Technical Note 04-48
ODBC プラグインコマンド

By Jamras Komoncharoensiri, 4D Evangelist
Technical Note 04-48

（原題: 2004 Advanced ODBC）

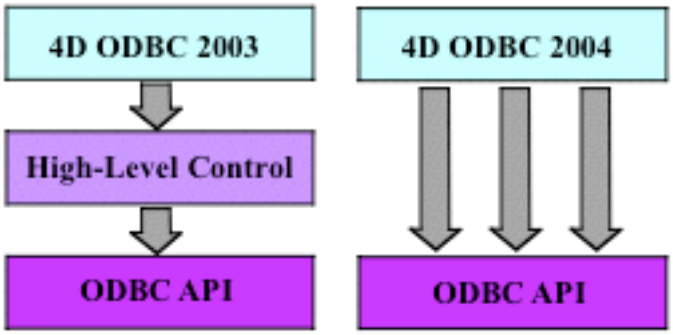
概要

この Tech Note では、4D ODBC Pro プラグインコマンドの概要を紹介しています。

4D では、ODBC はバージョン 2003 までプラグインとして提供されていました。4D ODBC には、ハイレベル/ローレベルコマンドが用意されていましたが、実のところプラグインはハイレベルに位置し、ODBC API との間にハイレベルコントロールのレイヤが一枚、存在しました。

バージョン 2004 では、ハイレベルコマンドは 4D のネイティブコマンドとして統合され、接続を確立や典型的な SQL ステートメントの実行などの簡単な処理は 4D 単体で可能になりました。

これに対し、まったく新たに作り直された 4D ODBC Pro プラグインは、完全な ODBC 互換性を 4D に付与することを目的としています。実際、プラグインコマンドはすべて冒頭に ODBC_がつく点を除けば、Microsoft 社の ODBC API で提供されているコマンド名とまったく同じです。これは従来のハイレベルコントロールレイヤが取り払われたことを意味します。



4D ODBC Pro プラグインの特徴

4D ODBC Pro プラグインは、最新のバージョンである ODBC 3.0 API に準拠して開発されました。結果として以前の ODBC プラグインではサポートされていなかった制御が可能になり、MacOS X でも動作します。

デフォルトで ODBC ドライバは同期モードで動作します。つまり、ある機能をコールされると、ドライバが処理を完了するまではアプリケーションに制御を戻しません。とはいえある種の機能は非同期、つまり、コールされて最低限の処理を済ませた後、速やかに制御をアプリケーションに戻すという動作でも構いません。4D ODBC Pro は以前のバージョンには欠けていた非同期モードをサポートしています。(非同期でコールできる機能はドライバによって異なります)

さらにブックマーク機能もサポートされるようになりました。ブックマークとは、ある行を特定するために使用される値のことです。ブックマークの内容は、ドライバとデータソースだけが把握しています。単なる行番号の場合もあれば、物理ディスクのアドレスである場合もあります。

最後に、バルクオペレーションもサポートされるようになりました。レコードの配列に対して発行することのできる同オペレーションは、ファンクションコールの回数だけでなくデータベースアクセス回数の削減にもつながります。より効率的にデータがバッファ保存されるからです。バルクオペレーションにより INSERT、UPDATE、DELETE などの SQL をループせずに最小限の時間で終えることができます。

必要なもの

ODBC Pro プラグインは、4th Dimension および 4D Server、ビルドされた 4D Runtime 用に提供されています。過去の ODBC ドライバ同様、利用には ODBC アドミニストレータおよび ODBC ドライバが必要です。前者は Windows/MacOS X ともにデフォルトで用意されていますが、後者は MacOS X の場合 http://www.openlinksw.com などから入手する必要があるかもしれません。

ODBC コマンドの使用

コマンドの記述形式は、3 パターンに分類することができます。

まずは結果セットを必要としない SQL ステートメント、次に結果としてレコードのセットを取得するもの（SELECT 文)、最後がストラクチャの情報を取得するものです。3 番目のパターンに関しては、プラグインが ODBC API からコマンドをマッピングしているので SQL を必要としません。

パターン 1

```
` Establish a connection to a datasource
$iResult:= ODBC_SQLAllocConnect($iConnID)
$iResult:= ODBC_SQLConnect($iConnID;$tDataSource;$tUName;$tPswd)
$iResult:= ODBC_SQLAllocStmt($iConnID;$iStmtID)
$iResult:= ODBC_SQLPrepare($iStmtID;$tSQL)
$iResult:= ODBC_SQLExecute($iStmtID)
` Release the SQL cursor/process and terminate the connection
$iResult:= ODBC_SQLCancel($iStmtID)
$iResult:= ODBC_SQLFreeStmt($iStmtID;SQL_UNBIND)
$iResult:= ODBC_SQLDisconnect($iConnID)
```

パターン 2

```
` Establish a connection to a datasource
$iResult:= ODBC_SQLAllocConnect($iConnID)
$iResult:= ODBC_SQLConnect($iConnID;$tDataSource;$tUName;$tPswd)
$iResult:= ODBC_SQLAllocStmt($iConnID;$iStmtID)
$iResult:= ODBC_SQLPrepare($iStmtID;$tSQL)
$iResult:= ODBC_SQLBindCol ($iStmtID;1;->array_1;->indicator)
$iResult:= ODBC_SQLBindCol ($iStmtID;2;->array_2;->indicator)
...
$iResult:= ODBC_SQLBindCol ($iStmtID;N;->array_N;->indicator)
$iResult:= ODBC_SQLExecute($iStmtID)
Repeat
$iResult:= ODBC_SQLFetch ($iStmtID)
Until ($iResult#0)
` Release the SQL cursor/process and terminate the connection
$iResult:= ODBC_SQLCancel($iStmtID)
$iResult:= ODBC_SQLFreeStmt($iStmtID;SQL_UNBIND)
$iResult:= ODBC_SQLDisconnect($iConnID)
```

