ADO.NET

1. Giới thiệu về ADO.NET

ADO.NET là một tập hợp các lớp (classes) cho phép các ứng dụng trên nền .NET có thể truy xuất thông tin trong các CSDL và các nguồn dữ liệu khác. ADO.NET cung cấp các phương pháp truy xuất thích hợp với nhiều loại dữ liệu, bao gồm CSDL MS SQL Server, các CSDL tương thích OLEDB, các CSDL phi quan hệ như MS Exchange Server, và các văn bản XML.

1.1. Đặc điểm

ADO.NET mang lại nhiều tiện ích cho người phát triển ứng dụng (đặc biệt là với những người đã quen với Visual Basic hay ADO):

- Mô hình lập trình tương tự như trong ADO: Điều này mang lại sự thuận tiện cho những người phát triển đã quen với mô hình lập trình ADO.
- Được thiết kế để làm việc với dữ liệu ngắt kết nối: ADO.NET được thiết kế để làm việc với dữ liệu ngắt kết nối trong môi trường ứng dụng đa tầng (multitier). Nó sử dụng XML như là một chuẩn định dạng để truyền các dữ liệu ngắt kết nối, điều này giúp việc trao đổi thông tin với các ứng dụng đầu cuối không dựa trên nền tảng Windows trở nên dễ dàng hơn.
- Có sẵn trong .NET Framework: Vì ADO.NET có sẵn trong .NET Framework nên ta có tất cả các ưu điểm như của .NET Framework, bao gồm cả việc dễ dàng phát triển các ứng dụng đa ngôn ngữ lập trình.
- Hỗ trợ XML: ADO và XML trong quá khứ có đôi chỗ không tương thích: ADO làm việc dựa trên dữ liệu quan hệ còn XML thì làm việc dựa trên dữ liệu phân cấp có thứ bậc (hierachical data, dữ liệu XML thường bao gồm các nút có quan hệ cha-con). ADO.NET mang lại cả 2 khả năng truy xuất dữ liệu này và cho phép ta tích hợp dữ liệu quan hệ và dữ liệu phân cấp, tương tự như sự kết hợp xen kẽ giữa XML và mô hình lập trình quan hệ

Các mô hình truy xuất CSDL trước đây, ví dụ như DAO hay ADO, thường xoay quanh môi trường dữ liệu có kết nối chặt chẽ, đó chính là một mô hình lập trình truy xuất CSDL. Hầu hết các loại CSDL quan hệ hiện nay đều hỗ trợ cho mô hình lập trình Client/Server, trong mô hình này, các Clients kết nối với Database Server để truy xuất dữ liệu và duy trì sự kết nối trong suốt phiên làm việc. Ví dụ MS Access, client mở một Form và nạp dữ liệu vào các điều khiển của Form để thao tác sau đó cập nhật lại CSDL thông qua một kết nối riêng.

Trong môi trường của Internet, việc duy trì kết nối tới CSDL là điều rất khó khăn vì bản thân giao thức HTTP sử dụng trong các ứng dụng Web chỉ thiết lập kết nối khi trao đổi dữ liệu. Mặt khác trong mô hình ứng dụng nhiều tầng, n-tier, là một môi trường phân tán, dữ liệu sẽ được truyền giữa các tầng khi có yêu cầu để xử lý. Ví dụ, data tier kết nối để truy xuất dữ liệu sau đó truyền cho tầng tiếp theo. Tại thời điểm này, data tier sẽ hủy kết nối tới CSDL để trả lại tài nguyên cho client khác sử dụng.

Một trong số các mục tiêu của ADO.NET là mở rộng khả năng làm việc ngắt kết nối với nguồn dữ liệu. (ADO 2.0 bắt đầu được thiết kế để thao tác khi ngắt kết nối với dữ liệu nguồn. Data Source Object và Remote Data Service của IE cho phép cache dữ liệu trên Client, recoredset được chuyển sang kiểu MIME và truyền qua HTTP nhưng tất cả chỉ là tập hợp giá trị các bản ghi). Nhiều đối tượng mà ta thường dùng của ADO đã được phát triển tương ứng lên thành các đối tượng của ADO.NET (như Connection, Command). Mặc dù vậy, ADO.NET còn có nhiều lớp mới để phục vụ mở rộng mô hình truy xuất dữ liệu như: DataSet, DataReader và DataAdapter

XML đã trở thành một chuẩn lưu trữ và trao đổi dữ liệu vì tệp XML chỉ là tệp text có thể trao đổi dễ dàng qua HTTP mà không cần phải chuyển đổi sang MIME hoặc UU. Các ứng dụng hiện nay đã được xây dựng để chuyển dữ liệu sang định dạng XML như SQL Server 2000, Oracle 8i, 9i. ASP.NET sử dụng file config với định dạng XML và Web Services sử dụng giao diện và dữ liệu XML thực thi bởi SOAP

ADO 1.0 và ADO 2.0 hoàn toàn không hỗ trợ XML mà IE phụ trách việc thao tác, từ phiên bản 2.1 XML mới được chuyển giao cho ADO. ADO và XML trong quá khứ có đôi chỗ không tương thích: ADO làm việc dựa trên dữ liệu quan hệ còn XML thì làm việc dựa trên dữ liệu phân cấp có thứ bậc (hierachical data, dữ liệu XML thường bao gồm các nút có quan hệ cha-con). ADO.NET mang lại cả 2 khả năng truy xuất dữ liệu này và cho phép ta tích hợp dữ liệu quan hệ và dữ liệu phân cấp, tương tự như sự kết hợp xen kẽ giữa XML và mô hình lập trình quan hệ

1.2. Các trình điều khiển và dữ liệu

ADO.NET sử dụng các trình cung cấp dữ liệu (hay còn gọi là các trình điều khiển dữ liệu) .NET để liên kết ứng dụng với nguồn dữ liệu. Trình cung cấp dữ liệu của .NET thì tương tự như trình cung cấp OLE-DB đã được sử dụng trong ADO. Mặc dù chúng chủ yếu liên quan đến việc đưa dữ liệu vào và ra khỏi cơ sở dữ liệu hơn là cung cấp giao diện cho tất cả các tính năng của CSDL.

