CLR 外部環境

SQL Anywhere Ver.11 は、6 つの外部ランタイム環境に対応しています。その例としては、C/C++ で記述した Embedded SQL や ODBC アプリケーションだけでなく、Java、Perl、PHP で記述されたアプリケーション、あるいは Microsoft の CLR (Common Language Runtime：共通言語ランタイム) をベースとした C# や Visual Basic などの言語で記述されたアプリケーションなどが挙げられます。

SQL Anywhere は、CLR のストアド・プロシージャや関数をサポートしています。CLR のストアド・プロシージャまたは関数は、SQL のストアド・プロシージャや関数と同様に動作します。ただし、プロシージャまたは関数のコードが C# または Visual Basic で記述されている場合や、プロシージャまたは関数がサーバ外部で実行される場合を除きます (つまり、個別の .NET 実行プログラム内部で実行)。この .NET 実行プログラムは、データベース当たりのインスタンスが 1 つだけであることに注意してください。CLR 関数とストアド・プロシージャを実行する接続では、すべて同じ .NET 実行プログラムのインスタンスを使用します。ただし、各接続用のネームスペースは、個別に設定します。接続の間は、静的状態が持続しますが、接続上でその状態を共有することはできません。

CLR 外部環境サンプル

ここでは、データベース・サーバが CLR 関数を実行し、Visual Studio 2008 用ライブラリの C# 言語を使用したサンプル・ソリューションが定義する 2 種類のメソッドを呼び出すという例について説明します。これらのメソッドからの戻り値は、SQL Anywhere データベースを移植する際に使用します。

要求事項

- Visual Studio 2008
- Visual Studio 2008 C# サンプル – 以下からも入手可能です。
  http://code.msdn.microsoft.com/csharpsamples

サンプルの実行

1. Visual Studio を起動して、ソリューション *LanguageSamples¥Libraries¥Libraries.sln* を開きます。

2. ソリューションを構築して、*Functions.dll* アセンブリを作成します。このアセンブリには、NumberOfDigits() と Factorial() の 2 つのメソッドがあり、CLR 関数から呼び出すことができます。

3. Windows Explorer を使用して、*Functions.dll* アセンブリを検索し、*C:¥Program Files¥SQL Anywhere 11¥Bin32* フォルダにコピーします。CLR 外部実行プログラム *dbextclr11.exe* が、DLL 用のパス環境変数を検索します。この実行プログラムは、GAC (Global Assembly Cache：グローバル・アセンブリ・キャッシュ) の検索は行わないのでご注意ください。

4. コマンド・プロンプトを開き、以下のコマンドを実行して、SQL Anywhere データベースを新規で作成します。
   dbinit clrtest.db

5. 以下のコマンドを実行して、データベースを起動します。
   dbeng11 clrtest.db

6. Interactive SQL を起動し、以下のコマンドを実行してデータベースに接続します。
   dbisql -c "uid=dba;pwd=sql;eng=clrtest"

7. CLR 関数の結果を保存するために、簡単なテーブルを作成します。以下の文を Interactive SQL にコピーして貼り付け、[**F5**] を押して実行します。

   ```
   CREATE TABLE clr_test (
      num       INTEGER NOT NULL,
      digits    INTEGER NULL,
      factorial INTEGER NULL,
      PRIMARY KEY (num)
   );
   ```

   このテーブルは、CLR 関数からの戻り値を、digits と factorial という 2 つのカラムに保存します。

8. 1 つ目の CLR 関数を作成します。以下の文を Interactive SQL にコピーして貼り付け、[**F5**] を押して実行します。

   ```
   CREATE FUNCTION clr_DigitCount(IN num LONG VARCHAR)
   RETURNS INT
   EXTERNAL NAME 'Functions.dll::Functions.DigitCount.NumberOfDigits(string) int'
   LANGUAGE CLR;
   ```

   この関数は、*Functions.dll* アセンブリで定義されている NumberOfDigits() メソッドを呼び出します。このメソッドは、文字列としてフォーマットされている整数を取り出して、その桁数を返します。この値は、CLR 関数の戻り値となります。

9. 2 つ目の CLR 関数を作成します。以下の文を Interactive SQL にコピーして貼り付け、[**F5**] を押して実行します。

```
CREATE FUNCTION clr_Factorial(IN num INT)
RETURNS INT
EXTERNAL NAME 'Functions.dll::Functions.Factorial.Calc(int) int'
LANGUAGE CLR;
```

