

Chương 3: Truy cập cơ sở dữ liệu với .NET

Tổng quan

Trong chương này, chúng ta sẽ bàn về cách làm sao để một chương trình C# sử dụng ADO.NET. Kết thúc chương này, chúng ta sẽ có được các kiến thức sau:

- Các kết nối cơ sở dữ liệu - làm sao để có thể sử dụng các lớp mới SqlConnection và OleDbConnection để kết nối và hủy kết nối với cơ sở dữ liệu. Các kết nối dùng các kiểu giống như chuỗi kết nối của các trình cung cấp OLEDB. Sau đó chúng ta sẽ làm thử một vài kết nối cơ sở dữ liệu, và phải bảo đảm rằng kết nối sẽ được đóng lại sau khi dùng, thông qua một vài ứng dụng đơn giản.
- Các lệnh thực thi - ADO.NET chứa một đối tượng command, thực thi SQL, hoặc có thể phát ra một stored procedure để trả về các giá trị. Các tùy chọn khác của đối tượng command sẽ được bàn kĩ, với các ví dụ cho từng tùy chọn được đưa ra trong các lớp Sql và OleDb.
- Stored Procedures - Làm sao để gọi các stored procedure bằng các đối tượng command, và làm sao kết hợp các giá trị trả về với dữ liệu trên trình khách.
- The ADO.NET object model - đây là một cách truyền đạt khác đến những đối tượng có sẵn với ADO, và các lớp DataSet, DataTable, DataRow, và DataColumn sẽ được bàn kĩ. Một DataSet có thể bao gồm các quan hệ giữa các table, cũng như các ràng buộc. Chúng sẽ được bàn kĩ.

3.1 Giới thiệu về ADO.NET

Giống như hầu hết các thành phần của .NET Framework, ADO.NET không chỉ là vỏ bọc của một vài API sẵn có. Nó chỉ giống ADO ở cái tên - các lớp và phương thức truy xuất dữ liệu đều khác hoàn toàn.

ADO (Microsoft's ActiveX Data Objects) là một thư viện của các thành phần COM đã từng được ca ngợi trong một vài năm trở lại đây. Phiên bản hiện tại là 2.7, các thành phần chủ yếu của ADO là Connection, Command, Recordset, và các Field object. Một connection có thể mở cơ sở dữ liệu, một vài dữ liệu được chọn vào một recordset, bao gồm các trường, dữ liệu này sau đó có thể thao tác, cập nhập lên server, và connection cần phải được đóng lại. ADO cũng giới thiệu một disconnected recordset, cái được dùng khi không muốn giữ kết nối trong một thời gian dài.

Có một vài vấn đề với ADO đó là sự không hài lòng về địa chỉ, sự công kênh của một disconnected recordset. Hỗ trợ này không cần thiết với sự tiến hoá của tin học "web-centric", vì vậy nó cần được loại bỏ. Có một số giống nhau giữa lập trình ADO.NET và ADO (không phải ở cái tên), vì thế việc chuyển từ ADO không quá khó khăn. Hơn thế nữa, nếu bạn dùng SQL Server, có một bộ các quản mới rất tuyệt cho việc thao tác bên ngoài cơ sở dữ liệu. Chừng đó lí do cũng đủ để các bạn quan tâm đến ADO.NET.

ADO.NET chứa hai không gian tên cơ sở dữ liệu - một cho SQL Server, và một cái khác cho các cơ sở dữ liệu được trình bày thông qua một giao diện OLE DB. Nếu cơ sở dữ liệu của bạn chọn là một bộ phận của OLE DB, bạn có thể dễ dàng kết nối với nó từ .NET - chỉ cần dùng các lớp OLE DB và kết nối thông qua các driver cơ sở dữ liệu hiện hành của bạn.

3.1.1 Các Namespace

Tất cả các ví dụ trong chương này truy xuất dữ liệu trong một vài cách. Các không gian tên sau chỉ ra các lớp và các giao diện được dùng cho việc truy xuất dữ liệu trong .NET:

- System.Data - Các lớp truy xuất dữ liệu chung
- System.Data.Common - Các lớp dùng chung bởi các data provider khác nhau
- System.Data.OleDb - Các lớp của OLE DB provider
- System.Data.SqlClient - Các lớp của SQL Server provider

- System.Data.SqlTypes - Các kiểu của SQL Server

Các lớp chính trong ADO.NET được liệt kê dưới đây:

3.1.2 Các lớp dùng chung

ADO.NET chứa một số lớp được dùng không quan tâm là bạn đang dùng các lớp của SQL Server hay là các lớp của OLE DB.

Các lớp trong không gian tên System.Data được liệt kê sau đây:

- DataSet - Đối tượng này chứa một bộ các DataTable, có thể bao gồm quan hệ giữa các bảng, và nó được thiết kế cho truy xuất dữ liệu không kết nối.
- DataTable - Một kho chứa dữ liệu. Một DataTable bao gồm một hoặc nhiều DataColumn, và khi được tạo ra nó sẽ có một hoặc nhiều DataRow chứa dữ liệu.
- DataRow - Một bộ giá trị, có bà con với một dòng trong bảng cơ sở dữ liệu, hoặc một dòng của bảng tính.
- DataColumn - Chứa cả định nghĩa của một cột, chẳng hạn như tên và kiểu dữ liệu.
- DataRelation - Một liên kết giữa hai DataTable trong một DataSet. Sử dụng cho khóa ngoại và các mối quan hệ chủ tớ.
- Constraint - Định nghĩa một qui tắc cho một DataColumn (hoặc một bộ các cột dữ liệu), như các giá trị là độc nhất.

Sau đây là hai lớp được tìm thấy trong không gian tên System.Data.Common:

- DataColumnMapping - Ánh xạ tên của một cột từ cơ sở dữ liệu vào tên của một cột trong một DataTable.
- DataTableMapping - Ánh xạ tên của một bảng từ cơ sở dữ liệu vào một bảng trong một DataSet.

3.1.3 Các lớp cơ sở dữ liệu chuyên biệt

Bổ sung cho các lớp dùng chung ở trên, ADO.NET có một số các lớp dữ liệu chuyên biệt được đưa ra dưới đây. Các lớp này thực thi một bộ các giao diện chuẩn được định nghĩa trong không gian tên System.Data, cho phép sử dụng các lớp có cùng kiểu giao diện. Ví dụ cả hai lớp SqlConnection và OleDbConnection thực thi giao diện IDbConnection.

- SqlCommand, OleDbCommand - Một vỏ bọc của các câu lệnh SQL hoặc các lời gọi stored procedure.
- SqlCommandBuilder, OleDbCommandBuilder - Một lớp sử dụng các câu lệnh SQL (chẳng hạn như các câu lệnh INSERT, UPDATE, và DELETE) từ một câu lệnh SELECT.
- SqlConnection, OleDbConnection - Kết nối với cơ sở dữ liệu. Giống như một ADO Connection.
- SqlDataAdapter, OleDbDataAdapter - Một lớp giữ các câu lệnh select, insert, update, và delete, chúng được sử dụng để tạo một DataSet và cập nhật Database.
- SqlDataReader, OleDbDataReader - Chỉ đọc, kết nối với data reader.
- SqlParameter, OleDbParameter - Định nghĩa một tham số cho một stored procedure.
- SqlTransaction, OleDbTransaction - Một giao tiếp cơ sở dữ liệu, được bọc trong một đối tượng.

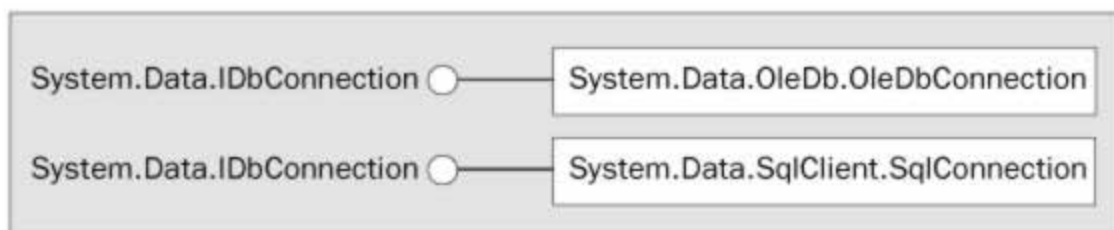
Một đặc tính quan trọng của các lớp ADO.NET là chúng được thiết kế để làm việc trong môi trường không kết nối, đóng một vai trò quan trọng trong thế giới "web-centric". Nó hiện được dùng để kiến trúc một server (chẳng hạn như mua sách qua mạng) để kết nối một server, lấy một vài dữ liệu, và làm việc trên những dữ liệu này trên PC khách trước khi kết nối lại và truyền dữ liệu trở lại để xử lý.

ADO 2.1 giới thiệu recordset không kết nối, nó cho phép dữ liệu có thể được lấy từ một cơ sở dữ liệu, được truyền cho trình khách để xử lý. Nó thường khó xử dụng do

cách ứng xử không kết không được thiết kế từ đầu. Các lớp ADO.NET thì khác - Sql/OleDb DataReader được thiết kế cho để dùng cho các cơ sở dữ liệu offline.

3.2 Sử dụng Database Connection

Trong trìn tự truy xuất cơ sở dữ liệu, bạn cần cung cấp các thông số kết nối, chẳng hạn như thiết bị mà cơ sở dữ liệu đang chạy, và khả năng đăng nhập của bạn. Bất kì ai đã từng làm việc với ADO sẽ dễ dàng quen với các lớp kết nối của .NET, OleDbConnection và SqlConnection:



Đoạn mã sau đây mô tả cách để tạo, mở và đóng một kết nối đến cơ sở dữ liệu Northwind. Các ví dụ trong chương này chúng ta dùng cơ sở dữ liệu Northwind, được cài đặt chung với các ví dụ của .NET Framework SDK:

```
using System.Data.SqlClient;

string source = "server=(local);" +
    "uid=QSUuser;pwd=QSPassword;" +
    "database=Northwind";

SqlConnection conn = new SqlConnection(source);

conn.Open();

// Do something useful

conn.Close();
```

Chuỗi kết nối sẽ trở nên thân thiện nếu bạn đã từng dùng ADO hay OLE DB trước đây - thật vậy, bạn có thể cắt và dán từ mã cũ của bạn, nếu bạn dùng OleDb provider.

Trong ví dụ chuỗi kết nối này, các tham số được dùng như sau (các tham số cách nhau bởi dấu chấm phẩy trong chuỗi kết nối).

- `server=(local)\\NetSDK` - Nó biểu diễn database server được kết nối. SQL Server cho phép một số các tiến trình database server processes khác nhau chạy trên cùng một máy, vì vậy ở đây chúng ta thực hiện kết nối với tiến trình NetSDK trên máy cục bộ.
- `uid=QSUUser` - Tham số này mô tả người dùng cơ sở dữ liệu. Bạn cũng có thể sử dụng User ID.
- `pwd=QSPassword` - và đây là password cho người dùng đó. .NET SDK là một bộ các cơ sở dữ liệu giống nhau, và user/password này được liên kết và được thêm vào trong quá trình cài đặt các ví dụ .NET. Bạn cũng có thể dùng Password.
- `database=Northwind` - Cái này mô tả loại dữ liệu để kết nối - mỗi tiến trình SQL Server có thể đưa ra một vài loại dữ liệu khác nhau.

Ví trên mở một kết nối cơ sở dữ liệu dùng chuỗi kết nối đã được định nghĩa, sau đó đóng kết nối lại. Khi kết nối đã được mở, bạn có thể phát các lệnh để thao tác trên cơ sở dữ liệu, và khi hoàn tất, kết nối có thể được đóng lại.

SQL Server có một chế độ bảo mật khác - nó có thể dùng chế độ bảo mật của Windows, vì thế các khả năng truy cập của Windows có thể truyền cho SQL Server. Với lựa chọn này bạn có thể bỏ đi các vị trí uid và pwd trong chuỗi kết nối, và thêm vào `Integrated Security=SSPI`.

Trong lúc download mã nguồn sẵn có cho chương này, bạn cần tìm file `Login.cs` nó đơn giản hóa các ví dụ trong chương này. Nó được kết nối với tất cả các mã ví dụ, và bao gồm thông tin kết nối cơ sở dữ liệu dùng cho các ví dụ; sau đó bạn có thể cung cấp tên server, user, and password một cách thích hợp. Nếu mặc định dùng Windows integrated security; bạn cần thay đổi username và password cho phù hợp.

Bây giờ chúng ta đã biết cách mở các kết nối, trước khi chuyển qua vấn đề khác chúng ta cần xem xét một vài thực hành tốt có liên quan đến các kết nối.

3.2.1 Sử dụng Connection

Một cách tổng quát, khi sử dụng các tài nguyên "hiếm" trong .NET, chẳng hạn như các kết nối cơ sở dữ liệu, các cửa sổ, hoặc các đối tượng đồ họa, tốt hơn hết bạn nên đảm bảo rằng các tài nguyên này luôn phải được đóng lại sau khi đã sử dụng xong. Dù vậy các nhà thiết kế của .NET có thể làm điều này nhờ trình thu gom rác, nó luôn làm sau bộ nhớ sau một khoảng thời gian nào đó, tuy nhiên nó nên được giải phóng càng sớm càng tốt.

Rõ ràng là khi viết mã truy xuất một cơ sở dữ liệu, việc giữ một kết nối càng ít thời gian càng tốt để không làm ảnh hưởng đến các phần khác. Trong nhiều tình huống tiêu cực, nếu không đóng một kết nối có thể khoá không cho các người dùng khác truy nhập vào các bảng dữ liệu đó, một tác hại to lớn đối với khả năng thực thi của ứng dụng. Việc đóng một kết nối cơ sở dữ liệu có thể coi là bắt buộc, vì thế ứng dụng này chỉ ra cách cấu trúc mã của bạn để giảm thiểu các rủi ro cho một mã nguồn mở.

