【A2】Delphi/C++テクニカルセッション

EMBARCADERO TECHNOLOGIES®

DEVELOPER CAMP

Delphi/C++Buider DB総ざらい（C++Builder編）

株式会社　日本情報システム
筑木真志

アジェンダ

- dbGoを使った接続
- C++Builderで埋め込みSQLを使用する

EMBARCADERO
TECHNOLOGIES®

DEVELOPER CAMP

dbGoを使った接続

dbGo（旧名：ADO Express）概要

- dbGoはADO(ActiveX Data Objects)経由でRDBにアクセスする

- Professional版ではサポートしていない、RDBにも接続可能
  - Microsoft SQL Server 2000/2005/2008
  - Microsoft Access 2003 (Microsoft JET 3.x/4.x)
  - Microsoft Access 2007 (Microsoft Office 12.0 Access Database Engine)
  - Oracle 8i/9i/10g/11g
  - IBM DB2 8.x/9.x
  - など

- 基本的な使用方法はDBExpressと変わりない
  - TSQLConnection → TADOConnection
  - TSQLTable → TADOTable

OLE DB プロバイダの選択

- Oracleの場合
  - Oracle Provider for OLE DB　(Oracle製)
  - Microsoft OLE DB Provider for Oracle （Microsoft製）
- SQL Serverの場合
  - SQL Server 2008 Native Client
  - Microsoft OLE DB Provider for SQL Server

接続出来ない場合の確認事項

- 標準のクライアントから普通に接続出来るか？
  - SQL Plus / SQL Developer (Oracle)
  - SQL Server Management Studio (SQL Server)
  - コントロール・センター (DB2)
  - など

- TADOTableのTableNameプロパティにスキーマ名を含める

  例） スキーマ名：Foo テーブル名：Bar

  TableName = "Bar"; → TableName = "Foo.Bar";

- TADOConnectionのCursorLocationプロパティの値を変更する
  - clUseServer（カーソルをサーバー側で持つ）に変更する

EMBARCADERO
TECHNOLOGIES®

DEVELOPER CAMP

C++Builderで埋め込みSQLを使用する

埋め込みSQLとは

- C/C++のソースコードに直接SQL文を記述して実行する

```
int main(int argc, char* argv[])
{
  EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR filename[8];
  VARCHAR mapname[128];
  double frame_lu_b;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL CONNECT :user IDENTIFIED BY :passwd USING :host;

  EXEC SQL DECLARE CURSOR1 CURSOR FOR
    SELECT FILENAME, MAPNAME, FRAME_LU_B FROM MAPTABLE ORDER BY FILENAME;

  EXEC SQL OPEN CURSOR1;
  EXEC SQL WHENEVER NOT FOUND DO BREAK;

  while (1) {
     EXEC SQL FETCH CURSOR1 INTO :filename, :mapname, :frame_lu_b;
     printf("FILENAME = %s, MAPNAME = %s frane_lu_b = %.4f¥n",
       filename.arr, mapname.arr, frame_lu_b);
  }

  EXEC SQL CLOSE CURSOR1;

  return 0;
}
```

DBコンポーネントとの違い

- 特徴
  - データベースとのやりとりはソースコードに直接記述する
  - プリコンパイラでホスト言語(C/C++)に変換してからコンパイルする
  - データベースのAPIを直接呼び出すのでパフォーマンスがよい
  - クエリ結果を配列にまとめてフェッチできる

- 欠点
  - デバッグが非常に面倒
  - データベースの実装に大きく依存するため汎用性が悪くなる
  - 文字コードの変換等は自前で行う必要がある
  - GUIとのやりとりも自前で行う必要がある

データベースとのやりとり

- ホスト変数
  - ホスト言語と埋め込みSQLとの間でデータをやりとるする変数
  - C/C++の基本型のみ可（Oracleの場合、VARCHAR型が用意）
  - ホスト変数は、BEGIN DECLARE SECTIONとEND DECLARE SECTIONでくくる

```
EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR filename[8];
  VARCHAR mapname[128];
  double frame_lu_b;
EXEC SQL END DECLARE SECTION;
```

  - ホスト変数をSQLに組み込むには、変数の前にコロン（:）をつける

```
EXEC SQL FETCH CURSOR1 INTO :filename, :mapname, :frame_lu_b;
```

- 実行結果
  - 変数sqlca.sqlcodeでSQLの実行結果（エラーコード）を参照する

Oracleの場合

- coff2omf.exeを使って、$(ORACLEHOME)¥precomp¥LIB¥orasql11.libをC++Builderで使用できるようにする

```
coff2omf orasql11.lib orasql11omf.lib
```

- プリコンパイラの実行

```
proc code=cpp cpp_suffix=cpp File1.pc
```

DB2の場合

- implib.exeを使って、$(DB2PATH)¥bin¥db2app.dllのC++Builder用のインポートライブラリを作る

```
implib db2app.dll db2appOMF.lib
```

- プリコンパイラの実行
  - DB2のコマンド・ウィンドウで以下を実行する

```
db2 connect to db2 user scott using tiger
db2 prep DB2Test.sqx OUTPUT DB2Test.cpp
db2 connect reset
```

- インクルードパスの設定
  - $(DB2PATH)¥includeを検索パスの最初にする