パターン 3

```
` Establish a connection to a datasource
$iResult:= ODBC_SQLAllocConnect($iConnID)
$iResult:= ODBC_SQLConnect($iConnID;$tDataSource;$tUName;$tPswd)
$iResult:= ODBC_SQLAllocStmt($iConnID;$iStmtID)
` ODBC_SQLTables, ODBC_SQLColumns, etc.
$iResult:= ODBC_SQLBindCol ($iStmtID;1;->array_1;->indicator)
$iResult:= ODBC_SQLBindCol ($iStmtID;2;->array_2;->indicator)
...
$iResult:= ODBC_SQLBindCol ($iStmtID;N;->array_N;->indicator)
Repeat
$iResult:= ODBC_SQLFetch ($iStmtID)
Until ($iResult#0)
` Release the SQL cursor/process and terminate the connection
$iResult:= ODBC_SQLCancel($iStmtID)
$iResult:= ODBC_SQLFreeStmt($iStmtID;SQL_UNBIND)
$iResult:= ODBC_SQLDisconnect($iConnID)
```

非同期モード

非同期モードは ODBC API 3.0 に準拠したバージョン 2004 ではじめて実現しています。この機能はデータベースのパフォーマンスを飛躍的に向上させる可能性を秘めています。

同期/非同期はコマンド ODBC_SQLSetConnectAttr に定数 SQL ATTR ASYNC ENABLE ON あるいは OFF を渡すことで設定できます。設定は接続ごとに行なうことができます。

ブックマークオペレーション

ブックマークを使用すれば、特定の結果セットを記憶させることができ、レコードをリロードしなくても後で参照することができます。

ブックマークは、ODBC_SQLSetStmtAttr に定数 SQL_ATTR_USE_BOOKMARKS を渡し、属性を SQL_UB_VARIABLE にしてからステートメントを Prepare します。

```
attrValue:=4 `To be fetched 4 row at the time
$Result:=ODBC_SQLSetStmtAttr($iStmtID;SQL_ATTR_ROW_ARRAY_SIZE;->attrValue)
attrValue:=SQL_UB_VARIABLE
$Result:=ODBC_SQLSetStmtAttr($iStmtID;SQL_ATTR_USE_BOOKMARKS;->attrValue)
$Result:=ODBC_SQLPrepare($iStmtID;$tSQL)
$Result:=ODBC_SQLExecute($iStmtID)
```

ステートメントが実行されるとブックマークは結果セットの 0 番列として返されます。0 番列は ODBC_SQLGetData で BLOB に受け取ることができます。

```
$Result:=ODBC_SQLGetData($iStmtID;0;->myBlob)
$Result:=ODBC_SQLSetStmtAttr($iStmtID;SQL_ATTR_FETCH_BOOKMARK_PTR;->myBlob)
```

ひとたびブックマークポインタが設定されれば、ODBC_SQLFetchScroll でその結果セットを再取得することができます。

```
$Result:=ODBC_SQLFetchScroll($iStmtID;8;0)  `SQL_FETCH_BOOKMARK=8
```

バルクオペレーション

バルクオペレーションによって、INSERT や種々のブックマーク操作をまとめて行なうことができます。コマンドは ODBC_SQLBulkOperations です。

バルク INSERT

クエリを実行して結果セットを作成します。

ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE を挿入する行の数に設定します。

挿入するデータ(上記と同じ要素数の配列)を ODBC_SQLBindCol でバインドします。

注記:
このときステートメント属性 SQL_ATTR_ROW_STATUS_PTR はヌルポインタ、あるいは指す配列のサイズが SQL_ATTR_ROW_ARRAY_SIZE と同一であるべきです。

ODBC_SQLBulkOperations($iStmtID;SQL_ADD)を実行します。

SQL_ATTR_ROW_STATUS_PTR が設定されていれば、結果を調べることができます。

ブックマークのバルク操作

ステートメント属性 SQL_ATTR_USE_BOOKMARKS を SQL_UB_VARIABLE にします。

クエリを実行して結果セットを作成します。

ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE を更新する行の数に設定します。

ODBC_SQLBindCol で更新するデータをバインドします。

ブックマークを取得するために再度 ODBC_SQLBindCol をコールします。(0 番列の配列)

バインドされたバッファのデータを更新します。

注記:
このときステートメント属性 SQL_ATTR_ROW_STATUS_PTR はヌルポインタ、あるいは指す配列のサイズが SQL_ATTR_ROW_ARRAY_SIZE と同一であるべきです。

ODBC_SQLBulkOperations($iStmtID;SQL_UPDATE_BY_BOOK MARK)を実行します。

SQL_ATTR_ROW_STATUS_PTR が設定されていれば、結果を調べることができます。データが更新されると、ドライバが配列の該当する値を SQL_ROW_UPDATED に変更します。

ODBC_SQLBulkOperations ($iStmtID;SQL_FETCH_BY_BOOKMARK)を実行すればデータをバッファに読み込んで更新が行なわれたことを確認できます。