , NumberOfPages SMALLINT
 , RetailPriceAmount NUMERIC(5,2)
 )
```

## Insert

Inserts records into a table.

Example:

```
INSERT INTO Book
  (BookNumber, CategoryCode, Title)
  VALUES(5, 'HR', 'The Lark Ascending')
```

## Select

The Select operation retrieves rows and columns from a table. The criteria for selection and the columns returned may be specified.

Examples:

SELECT * FROM Book

Selects all rows and columns from table Book.

SELECT Title, RetailPriceAmount FROM Book WHERE RetailPriceAmount > 20.0

Selects columns Title and RetailPriceAmount from table Book, returning only the rows that match the WHERE clause.

SELECT * FROM Book WHERE CatCode = 'LL' OR CatCode = 'RR'

Selects all columns from table Book, returning only the rows that match the WHERE clause.

SELECT * FROM Book WHERE CatCode IS NULL

Selects all columns from table Book, returning only rows where the CatCode column is NULL.

SELECT * FROM Book ORDER BY Title

Selects all columns from table Book, ordering by Title, in ascending order. To specify descending order, add DESC after the ORDER BY Title clause.

SELECT Title FROM Book WHERE RetailPriceAmount >= 20.0 AND RetailPriceAmount <= 35.0

Selects records where RetailPriceAmount conforms to the WHERE expression.

## Update

Updates records in a table.

Example:

UPDATE Incident SET X = 123 WHERE ASSET = 'BD34'

This example sets a field in column 'X' to the number 123, for the record where the column ASSET has the value 'BD34'.