ADO.NET 2.0 có 4 trình điều khiển dữ liệu .Net:

- Dành cho SQL Server 7 trở lên
- Dành cho Oracle
- Dành cho CSDL tương thích OLEDB
- Dành cho CDSL ODBC

Các Namespaces chứa các trình điều khiển dữ liệu .NET gồm:

- System.Data Chứa các lớp chuẩn để truy xuất dữ liệu
- System.Data.Common Chứa các lớp được chung bởi các provider
- System.Data.Odbc Chứa các lớp ODBC provider
- System.Data.Oledb Chứa các lớp OLE-DB provider

- System.Data.ProviderBase Lớp cơ sở mới chứa các trình điều khiển
- System.Data.Oracle Chứa các lớp Oracle provider
- System.Data.Sql Lớp mới chứa giao diện và các lớp truy xuất SQL
- System.Data.SqlClient Chứa các lớp SQL provider
- System.Data.SqlTypes Kiểu dữ liệu của SQL Server

- System.Xml Chứa các lớp thao tác với XML
- System.Xml.Schema Chứa các lớp thao tác với XML schema
- System.Xml.Serialization Chuyển đổi định dạng XML truyền qua SOAP
- System.Xml.Xpath Thao tác với tệp XML
- System.Xml.Xsl Chuyển đổi định dạng XML theo XSL hoặc XSLT

ADO.NET chứa tập hợp các lớp thao tác dữ liệu sử dụng chung cho các loại dữ liệu được định nghĩa trong System.Data

- DataSet Thiết kế sử dụng dữ liệu ngắt kết nối
- DataTable Bảng dữ liệu
- DataRow Bản ghi
- DataColumn Trường
- DataRelation Quan hệ giữa các bảng
- Constraint Quy định của trường

System.Data.Comm
on chứa hai lớp dữ liệu:

- DataColumnMapping Tên trường CSDL và tên trường trong bảng
- DataTableMapping Tên bảng CSDL và tên bảng trong DataSet

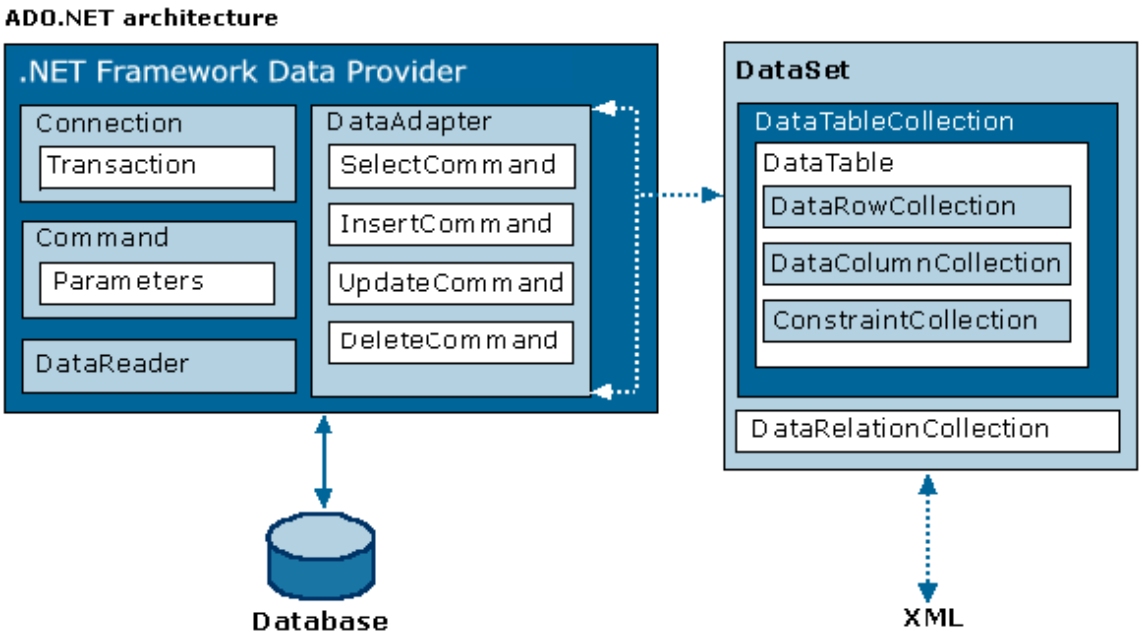
Ngoài các lớp dữ liệu sử dụng chung, ADO.NET cung cấp các lớp dữ liệu để thao tác với các loại dữ liệu khác nhau của các trình điều khiển. Tất cả các lớp dữ liệu này thực thi interface định nghĩa trong System.Data, bởi vậy chúng có thể được sử dụng giống nhau. Ví dụ, SqlConnection và OleDbConnection thực thi từ IDbConnection