Có hai cách để đảm bảo rằng các kết nối cơ sở dữ liệu được giải phóng sau khi dùng.

3.2.1.1 Tùy chọn một - *try/catch/finally*

Tùy chọn thứ nhất để đảm bảo rằng các tài nguyên được dọn sạch là sử dụng các khối lệnh `try...catch...finally`, và đảm bảo rằng bạn đã đóng các kết nối trong khối lệnh `finally`. Đây là một ví dụ nhỏ:

```
try
{
    // Open the connection
    conn.Open();

    // Do something useful
}
catch ( Exception ex )
```

```
{  
    // Do something about the exception  
}  
finally  
{  
    // Ensure that the connection is freed  
    conn.Close ( ) ;  
}
```

Với khối kết nối bạn có thể giải phóng bất kì tài nguyên nào mà bạn đã dùng. Vấn đề duy nhất trong phương thức này là bạn phải bảo đảm rằng bạn có đóng các kết nối - rất là dễ quên việc thêm vào khối finally, vì vậy một phong cách lập trình tốt rất quan trọng.

Ngoài ra, bạn có thể mở một số tài nguyên (chẳng hạn hai kết nối cơ sở dữ liệu và một file) trong một phương thức, vì vậy đôi khi các khối try...catch...finally trở nên khó đọc. Có một cách khác để đảm bảo rằng các tài nguyên được dọn dẹp - sử dụng câu lệnh.

3.2.1.2 Tùy chọn hai - Sử dụng khối câu lệnh

Trong lúc phát triển C#, phương thức .NET's dọn dẹp các đối tượng khi chúng không còn được tham chiếu nữa sử dụng các hủy bất định trở thành một vấn đề nóng hổi. Trong C++, ngay khi một đối tượng rời khỏi tầm vực, khối hủy tử của nó sẽ tự động được gọi. Nó là một điều rất mới cho các nhà thiết kế các lớp sử dụng tài nguyên, khi một hủy tử được sử dụng để đóng các tài nguyên nếu các người dùng quên làm điều đó. Một hủy tử C++ được gọi bất kì khi nào một đối tượng vượt quá tầm vực của nó - vì vậy khi một ngoại lệ được phát ra mà không được chặn, tất cả các hủy tử cần phải được gọi.

Với C# và các ngôn ngữ có quản khác, tất cả đều tự động, các khối hủy tử định trước được thay thế bởi trình thu gom rác, cái được dùng để tháo các tài nguyên tại một thời điểm trong tương lai. Chúng mang tính bất định, nghĩa là bạn sẽ không biết trước được khi nào thì việc đó sẽ xảy ra. Nếu quên không đóng một kết nối cơ sở dữ liệu có thể

là nguyên nhân gây ra lỗi khi chạy trong .NET. Mã sau đây sẽ giải thích cách để sử dụng giao diện IDisposable (đã được bàn kỹ trong chương 2) để giải phóng tài nguyên khi thoát khỏi khối using .

```
string source = "server=(local)\\NetSDK;" +  
    "uid=QSUuser;pwd=QSPassword;" +  
    "database=Northwind";  
using ( SqlConnection conn = new SqlConnection ( source ) )  
{  
    // Open the connection  
    conn.Open ( ) ;  
    // Do something useful  
}
```

Mệnh đề using đã được giới thiệu trong chương 2. Đối tượng trong mệnh đề using phải thực thi giao diện IDisposable, nếu không một sẽ tạo ra một lỗi biên dịch. Phương thức Dispose() sẽ tự động được gọi trong khi thoát khỏi khối using.

Khi xem mã IL của phương thức Dispose() của SqlConnection (và OleDbConnection), cả hai đều kiểm tra trạng thái của đối tượng kết nối, và nếu nó đang mở phương thức Close() sẽ được gọi.

Khi lập trình bạn nên dùng cả hai tùy chọn trên.Ở nhưng chỗ bạn cần các tài nguyên tốt nhất là sử dụng mệnh đề using(), dù vậy bạn cũng có thể sử dụng câu lệnh Close(), nếu quên không sử dụng thì khối lệnh using sẽ đóng lại giúp bạn. Không gì có thể thay thế được một bấy ngoại lệ tốt, vì thế tốt nhất bạn dùng trộn lẫn hai phương thức như ví dụ sau:

```
try
```

```
{
```

```
using (SqlConnection conn = new SqlConnection ( source ))
{
    // Open the connection
    conn.Open ( ) ;

    // Do something useful

    // Close it myself
    conn.Close ( ) ;
}
}
catch (Exception e)
{
    // Do something with the exception here...
}
```

Ở đây tôi đã gọi tường minh phương thức Close() mặc dù điều đó là không bắt buộc vì khối lệnh using đã làm điều đó thay cho bạn; tuy nhiên, bạn luôn chắc rằng bất kỳ tài nguyên nào cũng được giải phóng sớm nhất có thể - bạn có thể có nhiều mã trong khối lệnh mã không khoá tài nguyên.

Thêm vào đó, nếu một ngoại lệ xảy ra bên trong khối using, thì phương thức IDisposable.Dispose sẽ được gọi để bảo đảm rằng tài nguyên được giải phóng, điều này đảm bảo rằng kết nối cơ sở dữ liệu luôn luôn được đóng lại. Điều này làm cho mã dễ đọc và luôn đảm bảo rằng kết nối luôn được đóng khi một ngoại lệ xảy ra.

Cuối cùng, nếu bạn viết các lớp bao bọc một tài nguyên có lẽ luôn thực hiện giao diện `IDisposable` để đóng tài nguyên. Bằng cách dùng câu lệnh `using()` nó luôn đảm bảo rằng tài nguyên đó sẽ được dọn dẹp.

3.2.2 Transaction

Thường khi có nhiều hơn một cập nhật dữ cơ sở dữ liệu thì các thực thi này được thực hiện bên trong tầm vực của một transaction. Một transaction trong ADO.NET được khởi tạo bằng một lời gọi đến các phương thức `BeginTransaction()` trên đối tượng kết nối cơ sở dữ liệu. Những phương thức này trả về một đối tượng có thể thực thi giao diện `IDbTransaction`, được định nghĩa trong `System.Data`.

Chuỗi mã lệnh dưới đây khởi tạo một transaction trên một kết nối SQL Server:

```
string source = "server=(local)\\NetSDK;" +  
    "uid=QSUuser;pwd=QSPassword;" +  
    "database=Northwind";  
  
SqlConnection conn = new SqlConnection(source);  
conn.Open();  
  
SqlTransaction tx = conn.BeginTransaction();  
  
// Execute some commands, then commit the transaction  
tx.Commit();  
conn.Close();
```

Khi bạn khởi tạo một transaction, bạn có thể chọn bậc tự do cho các lệnh thực thi trong transaction đó. Bậc này chỉ rõ sự tự do của transaction này với các transaction khác xảy ra trên cơ sở dữ liệu. Các hệ cơ sở dữ liệu có thể hỗ trợ bốn tùy chọn sau đây:

Isolation Level	Description
ReadCommitted	Mặc định cho. Bậc này đảm bảo rằng dữ liệu đã ghi bởi transaction sẽ chỉ có thể truy cập được bởi một transaction khác sau khi nó hoàn tất công việc của mình.
ReadUncommitted	Tùy chọn này cho phép transaction của bạn có thể đọc dữ liệu trong cơ sở dữ liệu, dù cho dữ liệu đó đang được một transaction khác sử dụng. Ví dụ như, nếu hai người dùng truy cập cùng lúc vào một cơ sở dữ liệu, và người thứ nhất chèn một vài dữ liệu trong transaction của họ (đó là một Commit hoặc Rollback), và người thứ hai với tùy chọn bậc tự do là ReadUncommitted có thể đọc dữ liệu.
RepeatableRead	<p>Bậc này là một mở rộng của ReadCommitted, nó bảo đảm rằng nếu một lệnh tương tự được phát ra trong transaction, ensures that if the same statement is issued within the transaction, regardless of other potential updates made to the database, the same data will always be returned. This level does require extra locks to be held on the data, which could adversely affect performance.</p> <p>This level guarantees that, for each row in the initial query, no changes can be made to that data. It does however permit "phantom" rows to show up - these are completely new rows that another transaction may have inserted while your transaction is running.</p>
Serializable	This is the most "exclusive" transaction level, which in effect serializes access to data within the database. With this isolation level, phantom rows can never show up, so a SQL statement

Isolation Level	Description
	issued within a serializable transaction will always retrieve the same data. The negative performance impact of a Serializable transaction should not be underestimated - if you don't absolutely need to use this level of isolation, it is advisable to stay away from it.

3.3 Commands

Chúng ta lại nói lại về commands. Một command là một kiểu đơn giản, một chuỗi lệnh SQL được dùng để truy xuất dữ liệu. Một command có thể là một stored procedure, hoặc là tên của một bảng sẽ trả về:

```
string source = "server=(local)\\NetSDK;" +  
    "uid=QSError;pwd=QSPassword;" +  
    "database=Northwind";  
  
string select = "SELECT ContactName,CompanyName FROM Customers";  
  
SqlConnection conn = new SqlConnection(source);  
  
conn.Open();  
  
SqlCommand cmd = new SqlCommand(select, conn);
```

Các mệnh đề SqlCommand và OleDbCommand thường được gọi là CommandType, chúng được dùng để định nghĩa các mệnh đề SQL, một stored procedure, hoặc một câu lệnh SQL. Sau đây là một bảng liệt kê đơn giản về CommandType:

CommandType	Example
-------------	---------

CommandType	Example
Text (default)	<pre>String select = "SELECT ContactName FROM Customers"; SqlCommand cmd = new SqlCommand(select , conn);</pre>
StoredProcedure	<pre>SqlCommand cmd = new SqlCommand("CustOrderHist", conn); cmd.CommandType = CommandType.StoredProcedure; cmd.Parameters.Add("@CustomerID", "QUICK");</pre>
TableDirect	<pre>OleDbCommand cmd = new OleDbCommand("Categories", conn); cmd.CommandType = CommandType.TableDirect;</pre>

Khi thực thi một stored procedure, cần truyền các tham số cho procedure. Ví dụ trên cài đặt trực tiếp tham số @CustomerID, dù vậy có nhiều cách để cài giá trị tham số, chúng ta sẽ bàn kỹ trong phần sau của chương này.

chú ý: Kiểu TableDirect command không chỉ đúng cho OleDb provider – có một ngoại lệ xảy ra khi bạn cố dùng command này trong Sql provider.

3.3.1 Executing Commands

Bạn đã định nghĩa các command, và bạn muốn thực thi chúng. Có một số cách để phát ra các statement, dựa vào kết quả mà bạn muốn command đó muốn trả về. Các mệnh đề SqlCommand và OleDbCommand cung cấp các phương thức thực thi sau:

- ExecuteNonQuery() – Thực thi các command không trả về kết quả gì cả
- ExecuteReader() – Thực thi các command và trả về kiểu IDataReader
- ExecuteScalar() – Thực thi các command và trả về một giá trị đơn

Lớp SqlCommand cung cấp thêm một số phương thức sau

- ExecuteXmlReader() – Thực thi các command trả về một đối tượng XmlReader, các đối tượng được dùng để xem xét các XML được trả về từ cơ sở dữ liệu.

3.3.1.1 ExecuteNonQuery()

Phương thức này thường được dùng cho các câu lệnh UPDATE, INSERT, hoặc DELETE, để trả về số các mẫu tin bị tác động. Phương thức này có thể trả về các kết quả thông qua các tham số được truyền vào stored procedure.

```
using System;

using System.Data.SqlClient;

public class ExecuteNonQueryExample
{
    public static void Main(string[] args)
    {
        string source = "server=(local)\\NetSDK;" +
            "uid=QSUuser;pwd=QSPassword;" +
            "database=Northwind";

        string select = "UPDATE Customers " +
            "SET ContactName = 'Bob' " +
            "WHERE ContactName = 'Bill'";

        SqlConnection conn = new SqlConnection(source);

        conn.Open();

        SqlCommand cmd = new SqlCommand(select, conn);

        int rowsReturned = cmd.ExecuteNonQuery();
    }
}
```

```
Console.WriteLine("{0} rows returned.", rowsReturned);

conn.Close();

}

}
```

ExecuteNonQuery() trả về một số kiểu int cho biết số dòng bị tác động command.