ここでも、 *Functions.dll* アセンブリが Factorial() メソッドを定義するため、このメソッドは整数を取り出して、階乗 (*n*!) を戻します。これは、CLR 関数の戻り値となります。

10. CLR 関数を実行するためには、ストアド・プロシージャを作成して 2 種類の関数を呼び出し、これらの戻り値を使用してデータベースを移植します。以下の文を Interactive SQL にコピーして貼り付け、[**F5**] を押して実行します。

```
CREATE PROCEDURE sp_PopulateTable()
BEGIN
   CREATE VARIABLE i INTEGER;
   SET i = 1;

   WHILE i <= 12 LOOP
      INSERT INTO clr_test VALUES( i, clr_DigitCount( STRING(i) ),
clr_Factorial(i) );
      SET i = i + 1;
   END LOOP;
END
```

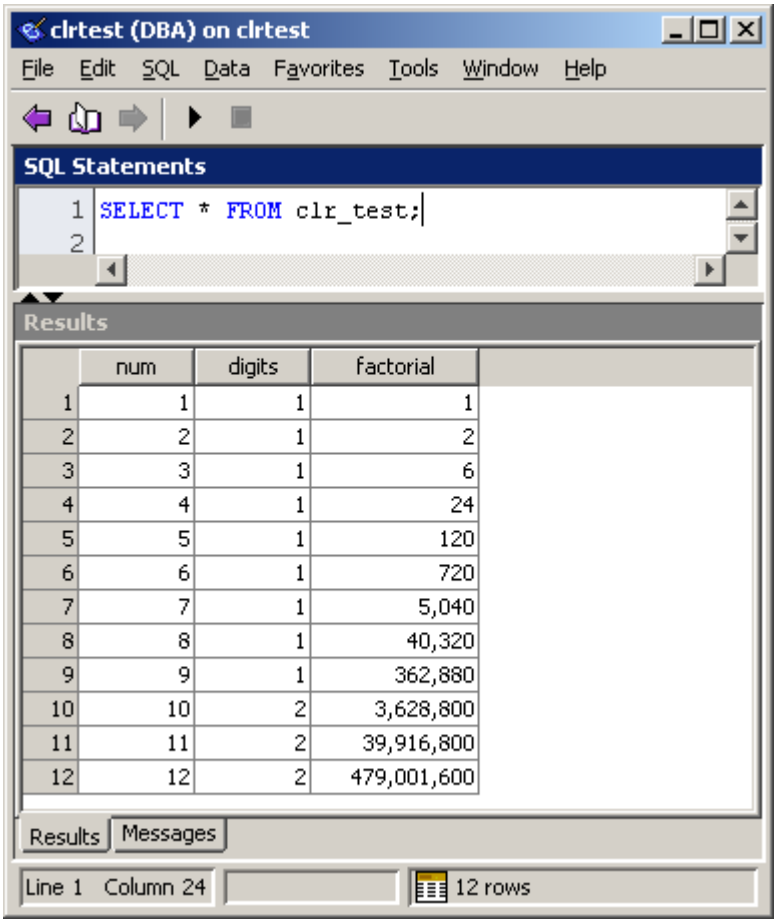
このストアド・プロシージャは、CLR 関数を呼び出し、戻り値を "clr_test" テーブルに挿入します。

11. 次に、Interactive SQL でストアド・プロシージャを実行します。

```
CALL sp_PopulateTable();
```

12. ストアド・プロシージャは、CLR 関数の入力パラメータとして、1～12 までの数字を使用します。結果の値は、"clr_test" テーブルに挿入されます。以下の文を実行して、コンテンツを取り出します。

```
SELECT * FROM clr_test;
```



「桁数」のカラム ([digits]) には、CLR 関数 "clr_DigitCount" を使用してデータが移植されます。一方「階乗」のカラム ([factorial]) には、CLR 関数 "clr_Factorial" を使用してデータが移植されます。関数はそれぞれ、*Functions.dll* アセンブリで定義されたメソッドを呼び出します。

13. これでサンプルは終了です。Interactive SQL とデータベース・サーバを終了してください。また、*Functions.dll* ファイルは、*C:¥Program Files¥SQL Anywhere 11¥Bin32* フォルダから削除してもかまいません。

まとめ

外部ランタイム環境では、SQL 以外の言語でストアド・プロシージャや関数を記述することができます。このため、既存のライブラリを再利用できるというメリットがあるだけでなく、開発者が C# や VB.NET などといった既存のプログラミング言語に関する知識を生かせるため、実装時間の削減も実現できます。