- <provider>Connection Kết nối với CSDL (tương tự như trong ADO)
- <provider>Command Thao tác truy vấn SQL hoặc thủ tục lưu
- <provider>CommandBuilder Tạo các lệnh SQL
- <provider>DataAdapter Lưu trữ dữ liệu để nạp vào DataSet
- <provider>Parameter Định nghĩa tham số cho thủ tục lưu
- <provider>Transaction Đồng bộ hóa cập nhật CSDL

Các trình điều khiển dữ liệu cung cấp của ADO.NET có các công cụ cho phép ta đọc, nhập, sửa, xóa dữ liệu trong môi trường phân cấp đa tầng. Hầu hết các đối tượng

trong 4 loại thư viện là tương đương và được phân biệt bằng phần tiền tố của tên. Ví dụ: <Sql>DataReader và <OleDb>DataReader.

1.3. Mô hình các đối tượng trong ADO.NET



Mô hình truy xuất dữ liệu với ADO được thực hiện qua đối tượng lưu trữ dữ liệu - RecordSet. Trong mô hình này, ta tạo một liên kết tới CSDL sử dụng OLE-DB provider hoặc ODBC sau đó thực thi truy vấn CSDL và lưu kết quả trong RecordSet. Mô hình truy xuất dữ liệu của ADO.NET cũng áp dụng nguyên tắc chung như vậy nhưng sử dụng các đối tượng khác là DataReader và XmlReader.

Điểm khác biệt quan trọng giữa ADO.NET và các công nghệ truy xuất dữ liệu trước đó là trong ADO.NET có một đối tượng gọi là **DataSet** – đối tượng này làm việc tách biệt và độc lập với nguồn dữ liệu. Bởi vậy, đối tượng DataSet làm việc như một thực thể độc lập. Ta có thể hiểu DataSet như một/nhiều tập hợp các bản ghi luôn luôn ngắt kết nối và không cần biết đến thông tin về nguồn (source) hay đích (destination) của dữ liệu nằm trong nó. Bên trong DataSet thì tương tự như trong một cơ sở dữ liệu: có các bảng (tables), các trường dữ liệu (columns), quan hệ (relationships), các views và nhiều thứ khác

2. Truy xuất CSDL trong ADO.NET

Trong phần này, chúng ta chủ yếu đề cập đến .NET Framework Data Provider dành cho CSDL SQL Server được cung cấp trong namespace System.Data.SqlClient. Để làm việc với các CSDL khác như Access hay các CSDL tương thích OLEDB, ta chỉ cần sử dụng các đối tượng của Provider tương ứng.

2.1. Connections

Các lớp Connection trong ADO.NET bao gồm các thuộc tính và phương thức thông dụng để tạo liên kết:

Thuộc tính:

ConnectionString	Tham số kết nối
ConnectionTimeout	Thời gian hủy kết nối
Da	
tabase	Tên CSDL
DataSource	Tên CSDL trên Server
PacketSize	Kích thước gói tin
ServerVersion	Phiên bản của Server
WorkstationId	Tên CSDL trên client
State	Trạng thái kết nối.

Nhận giá trị enum của ConnectionState

Broken, Closed, Connecting, Executing, Fetching, Open

Phương thức:

SqlConnection()	Cấu tử mặc định
SqlConnection(string)	Cấu tử với tham số ConnectionString
BeginTransaction()	Tạo một phiên và trả lại đối tượng SqlTransaction
ChangeDatabase(name)	Chuyển kết nối tới CSDL mới
Close()	Đóng kết nối
CreateCommand()	Tạo đối tượng SqlCommand
Dispose()	Hủy đối tượng
Open()	Mở kết nối theo ConnectionString

2.1.1. Kết nối CSDL

Để truy xuất CSDL ta cần phải tạo liên kết tới CSDL và cung cấp các tham số cần thiết để truy nhập như Server, tên CSDL và tài khoản truy cập.

Ví dụ:

```
[Visual Basic]
Public Sub CreateSqlConnection()
   Dim myConnectString As String = "Persist Security Info=False; Integrated
Security=SSPI; database=northwind; server=mySQLServer; Connect Timeout=30"
   Dim myConnection As New SqlConnection(myConnectString)
   myConnection.Open()
End Sub 'CreateSqlConnection
```

[C#]

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

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    string conString = "Data Source=(local);Initial Catalog=Employee;
        Integrated Security=True";

    protected void Button1_Click(object sender, EventArgs e)
    {
        SqlConnection conObject = new SqlConnection(conString);

        try
        {
            conObject.Open();
            if (conObject.State == ConnectionState.Open)
                Response.Write("Database Connection is Open");
        }
        catch(SqlException sqlexception)
        {
            Response.Write("ERROR ::" + sqlexception.Message);
        }
        catch(Exception ex)
        {
            Response.Write("ERROR ::"+ex.Message);
        }
        finally
        {
            conObject.Close();
        }
    }
}
```

Thông thường ta phải quản lý thông tin kết nối CSDL (ConnectionString), lưu trong file cấu hình hoặc hard-code trong chương trình. Để lưu trữ thông tin liên kết CSDL trong tệp **web.config**

```
<configuration>
    <connectionStrings>
        <add    name=”NorthWind”
                ProviderName=”System.Data.SqlClient”
                connectionString=”server=(MySQLServer);
                Integrated Security=SSPI; database=NorthWind” />
    </connectionStrings>
</configuration>
```

Trong tệp **machine.config** thiết lập trình điều khiển dư liệu để cho phép truy xuất