3.3.1.2 *ExecuteReader()*

Phương thức này thực hiện các lệnh trả về một đối tượng SqlDataReader hoặc OleDbDataReader. Đối tượng này có thể dùng để tạo ra các mẫu tin như mã sau đây:

```
using System;

using System.Data.SqlClient;

public class ExecuteReaderExample
{
    public static void Main(string[] args)
    {
        string source = "server=(local)\\NetSDK;" +
            "uid=QSUser;pwd=QSPassword;" +
            "database=Northwind";

        string select = "SELECT ContactName,CompanyName FROM Customers";

        SqlConnection conn = new SqlConnection(source);

        conn.Open();

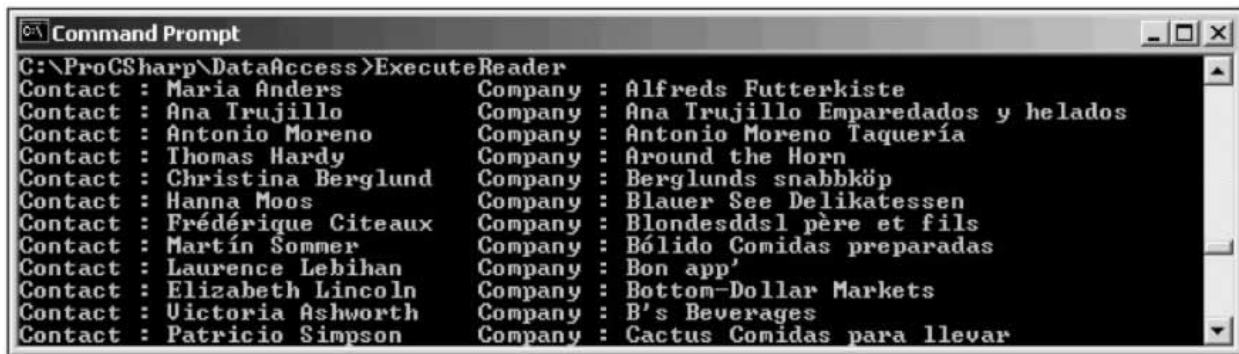
        SqlCommand cmd = new SqlCommand(select, conn);

        SqlDataReader reader = cmd.ExecuteReader();

        while(reader.Read())
```



```
{  
  
    Console.WriteLine("Contact : {0,-20} Company : {1}" ,  
  
        reader[0] , reader[1]);  
  
}  
  
}  
  
}
```



```
C:\ProCSharp\DataAccess>ExecuteReader  
Contact : Maria Anders          Company : Alfreds Futterkiste  
Contact : Ana Trujillo          Company : Ana Trujillo Emparedados y helados  
Contact : Antonio Moreno        Company : Antonio Moreno Taqueria  
Contact : Thomas Hardy          Company : Around the Horn  
Contact : Christina Berglund     Company : Berglunds snabbköp  
Contact : Hanna Moos            Company : Blauer See Delikatessen  
Contact : Frédérique Citeaux     Company : Blondesddsl père et fils  
Contact : Martín Sommer         Company : Bólido Comidas preparadas  
Contact : Laurence Leblan        Company : Bon app'  
Contact : Elizabeth Lincoln      Company : Bottom-Dollar Markets  
Contact : Victoria Ashworth      Company : B's Beverages  
Contact : Patricio Simpson       Company : Cactus Comidas para llevar
```

Các đối tượng SqlDataReader và OleDbDataReader sẽ được trình bày trong chương sau.

3.3.1.3 ExecuteScalar()

Trong nhiều trường hợp một câu lệnh SQL cần phải trả về một kết quả đơn, chẳng hạn như số các record của một bảng, hoặc ngày giờ hiện tại của server. Phương thức ExecuteScalar có thể dùng cho những trường hợp này:

```
using System;  
  
using System.Data.SqlClient;  
  
public class ExecuteScalarExample  
{  
  
    public static void Main(string[] args)  
  
    {  
  
        string source = "server=(local)\\NetSDK;" +
```

```
"uid=QUser;pwd=QSPassword;" +  
"database=Northwind";  
  
string select = "SELECT COUNT(*) FROM Customers";  
  
SqlConnection conn = new SqlConnection(source);  
  
conn.Open();  
  
SqlCommand cmd = new SqlCommand(select, conn);  
  
object o = cmd.ExecuteScalar();  
  
Console.WriteLine ( o ) ;  
  
}  
  
}
```

Phương thức trả về một đối tượng, Bạn có thể chuyển sang kiểu thích hợp.

3.3.1.4 *ExecuteXmlReader() (SqlClient Provider Only)*

Giống như tên đã gọi, nó có thể thực thi command và trả về một đối tượng XmlReader. SQL Server cho phép câu lệnh SQL SELECT dùng cho kiểu FOR XML. Mệnh đề này có thể có ba kiểu tùy chọn sau:

- FOR XML AUTO – tạo một cây cơ sở cho các bảng trong mệnh đề FROM
- FOR XML RAW – trả về một bộ các mẫu tin ánh xạ định các nhân tố, với các cột được ánh xạ đến các thuộc tính
- FOR XML EXPLICIT – bạn cần phải chỉ định hình dạng của cây XML trả về

Professional SQL Server 2000 XML (Wrox Press, ISBN 1-861005-46-6) diễn tả đầy đủ các thuộc tính này:

```
using System;  
  
using System.Data.SqlClient;
```

```
using System.Xml;

public class ExecuteXmlReaderExample
{
    public static void Main(string[] args)
    {
        string source = "server=(local)\\NetSDK;" +
            "uid=QSUser;pwd=QSPassword;" +
            "database=Northwind";

        string select = "SELECT ContactName,CompanyName " +
            "FROM Customers FOR XML AUTO";

        SqlConnection conn = new SqlConnection(source);
        conn.Open();

        SqlCommand cmd = new SqlCommand(select, conn);
        XmlReader xr = cmd.ExecuteXmlReader();

        xr.Read();

        string s;

        do
        {
            s = xr.ReadOuterXml();

            if (s!="")

                Console.WriteLine(s);

        } while (s!= "");

        conn.Close();
    }
}
```

```
}  
  
}
```

Chú ý rằng chúng ta có thể nhập không gian tên System.Xml namespace cho các kiểu trả về XML. Không gian này được dùng cho những khả năng của XML trong .NET Framework trong tương lai được trình bày kỹ trong chương 11.

Ở đây chúng bao gồm các mệnh đề FOR XML AUTO trong mệnh đề SQL, sau đó gọi phương thức ExecuteXmlReader(). Sau đây là kết quả của các mã lệnh trên:



```
C:\WINNT\system32\CMD.EXE  
<Customers ContactName="Maria Anders" CompanyName="Alfreds Futterkiste"/>  
<Customers ContactName="Ana Trujillo" CompanyName="Ana Trujillo Emparedados y h<br><Customers ContactName="Antonio Moreno" CompanyName="Antonio Moreno Taquería"/>  
<Customers ContactName="Thomas Hardy" CompanyName="Around the Horn"/>  
<Customers ContactName="Christina Berglund" CompanyName="Berglunds snabbköp"/>  
<Customers ContactName="Hanna Moos" CompanyName="Blauer See Delikatessen"/>  
<Customers ContactName="Frédérique Citeaux" CompanyName="Blondesddsl père et fi<br><Customers ContactName="Martin Sommer" CompanyName="Bólido Comidas preparadas"/>  
<Customers ContactName="Laurence Lebihan" CompanyName="Bon app&apos;/>  
<Customers ContactName="Elizabeth Lincoln" CompanyName="Acme, Inc."/>  
<Customers ContactName="Victoria Ashworth" CompanyName="B&apos;s Beverages"/>  
<Customers ContactName="Patricio Simpson" CompanyName="Cactus Comidas para llev<br><Customers ContactName="Francisco Chang" CompanyName="Centro comercial Moctezum<br><Customers ContactName="Yang Wang" CompanyName="Chop-suey Chinese"/>
```

Trong mệnh đề SQL, chúng ta có thể chỉ định, để các thành phần của kiểu Customers được hiển thị trong phần kết xuất. Để làm điều đó ta phải thêm các thuộc tính cho mỗi một cột trong cơ sở dữ liệu. Điều này sẽ tạo ra một sự phân mảnh trong việc chọn các mẫu tin từ cơ sở dữ liệu.

3.3.2 Gọi các Stored Procedure

Việc gọi một stored procedure với một đối tượng command đơn giản là định nghĩa tên của stored procedure cần dùng, thêm vào các tham số của procedure đó, thực thi command với một trong các phương thức đã giới thiệu ở phần trên.

Để dễ dàng cho việc lấy ví dụ trong phần này, Tôi đã định nghĩa một bộ các stored procedures dùng để chèn, cập nhật, và xoá các mẫu tin từ bảng Region trong cơ sở dữ

liệu Northwind. Tôi đã chọn bảng này vì nó đủ nhỏ để có thể áp dụng các ví dụ cho mỗi kiểu của storeprocedure.

3.3.2.1 Gọi một Stored Procedure không trả lại kết quả

Ví dụ này sẽ mô tả cách gọi một stored procedure không trả lại kết quả. Có hai procedure được định nghĩa dưới đây, một dùng cho việc cập nhật các mẫu Region sẵn có, và một dùng để xóa các mẫu tin trong Region.

Record Update

Cập nhật một mẫu Region là một công việc khá đơn giản, chỉ thay đổi một trường duy nhất (vì không thể cập nhật khóa chính). Bạn có thể gõ trực tiếp các ví dụ này trong SQL Server Query Analyzer, để cài đặt các stored procedure dùng cho phần này:

```
CREATE PROCEDURE RegionUpdate (@RegionID INTEGER,  
                                @RegionDescription NCHAR(50)) AS  
  
SET NOCOUNT OFF  
  
UPDATE Region  
  
SET RegionDescription = @RegionDescription  
  
WHERE RegionID = @RegionID  
  
GO
```

Một lệnh cập nhật trong một bảng thực tế hơn cần phải chọn lại và trả về trạng thái của các mẫu được cập nhật. Stored procedure này cần nhập vào hai tham số (@RegionID và @RegionDescription, và phát ra một câu lệnh UPDATE để thao tác trên cơ sở dữ liệu.

Để chạy stored procedure này trong mã .NET, bạn cần phải định nghĩa một lệnh SQL và thực thi nó:

```
SqlCommand aCommand = new SqlCommand("RegionUpdate", conn);
```

```
aCommand.CommandType = CommandType.StoredProcedure;
aCommand.Parameters.Add(new SqlParameter("@RegionID",
                                         SqlDbType.Int,
                                         0,
                                         "RegionID"));
aCommand.Parameters.Add(new SqlParameter("@RegionDescription",
                                         SqlDbType.NChar,
                                         50,
                                         "RegionDescription"));
aCommand.UpdatedRowSource = UpdateRowSource.None;
```

Đoạn mã này tạo một đối tượng SqlCommand mới tên là aCommand, và định nghĩa nó là một stored procedure. Sau đó chúng ta thêm vào các tham số nhập, cũng như các tham số chứa giá trị mong muốn trả về từ stored procedure để biết được các giá trị trong các dòng UpdateRowSource được liệt kê, chúng ta sẽ bàn kỹ vấn đề ở các phần sau của chương này.

Một command được tạo ra, có thể được thực thi bởi việc phát ra các lệnh sau:

```
aCommand.Parameters[0].Value = 999;
aCommand.Parameters[1].Value = "South Western England";
aCommand.ExecuteNonQuery();
```

Ở đây chúng ta đang cài đặt giá trị cho các tham số, sau đó thực thi stored procedure.

Các Command parameters có thể được cài đặt bằng chỉ số như đã trình bày ở trên, hoặc dùng tên.

Record Deletion

Stored procedure tiếp theo dùng để xóa một mẫu tin trong bảng Region:

```
CREATE PROCEDURE RegionDelete (@RegionID INTEGER) AS  
  
SET NOCOUNT OFF  
  
DELETE FROM Region  
  
WHERE    RegionID = @RegionID  
  
GO
```

Procedure này chỉ yêu cầu khóa chính của mẫu tin. Mã sử dụng một đối tượng SqlCommand để gọi stored procedure này như sau:

```
SqlCommand aCommand = new SqlCommand("RegionDelete" , conn);  
aCommand.CommandType = CommandType.StoredProcedure;  
aCommand.Parameters.Add(new SqlParameter("@RegionID" , SqlDbType.Int , 0 ,  
                                         "RegionID"));  
aCommand.UpdatedRowSource = UpdateRowSource.None;
```

Lệnh này chỉ chấp nhận một tham số đơn để thực thi RegionDelete stored procedure; đây là ví dụ cho việc cài đặt tham số theo tên:

```
aCommand.Parameters["@RegionID"].Value= 999;  
aCommand.ExecuteNonQuery();
```

3.3.2.2 Gọi Stored Procedure có các tham số trả về

Cả hai ví dụ về stored procedures ở trên đều không có giá trị trả về. Nếu một stored procedure bao gồm các tham số trả về, sau đó những phương thức này cần được định nghĩa trong .NET client rằng chúng có thể lấy giá trị trả về từ procedure.

Ví dụ sau chỉ ra cách chèn một mẫu tin vào cơ sở dữ liệu, và trả về khóa chính của mẫu tin đó.

Record Insertion

Bảng Region chỉ chứa khóa chính (RegionID) và một trường diễn giải (RegionDescription). Để chèn một mẫu tin, cần phải cung cấp khóa chính, và sau đó một mẫu tin mới sẽ được chèn vào cơ sở dữ liệu. Tôi đã chọn cách tạo khóa chính đơn giản nhất trong ví dụ này bằng cách tạo ra một số mới trong stored procedure. Phương thức được dùng hết sức thô sơ, tôi sẽ bàn kỹ về cách tạo khóa chính ở phần sau của chương. Và đây là ví dụ thô sơ của chúng ta:

```
CREATE PROCEDURE RegionInsert(@RegionDescription NCHAR(50),
                              @RegionID INTEGER OUTPUT)AS
SET NOCOUNT OFF
SELECT @RegionID = MAX(RegionID)+ 1
FROM Region
INSERT INTO Region(RegionID, RegionDescription)
VALUES(@RegionID, @RegionDescription)
GO
```

Insert procedure này tạo ra một mẫu tin Region mới. Khóa chính được phát ra bởi chính cơ sở dữ liệu, giá trị này được trả về như một tham số của procedure (@RegionID). Đây là một ví dụ đơn giản, nhưng đối với các bảng phức tạp hơn, nó thường không sử dụng các tham số trả về mà thay vào đó nó chọn các dòng được cập nhật và trả nó về cho trình gọi.

```
SqlCommand aCommand = new SqlCommand("RegionInsert" , conn);
aCommand.CommandType = CommandType.StoredProcedure;
aCommand.Parameters.Add(new SqlParameter("@RegionDescription" ,
                                         SqlDbType.NChar ,
                                         50 ,
```



```
        "RegionDescription"));
aCommand.Parameters.Add(new SqlParameter("@RegionID" ,
        SqlDbType.Int,
        0 ,
        ParameterDirection.Output ,
        false ,
        0 ,
        0 ,
        "RegionID" ,
        DataRowVersion.Default ,
        null));
aCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;
```

Đây là phần định nghĩa phức tạp hơn cho các tham số. Tham số thứ hai, @RegionID, được định nghĩa để bao gồm các tham số trực tiếp của nó, trong ví dụ này nó là Output. Chúng ta sử dụng tập hợp UpdateRowSource để thêm cờ OutputParameters trên dòng cuối của mã, cờ này cho phép chúng ta trả dữ liệu từ stored procedure này vào các tham số. Cờ này được dùng chủ yếu cho việc gọi các stored procedure từ một DataTable (được giải thích trong chương sau).

Việc gọi stored procedure này giống như các ví dụ trước, ngoại trừ ở đây chúng ta cần đọc tham số xuất sau khi thực thi procedure:

```
aCommand.Parameters["@RegionDescription"].Value = "South West";
aCommand.ExecuteNonQuery();
int newRegionID = (int) aCommand.Parameters["@RegionID"].Value;
```

Sau khi thực thi lệnh, chúng ta đọc giá trị tham số @RegionID và ép nó vào một integer.

Có thể bạn sẽ hỏi phải làm gì nếu stored procedure mà bạn gọi trả về các tham số xuất và một tập các dòng. Trong trường hợp này, định nghĩa các tham số tương ứng, sau đó gọi phương thức ExecuteNonQuery(), cũng có thể gọi một trong những phương thức khác (chẳng hạn như ExecuteReader()) nó cho phép bạn lấy các mẫu tin trả về.

3.4 Truy cập nhanh cơ sở dữ liệu với Data Reader

Một data reader là cách đơn giản nhất và nhanh nhất để chọn một vài dữ liệu từ một nguồn cơ sở dữ liệu, nhưng cũng ít tính năng nhất. Bạn có thể truy xuất trực tiếp một đối tượng data reader – Một minh dụ được trả về từ một đối tượng SqlCommand hoặc OleDbCommand từ việc gọi một phương thức ExecuteReader() – có thể là một đối tượng SqlCommand, một đối tượng SqlDataReader, từ một đối tượng OleDbCommand là một OleDbDataReader.

Mã lệnh sau đây sẽ chứng minh cách chọn dữ liệu từ bản Customers của cơ sở dữ liệu Northwind. Ví dụ kết nối với cơ sở dữ liệu chọn một số các mẫu tin, duyệt qua các mẫu tin được chọn và xuất chúng ra màn hình console.

Ví dụ này có thể dùng cho OLE DB provider. Trong hầu hết các trường hợp các phương thức của SqlClient đều được ánh xạ một một vào các phương thức của đối OleDBClient.

Để thực thi lại các lệnh đối với một OLE DB data source, lớp OleDbCommand được sử dụng. Mã lệnh dưới đây là một ví dụ một câu lệnh SQL đơn giản và đọc các mẫu tin được trả về bởi đối tượng OleDbDataReader.

Chú ý hai câu lệnh using dưới đây được dùng trong lớp OleDb:

```
using System;  
using System.Data.OleDb;
```

Tất cả các trình cung cấp dữ liệu đều sẵn chứa bên trong các data DLL, vì vậy chỉ cần tham chiếu đến System.Data.dll assembly để dùng cho các lớp trong phần này:

```
public class DataReaderExample
{
    public static void Main(string[] args)
    {
        string source = "Provider=SQLOLEDB;" +
            "server=(local)\\NetSDK;" +
            "uid=QSUser;pwd=QSPassword;" +
            "database=northwind";

        string select = "SELECT ContactName,CompanyName FROM Customers";
        OleDbConnection conn = new OleDbConnection(source);
        conn.Open();
        OleDbCommand cmd = new OleDbCommand(select, conn);
        OleDbDataReader aReader = cmd.ExecuteReader();
        while(aReader.Read())
        {
            Console.WriteLine("{0}' from {1}",
                aReader.GetString(0), aReader.GetString(1));
        }
        aReader.Close();
        conn.Close();
    }
}
```

Mã nguồn trên đây bao gồm các đoạn lệnh quen thuộc đã được trình bày trong các chương trước. Để biên dịch ví dụ này, ta dùng các dòng lệnh sau:

```
csc /t:exe /debug+ DataReaderExample.cs /r:System.Data.dll
```

Mã sau đây từ ví dụ trên cho phép tạo một kết nối OLE DB .NET, dựa trên chuỗi kết nối:

```
OleDbConnection conn = new OleDbConnection(source);  
  
conn.Open();  
  
OleDbCommand cmd = new OleDbCommand(select, conn);
```

Dòng thứ ba tạo một đối tượng OleDbCommand mới, dựa vào câu lệnh SELECT, kết nối sẽ thực thi câu lệnh này. Nếu bạn tạo một command hợp lệ, bạn có thể thực thi chúng để trả về một minh dụ OleDbDataReader:

```
OleDbDataReader aReader = cmd.ExecuteReader();
```

Một OleDbDataReader chỉ là một con trỏ "connected" định trước. Mặt khác, bạn có thể chỉ duyệt qua các mẫu tin được trả về, kết nối hiện tại sẽ lưu giữ các mẫu tin đó cho đến khi data reader bị đóng lại.

Lớp OleDbDataReader không thể tạo minh dụ một cách trực tiếp – nó luôn được trả về thông qua việc gọi phương thức ExecuteReader() của lớp OleDbCommand. Nhưng bạn có thể mở một data reader, có một số cách khác nhau để truy cập dữ liệu trong reader.

Khi một đối tượng OleDbDataReader bị đóng lại (thông qua việc gọi phương thức Close(), hoặc một đợt thu dọn rác), kết nối bên dưới có thể bị đóng lại thông qua một lời gọi phương thức ExecuteReader(). Nếu bạn gọi ExecuteReader() và truyền CommandBehavior.CloseConnection, bạn có thể ép kết nối đóng lại khi đóng reader.

Lớp OleDbDataReader có một bộ các quyền truy xuất thông qua các mảng quen thuộc:

```
object o = aReader[0];  
  
object o = aReader["CategoryID"];
```

Ở đây CategoryID là trường đầu tiên trong câu lệnh SELECT của reader, cả hai dòng trên đều thực hiện công việc giống nhau tuy nhiên cách hai hơi chậm hơn cách một – Tôi đã viết một ứng dụng đơn giản để thực thi việc lặp lại quá trình truy cập cho hàng triệu lần một cột trong một mẫu tin reader, chỉ để lấy một vài mẫu. Tôi biết bạn hầu như không bao giờ đọc một cột giống nhau hàng triệu lần, nhưng có thể là một số lần, bạn nên viết mã để tối ưu quá trình đó.

Bạn có biết kết quả là thế nào không, việc truy cập một triệu lần bằng số thứ tự chỉ tốn có 0.09 giây, còn dùng chuỗi kí tự phải mất 0.63 giây. Lí do của sự chậm trễ này là vì khi dùng chuỗi kí tự ta phải dò trong schema để lấy ra số thứ tự của cột từ đó mới truy xuất được cơ sở dữ liệu. Nếu bạn biết được các thông tin này bạn có thể viết mã truy xuất dữ liệu tốt hơn. Vì vậy việc dùng chỉ số cột là cách dùng tốt nhất.

Hơn thế nữa, OleDbDataReader có một bộ các phương thức type-safe có thể dùng để đọc các cột. Những phương thức này có thể đọc hầu hết các loại dữ liệu như GetInt32, GetFloat, GetGuid, vân vân.

Thí nghiệm của tôi khi dùng GetInt32 là 0.06 giây. Nhanh hơn việc dùng chỉ số cột, vì khi đó bạn phải thực hiện thao tác ép kiểu để đưa kiểu trả về kiểu integer. Vì vậy nếu biết trước schema bạn nên dùng các chỉ số thay vì tên.

Chắc bạn cũng biết nên giữ sự cân bằng giữa tính dễ bảo trì và tốc độ. Nếu bạn muốn dùng các chỉ mục, bạn nên định nghĩa các hằng số cho mỗi cột mà bạn sẽ truy cập.

Ví dụ dưới đây giống như ví dụ ở trên nhưng thay vì sử dụng OLE DB provider thì ở đây sử dụng SQL provider. Nhưng phần thay đổi của mã so với ví dụ trên được tô đậm.

```
using System;

using System.Data.SqlClient;

public class DataReaderSql

{

    public static int Main(string[] args)
```

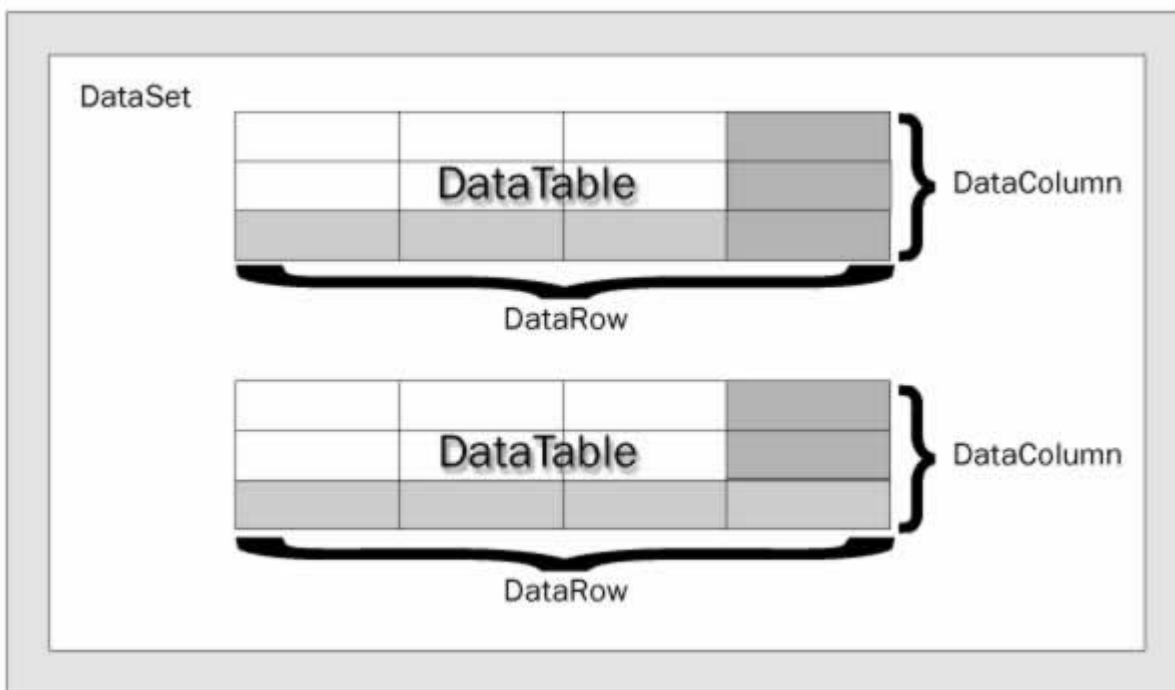
```
{  
  
    string source = "server=(local)\\NetSDK;" +  
        "uid=QSUuser;pwd=QSPassword;" +  
        "database=northwind";  
  
    string select = "SELECT ContactName,CompanyName FROM Customers";  
  
    SqlConnection conn = new SqlConnection(source);  
  
    conn.Open();  
  
    SqlCommand cmd = new SqlCommand(select , conn);  
  
    SqlDataReader aReader = cmd.ExecuteReader();  
  
    while(aReader.Read())  
    {  
        Console.WriteLine("{0}' from {1}" , aReader.GetString(0) ,  
            aReader.GetString(1));  
    }  
  
    aReader.Close();  
  
    conn.Close();  
  
    return 0;  
}  
}
```

Tôi đã chạy thử nghiệm của mình trên SQL provider, và kết quả là 0.13 giây cho một triệu lần truy cập bằng chỉ mục, và 0.65 giây nếu dùng chuỗi. Bạn có mong rằng SQL Server provider nhanh hơn so với OleDb, tôi đã test thử nghiệm của mình trong phiên bản .NET.

3.5 Managing Data và Relationships: The DataSet

Lớp DataSet được thiết kế như là một thùng chứa các dữ liệu không kết nối. Nó không có khái niệm về các kết nối dữ liệu. Thật vậy, dữ liệu được giữ trong một DataSet không quan tâm đến nguồn cơ sở dữ liệu – nó có thể chỉ là những mẫu tin chứa trong một file CSV, hoặc là những đầu đọc từ một thiết bị đo lường.

Một DataSet bao gồm một tập các bảng dữ liệu, mỗi bảng là một tập các cột dữ liệu và dòng dữ liệu. Thêm vào đó là các định nghĩa dữ liệu, bạn có thể định nghĩa các *link* giữa các DataSet. Mỗi quan hệ phổ biến giữa các DataSet là parent-child relationship. Một mẫu tin trong một bảng (gọi là Order) có thể liên kết với nhiều mẫu tin trong bảng khác (Bảng Order_Details). Quan hệ này có thể được định nghĩa và đánh dấu trong DataSet.



Phần dưới đây giải thích các lớp được dùng trong một DataSet.