```
<system.data>
    <DbProviderFactories>
        <add    name=”SqlClient Data Provider”
                invariant=”System.Data.SqlClient” support=”FF”
                description= “.Net Framework Data Provider for SqlServer”
                type=”System.Data.SqlClient.SqlClientFactory, System.Data,
                Version=2.0.3600.0, Culture=neutral,
                PublicKeyToken=b43a4d34444e345” />
    </DbProviderFactories >
</system.data>
```

Ta sử dụng thông tin kết nối trong tệp cấu hình bởi các lớp ConfigurationStringSettings và DbProviderFactory được định nghĩa trong System.Data.SqlClient

Ví dụ:

```
private DbConnection GetDatabaseConnection(string name) {
    ConnectionStringSettings setting=ConfigurationSettings.ConnectionStrings[name];
    DbProviderFactory factory=DbProviderFactories.GetFactory(setting.ProviderName);
    DbConnection conn = factory.CreateConnection();
    conn.ConnectionString = setting.ConnectionString;
    return conn;
}
```

2.1.2. Đồng bộ hóa

Thông thường khi có nhiều sự cập nhật tới CSDL, ta phải đồng bộ hóa trong transaction. Khi đồng bộ hóa việc cập nhật, ta bắt đầu bằng phương thức BeginTransaction(). Phương thức này trả lại đối tượng SqlTransaction thực thi IDbTransaction Interface định nghĩa trong System.Data.

Khi bắt đâu đồng bộ ta có thể chọn mức độ cách ly giữa các phiên cập nhật. Mức độ này xác định sự nhận biết những thay đổi trong một phiên cập nhật bởi các phiên khác. Mức độ cách ly bao gồm (không phải tất cả các CSDL đều hỗ trợ)

ReadCommitted: Mặc định cho SQL Server. Dữ liệu cập nhật của một phiên chỉ được phép truy nhập bởi các phiên khác khi đã commit

ReadUncommitted: Dữ liệu cập nhật của một phiên chỉ được phép truy nhập bởi các phiên khác

RepeattableRead: Nếu có thêm cùng cập nhật, đảm bảo trả lại cùng dữ liệu

Serializable: Không cho phép các phantom row hiển thị

Ví dụ:

```
string source = "Persist Security Info=False; Integrated Security=SSPI;
   database=northwind;server=mySQLServer;Connect Timeout=30";
SqlConnection conn = new SqlConnection(source);
myConnection.Open();
SqlTransaction tx = conn.BeginTransaction();
    // Các thao tác với dữ liệu
tx.Commit();
conn.Close();
```

2.1.3. Connection Pooling

Connection pooling (nhóm liên kết thường xuyên) cho phép ta tái sử dụng liên kết tới CSDL nguồn. Đây là kỹ thuật tối ưu hoá hiệu năng truy cập CSDL trong chương trình. Chế độ mặc định cho phép tính năng Connection pooling được bật với Connection String. Số lượng tối đa của nhóm liên kết thường xuyên là 100, và tối thiểu là 0.

Liên kết tới CSDL nguồn là tác vụ yêu cầu rất tốn thời gian và tài nguyên. Mỗi khi người dùng muốn liên kết tới CSDL, chuỗi liên kết sẽ được sẽ được phân tích cú pháp và kiểm tra định đạng chuỗi hợp lệ, đầy đủ các thông số xác thực. Bởi vậy sẽ tốn khá nhiều thời gian để mở và đóng liên kế

Để tối ưu hoá và giảm thiểu tài nguyên cho tác vụ mở và đóng cùng một liên kết thường xuyên tới một CSDL nguồn và tăng cường hiệu năng truy cập CSDL, ADO.NET sử dụng connection pooling là nhóm các liên kết thường xuyên truy cập CSDL.

2.2. Commands

Commands là các đối tượng cho phép truy xuất đến CSDL để lấy dữ liệu, cập nhật dữ liệu, thông qua các câu lệnh truy vấn SQL, chạy các truy vấn chứa sẵn (stored procedures) và gửi hoặc lấy thông tin qua các tham số (parameters). Commands bao gồm các thuộc tính và phương thức cơ bản

Thuộc tính:

CommandText	Chuỗi truy vấn hoặc tên thủ tục
CommandTimeout	Thời gian hủy thực thi câu lệnh
CommandType	Kiểu câu lệnh (Text, StoredProcedure, TableDirect)
Connection	Đối tượng Connection
Parameters	Tập hợp các đối tượng SqlParameter

Transaction	Đối tượng Transaction thực thi câu lệnh
UpdateRowSource	Xác định cách cập nhật khi sử dụng SqlDataAdapter

Phương thức:

SqlCommand()	
SqlCommand(CommandText)	
SqlCommand(CommandText, Connection)	
SqlCommand(CommandText, Connection, Transaction)	
Cancel()	Hủy thực thi
CreateParameter(name)	Tạo đối tượng SqlParameter
Dispose()	Hủy đối tượng
ExecuteNonQuery()	Thực thi truy vấn và trả lại số bản ghi bị tác động
ExecuteReader()	Thực thi và trả lại đối tượng SqlDataReader
ExecuteXmlReader()	Thực thi và trả lại đối tượng XmlReader
ExecuteScalar()	Thực thi và trả lại 1 giá trị đầu tiên
ResetCommandTimeout()	Thiết lập lại giá trị mặc định của CommandTimeout

2.2.1. Truy vấn CSDL quan hệ

Sau khi tạo đối tượng Command ta có thể thực thi truy vấn CSDL. Phương thức ExecuteNonquery thường sử dụng với các câu lệnh UPDATE, INSERT, hoặc DELETE khi ta chỉ cần quan tâm tới số bản ghi bị tác động. Tuy nhiên, ta có thể lấy kết quả nếu thực thi các thủ tục lưu có tham số trả về. Phương thức ExecuteReader trả lại kết quả là một đối tượng DataReader, sau đó ta có thể duyệt các bản ghi để thao tác. Phương thức ExecuteScalar trả lại một giá trị duy nhất, thông thường để đếm số bản ghi hoặc lấy thời gian trên server

Ví dụ:

```
using System;
using System.Data.SqlClient;
...
string source =     “server=(local); integrated security=SSPI;database=Northwind”;
string insert =     “INSERT INTO Customers(ContactName, CompanyName) ” +
                    “Values(‘Nguyen Van A’,’HanoiCom’)”;
SqlConnection conn = new SqlConnection(source);
conn.Open();
SqlCommand cmd = new SqlCommand(insert, conn);
int rowsReturn = cmd.ExecuteNonQuery()
response.write(rowsReturn);
conn.Close();
```

2.2.2. Truy vấn dữ liệu XML

SQL Server cho phép sử dụng câu lệnh truy vấn SELECT với mệnh đề FOR XML để chuyển định dạng dữ liệu theo 3 loại. Để truy vấn dữ liệu theo định dạng XML ta phải sử dụng Namespace System.Xml

FOR XML AUTO	Tạo dữ liệu dạng cây của bảng
FOR XML RAW	Chuyển bản ghi thành node, trường thành thuộc tính
FOR XML EXPLICIT	Cho phép ta xác định dạng thức của cây XML

Ví dụ:

```
using System;
using System.Data.SqlClient;
using System.Xml;
...
string source =  “server=(local); integrated security=SSPI; 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(); Response.Write(s);
} while (s != ””)
conn.Close();
```

2.2.3. Sử dụng tham số Parameters

Thuộc tính Parameters của Command bao gồm một tập hợp SqlParameterCollection, trong đó mỗi đối tượng SqlParameter là một tham số của SqlCommand. Để thực thi các thủ tục lưu bởi đối tượng Command, ta cần định nghĩa thủ tục lưu và các tham số sau đó gọi các phương thức truy vấn của đối tượng Command.

Để tạo một đối tượng tham số Parameter, ta có tập hợp phương thức

```
objParam = cmd.Parameters.Add()
objParam = cmd.Parameters.Add(properties)
```

trong đó properties là một hoặc các giá trị sau:

name	Tên tham số
db-type	Kiểu dữ liệu (SqlDbType, OleDbType...)
size	Kích thước (nguyên)
direction	Giá trị của ParameterDirection enum
is-nullable	Có chấp nhận giá trị Null (logic)
precision	Số lượng ký tự số
scale	Số chữ số thập phân
source-column name	Tên cột của bảng mà tham số lấy giá trị
source-column version	Giá trị của DataRowVersion enum (Current ...)
value	giá trị của tham số

Ví dụ:

```
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

SqlCommand cmd = new SqlCommand("RegionInsert", conn);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add(new SqlParameter("@RegionDescription"),
                                        SqlDbType.NChar, 50,
                                        "RegionDescription"));
cmd.Parameters.Add(new SqlParameter("@RegionID"),
                                        SqlDbType.Int, 0,
                                        ParameterDirection.Output,
                                        false, 0, 0,
                                        "RegionID",
                                        DataRowVersion.Default, null));
cmd.UpdatedRowSource = UpdateRowSource.OutputParameters;

cmd.Parameters["@RegionDescription"].Value = "South West"
cmd.ExecuteNonQuery();
int newRegionID = (int) cmd.Parameters["@RegionID"].Value;
if (newRegionID >0)
        Response.Write("Mã vùng mới được thêm vào là:" + newRegionID);
else
        Response.Write("Lỗi cập nhật bản ghi");
```

3. Đối tượng DataReader

DataReader là phương pháp đơn giản và nhanh nhất để lấy dữ liệu từ nguồn nhưng cũng là phương pháp ít khả năng thao tác với dữ liệu nhất. Để tạo một đối tượng SqlDataReader, ta phải thực hiện phương thức ExecuteReader() của đối tượng SqlCommand.

Sử dụng DataReader tương tự như RecordSet của ADO, ta có thể thực thi câu lệnh truy vấn SQL hoặc một thủ tục lưu để truy xuất một tập hợp bản ghi rồi duyệt tập hợp đó để thao tác với các bản ghi. DataReader sử dụng con trỏ "forward-only" nên ta chỉ có thể duyệt các bản ghi theo một chiều. Khi thực thi, DataReader giữ kết nối tới CSDL cho tới khi ta gọi phương thức close() để hủy liên kết.

Các thuộc tính và phương thức

Thuộc tính:

FieldCount	Chuỗi truy vấn hoặc tên thủ tục
HasRows	Thời gian hủy thực thi câu lệnh
IsClosed	Kiểu câu lệnh (Text, StoredProcedure, TableDirect)
Item	Đối tượng Connection
RecordAffected	Tập hợp các đối tượng SqlParameter
Transaction	Đối tượng Transaction thực thi câu lệnh

UpdateRowSource Xác định cách cập nhật khi sử dụng SqlDataAdapter

Phương thức:

Read() Chuyển con trỏ tới bản ghi tiếp theo

GetTYPE(ColumnIndex|ColumName) Lấy giá trị theo kiểu xác định

NextResult() Làm việc với tập bản ghi tiếp theo (nhiều SQL)

Close() Đóng DataReader

Ví dụ: ReadDR.aspx

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Page language="C#" %>
<div id="outResult" runat="server"/>
<script runat="server">
    void Page_Load(object s, EventArgs e) {
        string result = "<table>";
        string source = "server=(local); integrated security=SSPI;
        database=Northwind";
        string select = "SELECT * FROM Customers
            WHERE ContactName LIKE 'Ng%' ";
        SqlConnection conn = new SqlConnection(source);
        conn.Open();
        SqlCommand cmd = new SqlCommand(select, conn);
        SqlDataReader rd = cmd.ExecuteReader();
        while (rd.Read()) {
            result += "<tr><td>" + rd["ContactName"];
            result += "</td><td>" + rd["CompanyName"];
            result += "</td></tr>";
        }
        rd.Close();
        conn.Close();
        result += "</table>";
        outResult.InnerHtml = strResult;
    }
</script>
```