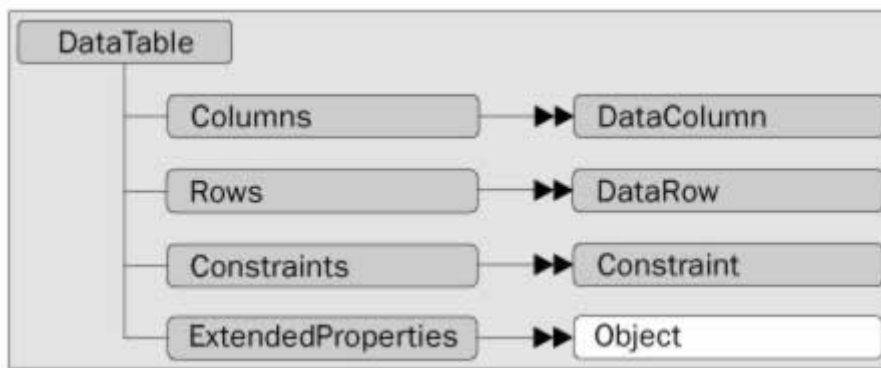
3.5.1 DataTable

Một data table rất giống một bảng cơ sở dữ liệu vật lý – nó bao gồm một bộ các cột với các thuộc tính riêng, và có thể không chứa hoặc chứa nhiều dòng dữ liệu. Một data

table có thể định nghĩa một khóa chính, bao gồm một hoặc nhiều cột, và cũng có thể chứa các ràng buộc của các cột. Tất cả các thông tin đó được thể hiện trong **schema**.

Có nhiều các để định nghĩa một schema cho một bảng dữ liệu riêng. Chúng sẽ được thảo luận ngay sau phần giới thiệu về cột dữ liệu và dòng dữ liệu.

Sơ đồ dưới đây chỉ ra một vài đối tượng có thể truy cập thông qua một bảng dữ liệu:



Một đối tượng DataTable (cũng như một DataColumn) có thể có một số các mở rộng riêng liên quan đến thuộc tính của nó. Tập hợp này có thể nằm trong thông tin user-defined gắn liền với đối tượng. Ví dụ, một cột có thể đưa ra một mặt nạ nhập liệu dùng để giới hạn các giá trị hợp lệ cho cột đó – một ví dụ về số phúc lợi xã hội Mỹ. Các thuộc tính mở rộng đặc biệt quan trọng khi dữ liệu được cấu trúc ở một tầng giữa và trả về cho client trong một số tiến trình. Bạn có thể lưu một chuẩn hợp lệ (như min và max) cho các số của các cột.

Khi một bảng dữ liệu được tạo ra, có thể do việc chọn dữ liệu từ một cơ sở dữ liệu, đọc dữ liệu từ một file, hoặc truy xuất thủ công trong mã, tập hợp Rows được dùng để chứa giá trị trả về.

Tập hợp Columns chứa các thể hiện DataColumn có thể được thêm vào bảng này. Những định nghĩa schema của dữ liệu, ví dụ như kiểu dữ liệu, tính khả rỗng, giá trị mặc định, v.v... Tập Constraints có thể được tạo ra bởi các ràng buộc khóa chính hoặc tính độc nhất.

Thông tin về sơ đồ của một bảng dữ liệu có thể được sử dụng trong việc biểu diễn của một bảng dữ liệu bằng DataGrid (chúng ta sẽ bàn về vấn đề này trong chương sau). Điều khiển DataGrid sử dụng các thuộc tính như kiểu dữ liệu của cột để quyết định điều khiển gì dùng cho cột đó. Một trường bit trong cơ sở dữ liệu có thể được biểu diễn như một checkbox trong DataGrid. Nếu một cột được định nghĩa trong cơ sở sơ đồ dữ liệu như là một NOT NULL, lựa chọn này được lưu trữ trong DataColumn vì vậy nó sẽ được kiểm tra khi người dùng cố gắng di chuyển khỏi một dòng.

3.5.2 DataColumn

Một đối tượng DataColumn định nghĩa các thuộc tính của một cột trong DataTable, chẳng hạn như kiểu dữ liệu của cột đó, chẳng hạn cột là chỉ đọc, và các sự kiện khác. Một cột có thể được tạo bằng mã, hoặc có thể được tạo tự động trong thời gian chạy.

Khi tạo một cột, tốt hơn hết là nên đặt cho nó một cái tên; nếu không thời gian chạy sẽ tự động sinh cho bạn một cái tên theo định dạng Columnn, n là mô số tự động tăng.

Kiểu dữ liệu của một cột có thể cài đặt bằng cách cung cấp trong cấu trúc của nó, hoặc bằng cách cài đặt thuộc tính DataType. Một khi bạn đã load dữ liệu vào một bảng dữ liệu bạn không thể sửa lại kiểu dữ liệu của một cột – nếu không bạn sẽ nhận một ngoại lệ.

Các cột dữ liệu có thể được tạo để giữ các kiểu dữ liệu của .NET Framework sau:

Boolean	Decimal	Int64	TimeSpan
Byte	Double	Sbyte	UInt16
Char	Int16	Single	UInt32

DateTime	Int32	String	UInt64
----------	-------	--------	--------

Một khi đã được tạo, bước tiếp theo là cài các thuộc tính khác cho đối tượng DataColumn, chẳng hạn như tính khả rỗng nullability, giá trị mặc định. Đoạn mã sau chỉ ra một số các tùy chọn được cài đặt trong một DataColumn:

```

DataColumn customerID = new DataColumn("CustomerID" , typeof(int));
customerID.AllowDBNull = false;
customerID.ReadOnly = false;
customerID.AutoIncrement = true;
customerID.AutoIncrementSeed = 1000;

DataColumn name = new DataColumn("Name" , typeof(string));
name.AllowDBNull = false;
name.Unique = true;

```

Các thuộc tính sau có thể được cài đặt trong một DataColumn:

Property	Description
AllowDBNull	Nếu là true, cho phép cột có thể chấp nhận DBNull.
AutoIncrement	Cho biết rằng dữ liệu của cột này là một số tự động tăng.
AutoIncrementSeed	Giá trị khởi đầu cho một cột AutoIncrement.
AutoIncrementStep	Cho biết bước tăng giữa các giá trị tự động, mặc định là 1.
Caption	Có thể dùng cho việc biểu diễn tên của cột trên màn hình.

Property	Description
ColumnMapping	Cho biết cách một cột ánh xạ sang XML khi một DataSet được lưu bằng cách gọi phương thức DataSet.WriteXml.
ColumnName	Tên của cột. Nó tự động tạo ra trong thời gian chạy nếu không được cài đặt trong cấu trúc.
DataType	Kiểu giá trị của cột.
DefaultValue	Dùng để định nghĩa giá trị mặc định cho một cột
Expression	Thuộc tính này định nghĩa một biểu thức dùng cho việc tính toán trên cột này

3.5.3 DataRow

Lớp này cấu thành các phần khác của lớp DataTable. Các cột trong một data table được định nghĩa trong các thuộc tính của lớp DataColumn. Dữ liệu của bảng thật sự có thể truy xuất được nhờ vào đối tượng DataRow. Ví dụ sau trình bày cách truy cập các dòng trong một bảng dữ liệu. Trước tiên là các thông tin về kết nối:

```
string source = "server=(local)\\NetSDK;" +  
    "uid=QSUuser;pwd=QSPassword;" +  
    "database=northwind";  
  
string select = "SELECT ContactName,CompanyName FROM Customers";  
  
SqlConnection conn = new SqlConnection(source);
```

Mã sau đây giới thiệu lớp SqlDataAdapter, được dùng để điền dữ liệu cho một DataSet. SqlDataAdapter sẽ phát ra các SQL, và điền vào một bảng Customers trong DataSet. Chúng ta sẽ bàn về lớp data adapter trong phần *Populating a DataSet* dưới đây.

```
SqlDataAdapter da = new SqlDataAdapter(select, conn);  
  
DataSet ds = new DataSet();  
  
da.Fill(ds, "Customers");
```

Trong mã dưới đây, bạn chú ý cách dùng chỉ mục của DataRow để truy xuất giá trị trong dòng đó. Giá trị của một cột có thể trả về bằng cách dùng một trong những chỉ mục được cài đặt. Chúng cho phép bạn trả về một giá trị cho biết số, tên, hoặc DataColumn:

```
foreach(DataRow row in ds.Tables["Customers"].Rows)  
  
    Console.WriteLine("{0}' from {1}", row[0], row[1]);
```

Một trong những điều quan trọng nhất của một DataRow là phiên bản của nó. Điều đó cho phép bạn nhận được những giá trị khác nhau cho một dòng cụ thể. Các phiên bản được mô tả trong bảng sau:

DataRowVersion Value	Description
Current	Giá trị sẵn có của cột. Nếu không xảy một hiệu chỉnh nào, nó sẽ mang giá trị gốc. Nếu có một hiệu chỉnh xảy ra, giá trị sẽ là giá trị hợp lệ cuối cùng được cập nhật.
Default	Giá trị mặc định (nói một cách khác, giá trị mặc định được cài đặt cho cột).
Original	Giá trị của cột trong cơ sở dữ liệu vào lúc chọn. Nếu phương thức AcceptChanges DataRow được gọi, thì giá trị này sẽ được cập nhật thành giá trị hiện tại.
Proposed	Khi các thay đổi diễn ra trên một dòng nó có thể truy lục giá trị thay đổi này. Nếu bạn gọi BeginEdit() trên một dòng và tạo các thay đổi, mỗi một cột giữ một giá trị cho đến khi

DataRowVersion Value	Description
	phương thức EndEdit() hoặc CancelEdit() được gọi.

Phiên bản của một cột có thể dùng theo nhiều cách. Một ví dụ cho việc cập nhật các dòng trong cơ sở dữ liệu, đó là một câu lệnh SQL phổ biến như sau:

```
UPDATE Products
SET   Name = Column.Current
WHERE ProductID = xxx
AND   Name = Column.Original;
```

Rõ ràng mã này không bao giờ được biên dịch, nhưng nó chỉ ra một cách dùng cho các giá trị hiện tại và gốc của một cột trong một dòng.

Để trả về một giá trị từ DataRow, dùng các phương thức chỉ mục thừa nhận một giá trị DataRowVersion như là một tham số. Đoạn mã sau đây chỉ ra cách đạt được tất cả các giá trị cho mỗi cột của một DataTable:

```
foreach (DataRow row in ds.Tables["Customers"].Rows )
{
    foreach ( DataColumn dc in ds.Tables["Customers"].Columns )
    {
        Console.WriteLine (" {0} Current = {1} ", dc.ColumnName ,
                           row[dc,DataRowVersion.Current]);

        Console.WriteLine ("   Default = {0} ", row[dc,DataRowVersion.Default]);
        Console.WriteLine ("   Original = {0} ", row[dc,DataRowVersion.Original]);
    }
}
```

Mỗi dòng có một cờ trạng thái gọi là RowState, nó có thể dùng để xác định thực thi nào là cần thiết cho dòng đó khi nó cập nhật cơ sở dữ liệu. Thuộc tính RowState có thể được cài đặt để theo dõi tất cả các trạng thái thay đổi trên DataTable, như thêm vào các dòng mới, xóa các dòng hiện tại, và thay đổi các cột bên trong bảng. Khi dữ liệu được cập nhật vào cơ sở dữ liệu, cờ trạng thái được dùng để nhận biết thực thi SQL nào sẽ xảy ra. Những cờ này được định nghĩa bởi bảng liệt kê DataRowState:

DataRowState Value	Description
Added	Dòng được vừa mới được thêm vào tập hợp DataTable's Rows. Tất cả các dòng được tạo trên máy khách đều được cài đặt giá trị này, và cuối cùng là phát ra câu lệnh SQL INSERT khi cập nhật cho cơ sở dữ liệu.
Deleted	Giá trị này cho biết dòng đó có thể được đánh dấu xóa trong DataTable bởi phương thức DataRow.Delete(). Dòng này vẫn tồn tại trong DataTable, nhưng không thể trông thấy từ màn hình (trừ khi một DataView được cài đặt rõ ràng). Các DataView sẽ được trình bày trong chương tiếp theo. Các dòng được đánh dấu trong DataTable sẽ bị xóa khỏi cơ sở dữ liệu khi nó được cập nhật.
Detached	Một dòng sẽ có trạng thái này ngay sau khi nó được tạo ra , và có thể cũng trả về trạng thái này bởi việc gọi phương thức DataRow.Remove(). Một dòng detached không được coi là một thành phần của bảng dữ liệu.
Modified	Một dòng sẽ được Modified nếu giá trị trong cột bất kì bị thay đổi.

DataRowState Value	Description
Unchanged	Một dòng sẽ không thay đổi kể từ lần cuối cùng gọi AcceptChanges().

Trạng thái của một dòng phụ thuộc vào phương thức mà dòng đó đã gọi. Phương thức AcceptChanges() thường được gọi sau một cập nhật dữ liệu thành công (có nghĩa là sau khi thực hiện cập nhật cơ sở dữ liệu).

Cách phổ biến nhất để thay đổi dữ liệu trong một DataRow là sử dụng chỉ số, tuy vậy nếu bạn có một số thay đổi bạn cũng cần gọi các phương thức BeginEdit() và EndEdit() methods.

Khi một cập nhật được tạo ra trên một cột trong một DataRow, sự kiện ColumnChanging sẽ được phát ra trên các dòng của DataTable. Nó cho phép bạn ghi đè lên thuộc tính ProposedValue của các lớp DataColumnChangeEventArgs, và thay đổi nó nếu muốn. Cách này cho phép các giá trị trên cột có hiệu lực. Nếu bạn gọi BeginEdit() trước khi tạo thay đổi, sự kiện ColumnChanging vẫn xảy ra. Chúng cho phép bạn tạo một sự thay đổi kép khi cố gọi EndEdit(). Nếu bạn muốn phục hồi lại giá trị gốc, hãy gọi CancelEdit().

Một DataRow có thể liên kết với một vài dòng khác của dữ liệu. Điều này cho phép tạo các liên kết có thể điều khiển được giữa các dòng, đó là kiểu master/detail. DataRow chứa một phương thức GetChildRows() dùng để thay đổi một mảng các dòng liên quan đến các cột từ một bản khác trong cùng DataSet như là dòng hiện tại. Chúng sẽ được trình bày trong phần *Data Relationships* nằm ở phần sau của chương này.

3.5.4 Schema Generation

Có ba cách để tạo một schema cho một DataTable. Đó là:

- Hãy để thời gian chạy làm điều đó giúp bạn

- Viết mã tạo các bảng
- Dùng trình tạo sơ đồ XML

3.5.4.1 Runtime Schema Generation

Ví dụ về DataRow ở trên đã chỉ ra mã để chọn dữ liệu từ một cơ sở dữ liệu và tạo ra một DataSet:

```
SqlDataAdapter da = new SqlDataAdapter(select , conn);  
DataSet ds = new DataSet();  
da.Fill(ds , "Customers");
```

Nó rõ ràng dễ sử dụng, nhưng nó cũng có một vài trở ngại. Một ví dụ là bạn phải làm việc với tên cột được chọn từ cơ sở dữ liệu, điều đó cũng tốt thôi, nhưng chắc rằng muốn đổi tên vật lý thành tên thân thiện hơn.

Bạn có thể thực hiện việc đổi tên một cách thủ công trong mệnh đề SQL, chẳng hạn như trong `SELECT PID AS PersonID FROM PersonTable`; bạn luôn được cảnh báo không nên đổi tên các cột trong SQL, chỉ thay thế một cột khi thật sự cần để tên xuất hiện trên màn hình được thân thiện hơn.

Một vấn đề tiềm ẩn khác không các trình phát DataTable/DataColumn tự động là bạn không thể điều khiển vượt quá kiểu của cột, các kiểu này được thời gian chạy lựa chọn cho bạn. Nó rất có ích trong việc chọn kiểu dữ liệu đúng cho bạn, nhưng trong nhiều trường hợp bạn muốn có nhiều khả năng hơn. Ví dụ bạn cần định nghĩa một tập các kiểu giá trị dùng cho một cột, vì vậy mã cần phải được viết lại. Nếu bạn chấp nhận kiểu giá trị mặc định cho các cột được tạo ra trong thời gian chạy, có thể là một số nguyên 32-bit.

Cuối cùng một điều rất quan trọng, đó là sử dụng các trình tạo bảng tự động, bạn không thể truy xuất dữ liệu access to the data within the DataTable – you are at the mercy of indexers, which return instances of object rather than derived data types. If you like sprinkling your code with typecast expressions then skip the following sections.

3.5.4.2 Hand-Coded Schema

Việc phát ra mã để tạo một DataTable, với đầy đủ các cột là một việc tương đối đơn giản. Các ví dụ trong phần này sẽ truy cập bảng Products từ cơ sở dữ liệu Northwind.

Products				
	Column Name	Data Type	Length	Allow Nulls
🔑	ProductID	int	4	
	ProductName	nvarchar	40	
	SupplierID	int	4	✓
	CategoryID	int	4	✓
	QuantityPerUnit	nvarchar	20	✓
	UnitPrice	money	8	✓
	UnitsInStock	smallint	2	✓
	UnitsOnOrder	smallint	2	✓
	ReorderLevel	smallint	2	✓
	Discontinued	bit	1	

Dưới đây là mã để tạo thủ công một DataTable, có sơ đồ như trên.

```
public static void ManufactureProductDataTable(DataSet ds)
{
    DataTable products = new DataTable("Products");
    products.Columns.Add(new DataColumn("ProductID", typeof(int)));
    products.Columns.Add(new DataColumn("ProductName", typeof(string)));
    products.Columns.Add(new DataColumn("SupplierID", typeof(int)));
    products.Columns.Add(new DataColumn("CategoryID", typeof(int)));
    products.Columns.Add(new DataColumn("QuantityPerUnit", typeof(string)));
    products.Columns.Add(new DataColumn("UnitPrice", typeof(decimal)));
}
```

```
products.Columns.Add(new DataColumn("UnitsInStock", typeof(short)));
products.Columns.Add(new DataColumn("UnitsOnOrder", typeof(short)));
products.Columns.Add(new DataColumn("ReorderLevel", typeof(short)));
products.Columns.Add(new DataColumn("Discontinued", typeof(bool)));
ds.Tables.Add(products);
}
```

Bạn có thể sửa đổi mã trong ví dụ DataRow và sử dụng các định nghĩa sau:

```
string source = "server=localhost;" +
    "integrated security=sspi;" +
    "database=Northwind";

string select = "SELECT * FROM Products";

SqlConnection conn = new SqlConnection(source);

SqlDataAdapter cmd = new SqlDataAdapter(select, conn);

DataSet ds = new DataSet();

ManufactureProductDataTable(ds);

cmd.Fill(ds, "Products");

foreach(DataRow row in ds.Tables["Products"].Rows)

    Console.WriteLine("{0}' from {1}", row[0], row[1]);
```

Phương thức ManufactureProductDataTable() tạo một DataTable mới, thay đổi cho từng cột, và sau đó thêm nó vào danh sách các bảng trong DataSet. DataSet có một bộ chỉ mục nắm giữ tên của bảng và trả về DataTable được gọi.

Ví dụ trên không thật sự là bảo toàn kiểu, Tôi đã dùng bộ chỉ mục cột để lấy dữ liệu. Tốt hơn hết là dùng một lớp (hoặc một bộ các lớp) để điều khiển các DataSet,

DataTable, và DataRow, dùng để định nghĩa các bộ truy xuất bảo vệ kiểu cho các bảng, các dòng, các cột. Bạn có thể viết mã của mình – đó quả là một công việc chán nản, bạn có thể sử dụng các lớp bảo vệ kiểu sẵn có.

.NET Framework có các hỗ trợ cho việc dùng các sơ đồ XML để định nghĩa một DataSet, DataTable, và các lớp khác mà chúng ta có thể làm trong phần này.

3.5.5 Các quan hệ dữ liệu (Relationships)

Khi viết một ứng dụng, thường cần phải có sẵn nhiều bảng để lưu trữ thông tin. Lớp DataSet là một nơi để chứa các thông tin này.

Lớp DataSet là một thiết kế để tạo nên các mối quan hệ giữa các các bảng. Mã trong phần này được tôi thiết kế để tạo bằng tay mối quan hệ cho hai bảng dữ liệu. Vì vậy, nếu bạn không có SQL Server hoặc cơ sở dữ liệu NorthWind, bạn cũng có thể chạy ví dụ này.

```
DataSet ds = new DataSet("Relationships");  
ds.Tables.Add(CreateBuildingTable());  
ds.Tables.Add(CreateRoomTable());  
ds.Relations.Add("Rooms",  
    ds.Tables["Building"].Columns["BuildingID"],  
    ds.Tables["Room"].Columns["BuildingID"]);
```

Các bảng đơn giản chứa một khóa chính và một trường tên, trong đó bảng Room có một khóa ngoại BuildingID.



Sau đó thêm một số dữ liệu cho mỗi bảng.

```
foreach(DataRow theBuilding in ds.Tables["Building"].Rows)
{
    DataRow[] children = theBuilding.GetChildRows("Rooms");
    int roomCount = children.Length;

    Console.WriteLine("Building {0} contains {1} room{2}",
        theBuilding["Name"],
        roomCount,
        roomCount > 1 ? "s" : "");

    // Loop through the rooms
    foreach(DataRow theRoom in children)
    {
        Console.WriteLine("Room: {0}", theRoom["Name"]);
    }
}
```

Sự khác biệt lớn nhất giữa DataSet và kiểu đối tượng Recordset cổ điển là sự biểu hiện của quan hệ. Trong một Recordset cổ điển, một quan hệ được biểu diễn là một cột giả trong dòng. Cột này bản thân nó là một Recordset có thể lặp lại. Trong ADO.NET, một quan hệ đơn giản là một lời gọi phương thức GetChildRows():

```
DataRow[] children = theBuilding.GetChildRows("Rooms");
```

Phương thức này có một số kiểu, ví dụ trên chỉ ra cách dùng tên của quan hệ. Nó trả về một mảng các dòng có thể cập nhật bằng bộ chỉ mục như đã đề cập ở các ví dụ trước đây.

Thích thú hơn là quan hệ dữ liệu có thể xem xét theo hai cách. Không chỉ có thể đi từ cha đến con, mà có thể tìm được các dòng cha của một mẫu tin con bằng cách sử dụng thuộc tính ParentRelations trên lớp DataTable. Thuộc tính này trả về một

DataRelationCollection, có thể truy cập bằng kí hiệu mảng [] (ví dụ, ParentRelations["Rooms"]), hoặc dùng luân phiên phương thức GetParentRows() như mã dưới đây:

```
foreach(DataRow theRoom in ds.Tables["Room"].Rows)
{
    DataRow[] parents = theRoom.GetParentRows("Rooms");
    foreach(DataRow theBuilding in parents)
        Console.WriteLine("Room {0} is contained in building {1}",
            theRoom["Name"],
            theBuilding["Name"]);
}
```

Có hai phương thức với rất nhiều các cài đặt đề khác nhau để trả về các dòng cha – GetParentRows() (trả về một mảng các dòng), hoặc GetParentRow() (trả về một dòng cha duy nhất của một quan hệ).

3.5.6 Ràng buộc dữ liệu

Thay đổi kiểu dữ liệu của một cột đã được tạo trên một máy đơn không chỉ là một khả năng tuyệt vời của một DataTable. ADO.NET cho phép bạn tạo một tập các ràng buộc trên một cột (hoặc nhiều cột), dùng cho các nguyên tắc chuẩn hóa dữ liệu.

Thời gian chạy hỗ trợ các kiểu ràng buộc sau, như là các lớp trong không gian System.Data.

Constraint	Description
ForeignKeyConstraint	Thực một liên kết giữa hai DataTables trong một DataSet

Constraint	Description
UniqueConstraint	Bảo đảm tính độc nhất của cột

3.5.6.1 Cài đặt khóa chính

Một điều phổ biến của một bảng trong một cơ sở dữ liệu quan hệ, bạn có thể cung cấp một khóa chính, dựa vào một hoặc nhiều cột trong một DataTable.

Mã sau tạo một khóa chính cho bảng Products, mà sơ đồ của nó đã được tạo bằng thủ công trong các ví dụ trên.

Chú ý rằng một khóa chính của một bảng chỉ là một kiểu của ràng buộc. Khi một khóa chính được thêm vào một DataTable, thời gian chạy cũng phát ra một ràng buộc độc nhất trên khóa chính. Bởi vì thực tế không tồn tại kiểu ràng buộc PrimaryKey – một khóa chính đơn giản là một ràng buộc duy nhất trên một hoặc nhiều cột.

```
public static void ManufacturePrimaryKey(DataTable dt)
{
    DataColumn[] pk = new DataColumn[1];
    pk[0] = dt.Columns["ProductID"];
    dt.PrimaryKey = pk;
}
```

Một khóa chính có thể bao gồm một vài cột, nó được xem như là một mảng các DataColumnns. Một khóa chính của một bảng được cài đặt trên những cột này đơn giản được coi là một mảng của các cột làm nên thuộc tính.

Để kiểm tra các ràng buộc của một bảng, bạn có thể lập lại ConstraintCollection. Đối với ràng buộc tự sinh như ví dụ trên, tên của ràng buộc sẽ là Constraint1. Nó không phải là một tên tốt, vì vậy tốt nhất là nên tạo ràng buộc trước sau đó định nghĩa các cột tạo nên khóa chính, như chúng ta sẽ làm dưới đây.

Là một lập trình viên cơ sở dữ liệu lâu năm, tôi nhận thấy rằng tên của một ràng buộc cần phải thật rõ nghĩa. Mã dưới đây định danh ràng buộc trước khi tạo khóa chính:

```
DataColumn[] pk = new DataColumn[1];

pk[0] = dt.Columns["ProductID"];

dt.Constraints.Add(new UniqueConstraint("PK_Products", pk[0]));

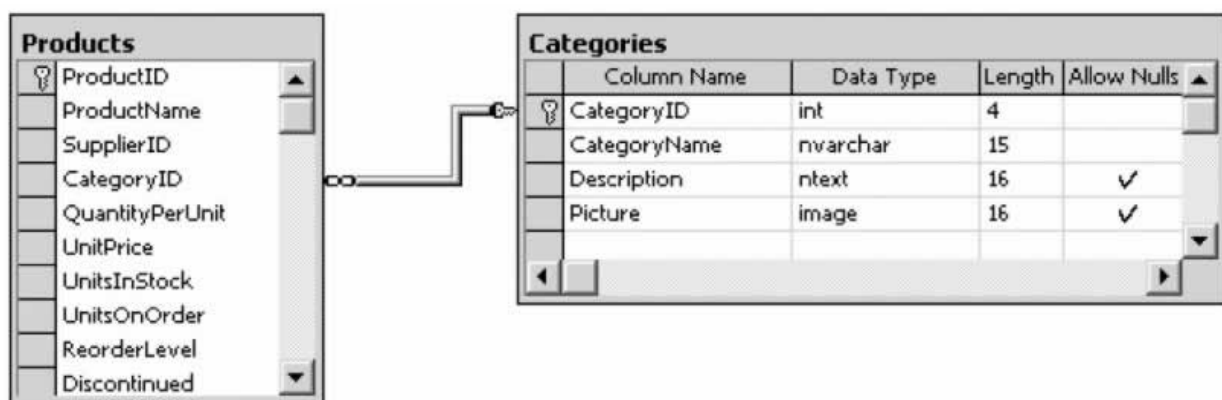
dt.PrimaryKey = pk;
```

Ràng buộc duy nhất có thể áp dụng cho bao nhiêu cột tùy thích.