Chú ý: Thông thường DataReader được kết hợp với các ListControl bởi DataBinding để hiển thị dữ liệu mà không cần phải duyệt qua các bản ghi

```
using System;
using System.Data;
using System.Data.SqlClient;
public partial class DataBindRD : System.Web.UI.Page
{
   SqlDataReader reader;
   SqlConnection con;
   protected void Page_Load(object sender, EventArgs e)
```

```
    {
        con = new SqlConnection("Data Source=(local); Initial Catalog=Employee;
            Integrated Security=True");
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = con;
        cmd.CommandText = "select * from tblemps";
        cmd.CommandType = CommandType.Text;
        try
        {
            con.Open();
            reader = cmd.ExecuteReader();
            GridView1.DataSource = reader;
            GridView1.DataBind();
        }
        catch (Exception ex)
        {
            Label1.Text = "ERROR :: " + ex.Message;
        }
        finally
        {
            reader.Close();
            con.Close();
        }
    }
}
```

Kết quả:

EmpID	Name	Gender	Salary	Address	DEPID
1	Raj	Male	25000	Pune	1
2	Ramesh	Male	20000	Nagpur	1
3	David	Male	35000	USA	2
4	Zeny	Female	42000	Canada	2
5	Nishant	Male	50000	Pune	3
6	Digvijay	Male	45000	New York	3

4. Truy xuất dữ liệu ngắt kết nối

4.1. DataAdapter (SqlDataAdapter)

- SqlDataAdapter đóng vai trò là cầu nối giữa đối tượng DataSet và SQL Server để truy xuất, lấy và cập nhật dữ liệu. SqlDataAdapter thực hiện vai trò này nhờ các phương thức:
  - FillSchema: lấy cấu trúc từ nguồn dữ liệu, trong trường hợp ta phải đồng bộ dữ liệu về nguồn cần phải lấy đúng cấu trúc và khoá
  - Fill: lấy dữ liệu từ nguồn dữ liệu đưa vào DataSet, làm cho dữ liệu trong DataSet đồng bộ với dữ liệu ở nguồn dữ liệu.
  - Update: Cập nhật dữ liệu từ DataSet về nguồn dữ liệu, làm cho dữ liệu ở nguồn dữ liệu đồng bộ với những thay đổi trong DataSet
- Khi SqlDataAdapter điền dữ liệu vào DataSet (Fill), nó sẽ tạo ra các bảng (tables) cần thiết với các trường (fields) cho dữ liệu nó lấy về nếu các bảng đó chưa tồn tại trong DataSet. Mặc dù vậy, khóa chính (primary key) sẽ không được tự động tạo ra trong cấu trúc của mỗi table đó trừ khi thuộc tính MissiSchemaAction được nhận giá trị AddWithKey. Ta cũng có thể sử dụng SqlDataAdapter để tạo ra cấu trúc cho DataSet (bao gồm cả Khóa chính) trước khi điền dữ liệu, bằng cách sử dụng phương thức FillSchema. (Chú ý: Ta có thể sử dụng Constraints để tạo khoá chính)
- SqlDataAdapter được sử dụng với sự kết hợp của SqlConnection và SqlCommand để tăng hiệu suất khi làm việc với CSDL Microsoft SQL Server.
- Ngoài ra, SqlDataAdapter còn có các đối tượng SelectCommand, InsertCommand, DeleteCommand, UpdateCommand và TableMappings để giúp đơn giản hóa quá trình truy xuất và cập nhật dữ liệu.

4.2. Dataset

- Đối tượng DataSet, một bản Cache trong bộ nhớ của dữ liệu lấy về từ nguồn dữ liệu, là thành phần chính trong công nghệ ADO.NET. DataSet có tập hợp các đối tượng DataTable mà ta có thể thiết lập quan hệ giữa chúng bằng các đối tượng DataRelation. Ta cũng có thể áp đặt các ràng buộc dữ liệu trong DataSet sử dụng các đối tượng UniqueConstraint và ForeignKeyConstraint.
- Khi các đối tượng DataTable chứa dữ liệu, tập hợp DataRelationCollection cho phép ta duyệt qua sơ đồ của các bảng. Các bảng đươc chứa trong tập hợp DataTableCollection và có thể truy cập qua thuộc tính Tables. Khi truy xuất đến các đối tượng DataTable, lưu ý rằng tên của chúng là có phân biệt chữ hoa chữ thường tùy tình huống. Ví dụ: nếu có một DataTable tên là "mytable" và một DataTable khác tên là "MyTable" thì khi đó tên của mỗi DataTable là phân biệt chữ hoa chữ thường. Mặc dù vậy, nếu trong đó chỉ có bảng "mytable" còn bảng "MyTable" không tồn tại, thì tên bảng lại không phân biệt chữ hoa chữ thường.
- Đối tượng DataSet có thể đọc, ghi dữ liệu và cấu trúc ra thành các tài liệu XML. Dữ liệu và cấu trúc đó có thể được truyền qua giao thức HTTP và được sử dụng bởi bất kì ứng dụng, trên bất kì môi trường có hỗ trợ XML nào. Ta lưu cấu trúc của DataSet bằng phương thức WriteXmlSchema, và lưu cả cấu trúc với dữ liệu của DataSet

bằng phương thức WriteXml. Để đọc lại một tài liệu XML có chứa cấu trúc và dữ liệu, ta dùng phương thức ReadXml

- Các bước cơ bản để làm việc với DataSet là:
    - Tạo và điền dữ liệu vào các DataTable trong DataSet từ nguồn dữ liệu bằng cách sử dụng DataAdapter (ở đây chúng ta dùng SqlDataAdapter để làm việc với CSDL SQL Server)
    - Thay đổi dữ liệu trong DataTable bằng cách thêm (add), sửa (update), xóa (delete) các đối tượng DataRow (đối tượng thể hiện các bản ghi trong DataTable)
    - Gọi phương thức GetChanges để tạo ra DataSet thứ hai chứa những bản ghi đã bị thay đổi.
    - Gọi phương thức Update của DataAdapter để cập nhật DataSet thứ hai.
    - Gọi phương thức Merge để trộn những thay đổi đã được cập nhật vào nguồn dữ liệu của DataSet thứ hai vào DataSet ban đầu.
    - Gọi phương thức AcceptChanges của DataSet để chấp nhận các thay đổi hoặc gọi phương thức RejectChanges để từ chối thay đổi.

Ví dụ: DataSet

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
public partial class Default : System.Web.UI.Page
{
   SqlConnection conn;
   SqlDataAdapter adapter;
   DataSet ds;
   protected void Page_Load(object sender, EventArgs e)
   {
      string cs=ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
      try
      {
         conn = new SqlConnection(cs);
         adapter = new SqlDataAdapter("select * from tblEmps", conn);
         ds = new DataSet();
         adapter.Fill(ds);
         GridView1.DataSource = ds;
         GridView1.DataBind();
      }
      catch(Exception ex)
      {
         Label1.Text = "ERROR :: " + ex.Message;
      }
      finally
      {
```

```
            ds.Dispose();
            conn.Dispose();
        }
    }
}
```

Sự khác biệt khi sử dụng DataReader và DataSet

DataSet	DataReader
Làm việc với mô hình ngắt liên kết	Cung cấp môi trường làm việc có liên kết.
Hiệu năng truy cập dư liệu chậm hơn DataReader.	Truy cập dữ liệu nhanh.
Đối tượng trong bộ nhớ. Ta có thể thao tác với các bảng dữ liệu theo bất kỳ hình thức nào.	Đối tượng làm việc theo cơ chế forward-only và read only object.
Hỗ trợ mở đóng connection tự động	Cần phải thao tác open and close connection.
Có thể chứa nhiều bảng với các quy định ràng buộc và quan hệ.	Tại mỗi thời điểm chỉ làm việc với một tập hợp các bản ghi trong một bảng.
Dataset objects hỗ trợ XML	Không hoàn toàn hỗ trợ XML
Sử dụng phương thức fill() của DataAdapter để nạp dữ liệu cho dataset.	DataReader sử dụng phương thức read() để đọc từng bản ghi.

4.3. DataTable

- Đối tượng DataTable thể hiện một bảng dữ liệu trong bộ nhớ (In-memory data table).
- DataTable là đối tượng nằm trong trung tâm của ADO.NET. Các đối tượng khác có sử dụng DataTable là DataSet và DataView.
- Khi truy cập vào đối tượng DataTable, lưu ý tên bảng của mỗi DataTable là phân biệt chữ hoa chữ thường.
- Nếu ta tạo ra một DataTable bằng cách lập trình, trước hết ta phải định nghĩa cấu trúc của nó bằng cách thêm các đối tượng DataColumn vào trong tập hợp DataColumnCollection (truy xuất qua thuộc tính Columns)
- Để thêm bản ghi vào DataTable, trước hêt phải sử dụng thuộc tính NewRow để tạo ra một đối tượng DataRow. Phương thức NewRow trả lại một bản ghi theo cấu trúc của DataTable với các trường (cột) được định nghĩa trong DataColumnCollection. Tối đa một DataTable có thể chứa 16,777,216 bản ghi.

- Cấu trúc của DataTable được định nghĩa bởi tập hợp DataColumnCollection trong đó có chứa các DataColumn. Có thể truy xuất tập hợp DataColumnCollection thông qua thuộc tính DataColumns.
- DataTable chứa một tập hợp các đối tượng Contraint được dùng để đảm bảo các ràng buộc về dữ liệu.
- Để xác định khi nào các thay đổi xảy ra đối với dữ liệu trong bảng, ta sử dụng những sự kiện sau: RowChanged, RowChanging, RowDeleting, và RowDeleted. (các sự kiện này chủ yếu được dùng trong lập trình Windows Form).

Ví dụ: DataTableAddNew