3.5.6.2 Tạo một khóa ngoại

Ngoài các ràng buộc duy nhất, một DataTable có thể chứa các ràng buộc khóa ngoại. Nó thường được áp dụng cho các mối quan hệ chủ tớ, nhưng cũng có thể dùng để tạo bảng sao các cột giữa các bảng nếu bạn tạo một ràng buộc chính xác. Một quan hệ chủ tớ là một mẫu tin cha có thể có nhiều mẫu tin con, liên kết với khóa chính của mẫu tin cha.

Một ràng buộc khóa ngoại có thể chỉ thực thi trên các bảng bên trong một DataSet, ví dụ dưới đây sử dụng bảng Categories trong cơ sở dữ liệu Northwind, và tạo một ràng buộc giữa nó với bảng Products table.



Bước đầu tiên là tạo một bảng dữ liệu mới cho bảng Categories.

```
DataTable categories = new DataTable("Categories");
```

```
categories.Columns.Add(new DataColumn("CategoryID", typeof(int)));
categories.Columns.Add(new DataColumn("CategoryName", typeof(string)));
categories.Columns.Add(new DataColumn("Description", typeof(string)));
categories.Constraints.Add(new UniqueConstraint("PK_Categories",
        categories.Columns["CategoryID"]));
categories.PrimaryKey = new DataColumn[1]
        { categories.Columns["CategoryID"]};
```

Dòng cuối cùng của mã trên tạo một khóa chính cho bảng Categories. Khóa chính là một cột đơn, tất nhiên nó cũng thể tạo một khóa chính trên nhiều cột bằng các dùng kí tự mảng.

Sau đó tôi tạo một ràng buộc giữa hai bảng:

```
DataColumn parent = ds.Tables["Categories"].Columns["CategoryID"];
DataColumn child = ds.Tables["Products"].Columns["CategoryID"];
ForeignKeyConstraint fk =
    new ForeignKeyConstraint("FK_Product_CategoryID", parent, child);
fk.UpdateRule = Rule.Cascade;
fk.DeleteRule = Rule.SetNull;
ds.Tables["Products"].Constraints.Add(fk);
```

Ràng buộc này dùng để liên kết giữa Categories.CategoryID và Products.CategoryID. Có bốn cấu trúc khác nhau cho ForeignKeyConstraint, nhưng tôi khuyên bạn nên dùng tên của ràng buộc.

3.5.6.3 Tạo các ràng buộc Update và Delete

Bổ sung cho phần định nghĩa tất nhiên là một vài kiểu của ràng buộc giữa các bảng cha và con, bạn có thể định nghĩa phải làm gì trong một ràng buộc cập nhật.

Ví dụ trên tạo một qui tắc cập nhật và một qui tắc xóa. Những qui tắc này được dùng khi một sự kiện được phát ra trên cột (hoặc dòng) trong bảng cha, và qui tắc được dùng để quyết định chuyện gì sẽ xảy ra trong bảng con. Có bốn qui tắc khác nhau có thể áp dụng được liệt kê trong Rule enumeration:

- Cascade – Nếu khóa cha được cập nhật sau đó copy giá trị mới này cho tất cả các mã của khóa con. Nếu mẫu cha bị xóa, thì xóa luôn các mẫu con. Nó là tùy chọn mặc định.
- None – Không làm gì hết. Tùy chọn này sẽ bỏ các dòng mô côi khỏi bảng dữ liệu con.
- SetDefault – Mỗi thay đổi trên dòng con được mang giá trị mặc định của nó, nếu nó được định nghĩa trước.
- SetNull – Tất cả các dòng được chọn là DBNull.

Chú ý: Các ràng buộc chỉ có hiệu lực trong một DataSet nếu thuộc tính EnforceConstraints của DataSet là true.

3.6 DataSet

Trước tiên bạn đã định nghĩa sơ đồ của bộ dữ liệu của bạn, với đầy đủ các DataTable, DataColumn, Constraint, và những gì cần thiết, bạn nên tạo DataSet với một vài thông tin bổ sung. Có hai cách chính để đọc dữ liệu từ một nguồn bên ngoài và chèn nó vào DataSet:

- Dùng trình cung cấp dữ liệu
- Đọc XML vào trong DataSet

3.6.1 Tạo một DataSet dùng một DataAdapter

Đoạn mã về dòng dữ liệu được giới thiệu trong lớp SqlDataAdapter, được trình bày như sau:

```
string select = "SELECT ContactName,CompanyName FROM Customers";  
SqlConnection conn = new SqlConnection(source);  
SqlDataAdapter da = new SqlDataAdapter(select , conn);  
DataSet ds = new DataSet();  
da.Fill(ds , "Customers");
```

SqlDataAdapter và OleDbDataAdapter là hai lớp xuất phát từ một lớp cơ bản chứ không phải là một bộ các giao diện, và nhất là các lớp SqlClient- hoặc OleDb. Cây kế thừa được biểu diễn như sau:

```
System.Data.Common.DataAdapter  
System.Data.Common.DbDataAdapter  
System.Data.OleDb.OleDbDataAdapter  
System.Data.SqlClient.SqlDataAdapter
```

Trong quá trình lấy dữ liệu từ một DataSet, cần phải có một vài lệnh được dùng để chọn dữ liệu. Nó có thể là một câu lệnh SELECT, một stored procedure, hoặc OLE DB provider, một TableDirect command. Ví dụ trên sử dụng một trong những cấu trúc sẵn có trong SqlDataAdapter để truyền câu lệnh SELECT vào một SqlCommand, và phát nó khi gọi phương thức Fill() trên adapter.

Trở lại với các ví dụ về stored procedures trong chương trước, Tôi đã định nghĩa các stored procedure INSERT, UPDATE, và DELETE, nhưng chưa đưa ra một procedure để SELECT dữ liệu. Chúng ta sẽ lấp lỗ hổng này trong phần sau, và chỉ ra cách làm sao để gọi một stored procedure từ một SqlDataAdapter để tạo dữ liệu cho một DataSet.

3.6.2 Sử dụng một Stored Procedure trong một DataAdapter

Trước tiên chúng ta cần định nghĩa một stored procedure và cài nó vào cơ sở dữ liệu database. Stored procedure để SELECT dữ liệu như sau:

```
CREATE PROCEDURE RegionSelect AS  
  
SET NOCOUNT OFF  
  
SELECT * FROM Region  
  
GO
```

Ví dụ này tương đối đơn giản nó thật không xứng tầm với một stored procedure, chỉ là một câu lệnh SQL đơn giản. Stored procedure này có thể đánh vào SQL Server Query Analyzer, hoặc bạn có thể chạy file StoredProc.sql để sử dụng ví dụ này.

Tiếp theo, chúng ta cần định nghĩa một SqlCommand để thực thi stored procedure này. Một lần nữa mã rất đơn giản, và hầu hết đã được đưa ra trong các phần trên:

```
private static SqlCommand GenerateSelectCommand(SqlConnection conn )  
{  
  
    SqlCommand aCommand = new SqlCommand("RegionSelect" , conn);  
  
    aCommand.CommandType = CommandType.StoredProcedure;  
  
    aCommand.UpdatedRowSource = UpdateRowSource.None;  
  
    return aCommand;  
  
}
```

Phương thức này phát ra SqlCommand để gọi thủ tục RegionSelect khi thực thi. Và cuối cùng là móc nối nó với một SqlDataAdapter thông qua lời gọi phương thức Fill():

```
DataSet ds = new DataSet();  
  
// Create a data adapter to fill the DataSet
```

```
SqlDataAdapter da = new SqlDataAdapter();  
  
// Set the data adapter's select command  
  
da.SelectCommand = GenerateSelectCommand (conn);  
  
da.Fill(ds , "Region");
```

Ở đây tôi tạo một SqlDataAdapter mới, xem SqlCommand được phát ra thông qua thuộc tính SelectCommand của data adapter, và gọi Fill(), để thực thi stored procedure và chèn tất cả các dòng vào the Region DataTable.

3.7 Thay đổi DataSet

Sau khi soạn thảo dữ liệu trong một DataSet, cũng có những lúc cần phải thay đổi nó. Một ví dụ khá phổ biến đó là chọn dữ liệu từ một cơ sở dữ liệu, biểu diễn nó cho người dùng, và cập nhật cho cơ sở dữ liệu.

3.7.1 Cập nhật với DataAdapter

Một SqlDataAdapter có thể bao gồm SelectCommand, một InsertCommand, UpdateCommand, và DeleteCommand. Giống như tên gọi, những đối tượng này là những thể hiện của SqlCommand (hoặc OleDbCommand dùng cho OleDbDataAdapter), vì vậy những câu lệnh này có thể chuyển thành SQL hoặc một stored procedure.

Trong ví dụ này, tôi đã khôi phục lại các mã stored procedure từ phần *Calling Stored Procedures* để chèn, cập nhật, và xóa các mẫu tin Region.

3.7.1.1 Chèn một dòng mới

Có hai cách để thêm một dòng mới vào một DataTable. Cách thứ nhất là gọi phương thức NewRow, để trả về một dòng trống sau đó định vị và thêm vào tập Rows như sau:

```
DataRow r = ds.Tables["Region"].NewRow();  
  
r["RegionID"]=999;
```

```
r["RegionDescription"]="North West";  
ds.Tables["Region"].Rows.Add(r);
```

Cách thứ hai để thêm một dòng mới là truyền một mảng dữ liệu vào phương thức Rows.Add() giống như sau:

```
DataRow r = ds.Tables["Region"].Rows.Add  
(new object [] { 999 , "North West" });
```

Mỗi dòng trong DataTable sẽ cài RowState là Added. Ví dụ sẽ xổ ra các mẫu tin trước khi nó thay đổi được cập nhật cho dữ liệu, vì vậy sau khi thêm các dòng sau vào DataTable, các dòng sẽ giống như sau. Chú ý rằng cột bên phải là trạng thái dòng.

3.7.1.2 *New row pending inserting into database*

1	Eastern	Unchanged
2	Western	Unchanged
3	Northern	Unchanged
4	Southern	Unchanged
	999 North West	Added

Để cập nhật cơ sở dữ liệu từ một DataAdapter, gọi phương thức Update() như sau đây:

```
da.Update(ds , "Region");
```

Đối với một dòng mới trong DataTable, sẽ thực thi stored procedure và xuất ra các mẫu tin trong DataTable một lần nữa.

3.7.1.3 *New row updated and new RegionID assigned by database*

1	Eastern	Unchanged
2	Western	Unchanged
3	Northern	Unchanged

4 Southern Unchanged

5 North West Unchanged

Hãy nhìn dòng cuối của DataTable. Tôi đã nhập RegionID trong mã là 999, nhưng sau khi sẽ đổi RegionInsert stored procedure giá trị được đổi thành 5. Có sở dữ liệu thường tạo khoá chính cho bạn.

```
SqlCommand aCommand = new SqlCommand("RegionInsert" , conn);

aCommand.CommandType = CommandType.StoredProcedure;

aCommand.Parameters.Add(new SqlParameter("@RegionDescription" ,

    SqlDbType.NChar ,

    50 ,

    "RegionDescription"));

aCommand.Parameters.Add(new SqlParameter("@RegionID" ,

    SqlDbType.Int,

    0 ,

    ParameterDirection.Output ,

    false ,

    0 ,

    0 ,

    "RegionID" , // Defines the SOURCE column

    DataRowVersion.Default ,

    null));

aCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;
```

Chuyện gì sẽ xảy ra khi một data adapter phát các lệnh này, các tham số xuất sẽ được ánh xạ trở lại mã nguồn của dòng. Stored procedure có một tham số xuất ánh xạ trở lại DataRow.

Giá trị của UpdateRowSource được cho trong bảng sau:

UpdateRowSource Value	Description
Both	Một stored procedure có thể trả về nhiều tham số xuất và cũng có thể là một cơ sở dữ liệu gồm các mẫu tin đã được cập nhật.
FirstReturnedRecord	Nó trả về một mẫu dữ liệu đơn, nội dung của mẫu tin đó có thể được trộn vào DataRow nguồn. Nó có ích đối với một bảng có các cột mang giá trị mặc định hoặc tính toán, sau khi một câu lệnh INSERT chúng cần phải được đồng bộ với các DataRow trên máy trạm. Một ví dụ có thể là be 'INSERT (columns) INTO (table) WITH (primarykey)', sau đó là 'SELECT (columns) FROM (table) WHERE (primarykey)'. Các mẫu tin trả về có thể trộn vào tập các dòng.
None	Tất cả dữ liệu trả về từ câu lệnh đều bị vứt bỏ.
OutputParameters	Bất kì tham số xuất nào của câu lệnh đều được ánh xạ vào một cột thích hợp trong DataRow.

3.7.1.4 Cập nhật một dòng đã có

Cập nhật một dòng có sẵn trong DataTable chỉ là một trường hợp việc sử dụng bộ chỉ mục của lớp DataRow với tên của một cột hoặc số thứ tự của cột, giống như ví dụ sau đây:

```
r["RegionDescription"]="North West England";
```

```
r[1] = "North East England";
```

Các hai câu lệnh đều cho cùng kết quả:

Changed	RegionID	5	description
1	Eastern		Unchanged
2	Western		Unchanged
3	Northern		Unchanged
4	Southern		Unchanged
5	North West England		Modified

Trong quá trình cập nhật cơ sở dữ liệu, trạng của dòng được cập nhật sẽ được gán là Modified.

3.7.1.5 RowXóa một dòng

Xóa một dòng là kết quả của việc gọi phương thức Delete():

```
r.Delete();
```

Một dòng được xóa có trạng thái là Deleted, nhưng bạn không thể đọc các cột từ một dòng đã xóa, nó không còn giá trị nữa. Khi gọi phương thức Update(), tất cả các dòng bị xóa sẽ sử dụng DeleteCommand, trong trường hợp này sẽ chạy stored procedure RegionDelete.

3.8 Làm việc với ADO.NET

Phần cuối cùng này sẽ cố gắng đưa ra nhưng kịch bản phổ biến khi phát triển các ứng dụng truy cập cơ sở dữ liệu với ADO.NET.

3.8.1 Phân tầng các ứng dụng

Việc sản xuất các phần mềm tương tác với dữ liệu thường chia ứng dụng thành nhiều tầng. Một mô hình phổ biến của một ứng dụng phân tầng là các dịch vụ dữ liệu phân tầng, và một cơ sở dữ liệu phân tầng.

Một trong những cái khó của mô hình này là việc phân tách dữ liệu giữa các tầng, và định dạng truyền giữa các tầng. ADO.NET đã giải quyết các vấn đề này và đã sớm hỗ trợ cho kiểu cấu trúc này.

3.8.2 Sao chép và trộn dữ liệu

Thật khó để copy một DB recordset? Trong In .NET thật dễ dàng để sao chép một DataSet:

```
DataSet source = {some dataset};  
  
DataSet dest = source.Copy();
```

Nó tạo một bản copy của DataSet nguồn – từng DataTable, DataColumn, DataRow, và Relation sẽ được sao chép y chẵn, và tất cả dữ liệu với các trạng thái trong file nguồn đều được sao chép. Nếu như bạn chỉ muốn sao chép sơ đồ của DataSet, bạn có thể làm như sau:

```
DataSet source = {some dataset};  
  
DataSet dest = source.Clone();
```

Nó chỉ sao chép tất cả các table, relation, v.v. Tất nhiên, DataTable sẽ rỗng.

Một thực tế phổ biến khi viết các hệ thống phân tầng, dựa trên Win32 hoặc web, là có truyền dữ liệu giữa các lớp càng ít càng tốt.

DataSet có phương thức GetChanges() để giải quyết các yêu cầu này. Phương thức đơn giản này thực thi một loạt các công việc và trả về một DataSet với những dòng được cập nhật trong dataset nguồn. Đây là ý tưởng truyền dữ liệu giữa các tầng, chỉ một tập nhỏ dữ liệu được truyền.

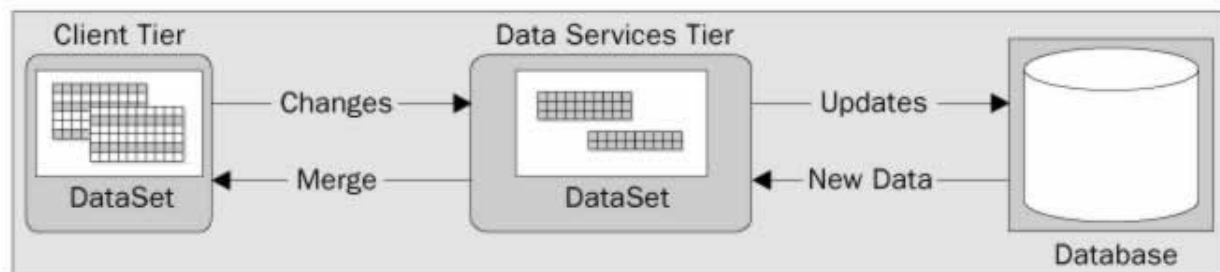
Ví dụ sau chỉ ra cách tạo một "changes" DataSet:

```
DataSet source = {some dataset};  
  
DataSet dest = source.GetChanges();
```

Bên dưới lớp vỏ bọc là rất nhiều thứ hấp dẫn. Có hai quá tải của phương thức `GetChanges()`. Một quá tải lấy giá trị của một `DataRowState`, và chỉ trả về các trạng thái tương ứng. `GetChanges()` đơn giản gọi `GetChanges(Deleted | Modified | Added)`, và kiểm tra nếu để bảo đảm rằng có một vài thay đổi bằng cách gọi `HasChanges()`. Nếu không có thay đổi nào, một giá trị được trả về ngay lập tức.

Tiếp theo là sao chép `DataSet`. Trước tiên, một `DataSet` mới bỏ qua các ràng buộc (`EnforceConstraints = false`), sau đó mỗi dòng đã thay đổi được sao chép vào một `DataSet` mới.

Như vậy bạn có một `DataSet` chỉ chứa các thay đổi, sau đó bạn có thể truyền dữ liệu này qua các tầng để xử lý. Khi dữ liệu được cập nhật vào cơ sở dữ liệu, "changes" `DataSet` có thể trả về cho trình gọi (trong ví dụ này, một vài tham số xuất từ các stored procedure đã cập nhật trong các cột). Những thay đổi này có thể trộn vào bộ `DataSet` bằng cách dùng phương thức `Merge()`. Tiến trình này được mô tả như sau:



3.8.3 Tạo khoá với SQL Server

Stored procedure `RegionInsert` trong ví dụ ở phần trước đã từng tạo ra một giá trị khóa chính để chèn vào cơ sở dữ liệu. Phương thức tạo khoá đó còn thô sơ và không linh động, vì vậy một ứng dụng thực tế cần dùng đến các kĩ thuật tạo khóa cao cấp hơn.

Đầu tiên có thể là định nghĩa một định dạng cột đơn giản, và trả về giá trị `@@IDENTITY` từ một stored procedure. Stored procedure dưới đây sử dụng bảng `Categories` trong cơ sở dữ liệu `Northwind`. Gõ stored procedure này vào `SQL Query Analyzer`:

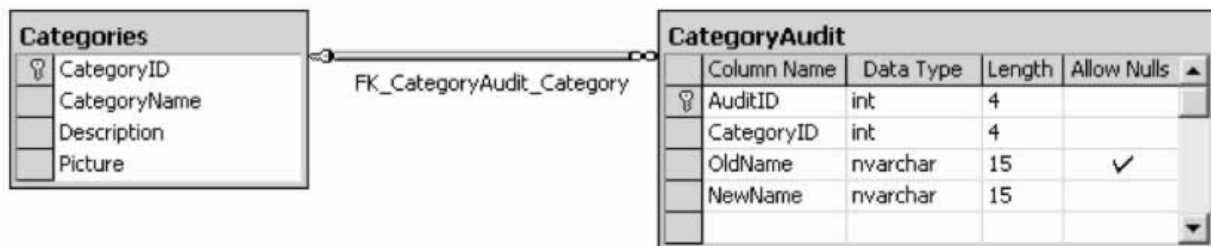
```
CREATE PROCEDURE CategoryInsert(@CategoryName NVARCHAR(15),  
                                @Description NTEXT,  
                                @CategoryID INTEGER OUTPUT) AS  
  
SET NOCOUNT OFF  
  
INSERT INTO Categories (CategoryName, Description)  
  
VALUES(@CategoryName, @Description)  
  
SELECT @CategoryID = @@IDENTITY  
  
GO
```

Nó chèn một dòng mới vào bảng Category, và trả về khóa chính cho trình gọi. Bạn có thể kiểm tra procedure này bằng cách gõ dòng SQL sau vào Query Analyzer:

```
DECLARE @CatID int;  
  
EXECUTE CategoryInsert 'Pasties' , 'Heaven Sent Food' , @CatID OUTPUT;  
  
PRINT @CatID;
```

Khi thực thi một bó lệnh, nó sẽ chèn một dòng mới vào bảng Categories, và trả về nhận dạng của dòng mới này, sau đó biểu diễn cho người dùng.

Giả sử rằng sau một vài tháng sử dụng, một ai đó muốn có một sổ theo dõi đơn giản, để báo cáo những cập nhật và sửa đổi trên category name. Bạn sẽ định nghĩa một bảng như sau, để chỉ ra các giá trị mới và cũ của category:



Mã sẵn có trong StoredProcs.sql. Cột AuditID được định nghĩa như một cột IDENTITY. Sau đó bạn cấu trúc một cặp trigger để báo cáo các thay đổi trên trường CategoryName:

```
CREATE TRIGGER CategoryInsertTrigger
ON Categories
AFTER UPDATE
AS
INSERT INTO CategoryAudit(CategoryID , OldName , NewName )
SELECT old.CategoryID, old.CategoryName, new.CategoryName
FROM Deleted AS old,
     Categories AS new
WHERE old.CategoryID = new.CategoryID;
GO
```

Bạn phải dùng Oracle stored procedure, SQL Server không hỗ trợ nội dung OLD và NEW của các dòng, thay vì chèn một trigger nó có một bộ bảng trong bộ nhớ gọi là Inserted, để xóa và cập nhật, các dòng cũ tồn tại trong bảng Deleted.

Trigger này nhận CategoryID cho các cột giả và lưu các giá trị cũ và mới của cột CategoryName.

Giờ đây, khi bạn gọi một stored procedure để chèn một CategoryID mới, bạn nhận một giá trị nhận dạng; Dĩ nhiên, nó không còn là giá trị nhận của dòng được chèn vào bảng Categories, nó là một giá trị mới được tạo trong bảng CategoryAudit. Ouch!

Để xem vấn đề, mở SQL Server Enterprise manager, xem nội dung của bảng Categories table.

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish
▶	20	Pasties	Heaven Sent Grub
*			

Bảng này liệt kê tất cả categories tôi có trong thể hiện của cơ sở dữ liệu.

Giá trị nhận dạng tiếp theo cho bảng Categories có thể là 21, vì vậy chúng ta sẽ chèn một dòng mới bằng cách thực thi mã sau đây, và xem nó trả về ID nào:

```
DECLARE @CatID int;

EXECUTE CategoryInsert 'Pasties', 'Heaven Sent Food', @CatID OUTPUT;

PRINT @CatID;
```

Giá trị trả về trên máy của tôi là 17. Khi xem bảng CategoryAudit, tôi nhận ra rằng đó là nhận dạng của dòng mới chèn trong bảng audit, không phải của category.

	AuditID	CategoryID	OldName	NewName
▶	17	30	<NULL>	Vegetables
*				

Đó là vì @@IDENTITY trả về giá trị nhận dạng cuối.

Có hai nhận dạng cơ bản bạn có thể sử dụng thay cho @@IDENTITY, chúng cũng không thể giải quyết vấn đề trên. Đầu tiên là SCOPE_IDENTITY(), sẽ trả về giá trị nhận dạng cuối cùng trong tầm vực hiện tại. SQL Server định nghĩa tầm vực như một stored procedure, trigger, hoặc hàm. Nếu vì một ai đó thêm một câu lệnh INSERT khác vào stored procedure, thì bạn sẽ nhận một giá trị không mong chờ.

IDENT_CURRENT() sẽ trả về giá trị nhận dạng cuối cùng được phát ra trên một bảng trong bất cứ tầm vực nào, trong trường hợp này, nếu hai người dùng đang truy cập SQL Server cùng một lúc, nó có thể nhận giá trị của người khác.

Chỉ có cách quản lý thủ công, bằng cách dùng cột IDENTITY trong SQL Server.

3.8.4 Performance

Bộ managed provider hiện tại của .NET có một vài giới hạn – bạn có thể chọn OleDb hoặc SqlClient; OleDb cho phép kết nối với bất kỳ nguồn dữ liệu nào nếu nó là một OLE DB driver (chẳng hạn như Oracle), còn SqlClient là một trình cung cấp dùng riêng cho SqlServer.

Trình cung cấp SqlClient đã được viết hoàn toàn bằng mã có quản, và sử dụng một vài lớp để kết nối cơ sở dữ liệu. Trình cung cấp viết các gói **TDS (Tabular Data Stream)** trực tiếp từ SQL Server, về bản chất nó nhanh hơn OleDb provider, nó có thể duyệt qua các lớp trước khi tác động vào cơ sở dữ liệu.

Để kiểm tra điều đó, hãy chạy mã sau trên cùng cơ sở dữ liệu, khác biệt ở chỗ sử dụng SqlClient managed provider trên ADO provider:

```
SqlConnection conn = new SqlConnection(Login.Connection);
conn.Open();

SqlCommand cmd = new SqlCommand ( "update tempdata set AValue=1 Where ID=1" ,
                                   conn);

DateTime initial, elapsed ;
initial = DateTime.Now ;
for(int i = 0; i < iterations; i++)
    cmd.ExecuteNonQuery();
```

```
elapsed = DateTime.Now ;
```

```
conn.Close();
```

OLE DB thường sử dụng OleDbCommand hơn là SqlCommand. Tôi đã tạo một bảng cơ sở dữ liệu nhỏ với hai cột như dưới đây, và điền vào đó một dòng đơn:

Câu lệnh SQL được sử dụng là một câu lệnh UPDATE đơn giản:

```
UPDATE TempData SET AValue = 1 WHERE ID = 1.
```

SQL là đơn giản nhất trong các provider. Kết quả tính bằng giây cho được liệt kê trong bảng sau:

Provider	100	1000	10000	50000
OleDb	0.109	0.798	7.95	39.11
Sql	0.078	0.626	6.23	29.27

Nếu bạn chỉ hướng vào SQL Server thì dĩ nhiên bạn sẽ chọn Sql provider. Trong thực tế, nếu bạn sử dụng các cơ sở dữ liệu khác bạn sẽ sử dụng OleDb provider.