```
using System;
using System.Data;
public partial class DataTableAddNew : System.Web.UI.Page
{
   DataTable dt;
   protected void btnAddNewRow_Click(object sender, EventArgs e)
   {
      DataRow dr = dt.NewRow();
      dr["EmployeeID"] = 666;
      dr["Name"] = "Atul";
      dr["City"] = "Mumbai";
      dr["Salary"] = 12000;
      dr["Department"] = "CS";

      dt.Rows.Add(dr);
      GridView1.DataBind();
   }
   protected void Page_Load(object sender, EventArgs e)
   {
      dt = new DataTable();

      dt.Columns.Add("EmployeeID", typeof(int));
      dt.Columns.Add("Name", typeof(string));
      dt.Columns.Add("City", typeof(string));
      dt.Columns.Add("Salary", typeof(double));
      dt.Columns.Add("Department", typeof(string));

      dt.Rows.Add(111, "Raj", "Nagpur", 45000, "IT");
      dt.Rows.Add(222, "Neha", "Kanpur", 20000,"Accounts");
      dt.Rows.Add(333, "Amit", "New York", 30000, "Management");
      dt.Rows.Add(444, "Digvijay", "Kanpur", 35000, "IT");
      dt.Rows.Add(555, "Rajesh", "Delhi", 25000, "HR");

      GridView1.DataSource = dt.DefaultView;
      GridView1.DataBind();
```

```
    }
}
```

4.4. DataView

- Thể hiện một cách nhìn (view) khác đối với dữ liệu trong DataTable: có thể kết gán với các điều khiển, sắp xếp, lọc, tìm kiếm, cập nhật và duyệt các bản ghi.
- Tính năng chính của DataView chính là để kết gán dữ liệu (DataBinding) trong cả Windows Forms và Web Forms.
- DataView còn có thể được cấu hình để thể hiện một tập con của dữ liệu từ DataTable. Khả năng này cho phép ta gán 2 điều khiển vào cùng một DataTable nhưng trên đó thể hiện các phiên bản khác nhau của dữ liệu. Ví dụ, một điều khiển được dùng để kết gán với một DataView thể hiện tất cả các bản ghi trong DataTable, trong khi điều khiển thứ hai thể hiện các bản ghi bị xóa khỏi DataTable.
- Để lọc và sắp xếp dữ liệu trong view, ta qui định thuộc tính RowFilter và thuộc tính Sort. Sau đó sử dụng thuộc tính Item để trả lại một DataRowView (view một bản ghi).
- Ta cũng có thể thêm hoặc xóa các bản ghi sử dụng các phương thức AddNew và Delete. Khi ta sử dụng các phương thức này, thuộc tính RowStateFilter có thể nhận các giá trị để chỉ những bản ghi bị xóa hoặc mới thêm có thể hiển thị trong DataView.

Ví dụ: DataViewSort

```
using System;
using System.Data;
public partial class DataViewSort : System.Web.UI.Page
{
   DataTable dt;
   protected void Page_Load(object sender, EventArgs e)
   {
      dt = new DataTable();

      dt.Columns.Add("EmployeeID", typeof(int));
      dt.Columns.Add("Name", typeof(string));
      dt.Columns.Add("City", typeof(string));
      dt.Columns.Add("Salary", typeof(double));
      dt.Columns.Add("Department", typeof(string));

      dt.Rows.Add(111, "Raj", "Nagpur", 45000, "IT");
      dt.Rows.Add(222, "Neha", "Kanpur", 20000,"Accounts");
      dt.Rows.Add(333, "Amit", "New York", 30000, "Management");
      dt.Rows.Add(444, "Digvijay", "Kanpur", 35000, "IT");
      dt.Rows.Add(555, "Rajesh", "Delhi", 25000, "HR");
```

```
        DataView dw = new DataView(dt);

        dw.Sort = ”Name”;
        GridView1.DataSource = dw;
        GridView1.DataBind();

        dw.Sort = “Salary ASC”;
        GridView2.DataSource = dw;
        GridView2.DataBind();

    }
}
```

4.5. CommandBuilder (SqlCommandBuilder)

Lớp dữ liệu **SqlCommandBuilder** cung cấp tính năng phản hồi các thay đổi trong DataSet hoặc một thực thể của SQL data. Khi đối tượng SqlCommandBuilder được tạo, nó tự động tạo câu lệnh Transact-SQL một bảng có sự thay đổi trạng thái dữ liệu. Đối tượng CommandBuilder thực hiện lắng nghe sự kiện RowUpdating khi thiết lập DataAdapter được thiết lập.

Đối tượng CommandBuilder tự động tạo các đối tượng Command tương ứng với các thay đổi giá trị dư liệu trong bảng. Các đối tượng InsertCommand, UpdateCommand và DeleteCommand được khởi tạo dựa vào đối tượng SelectCommand lấy dữ liệu ban đầu. Trong trường hợp có sự thay đổi SelectCommand của DataAdapter, ta phải gọi phương thức RefreshScheme để cập nhật lại thuộc tính command của DA.

Ví dụ: CommandBuilder

```
SqlConnection myConnection = new
        SqlConnection("server=(local)\\SQLExpress;Integrated
        Security=SSPI;database=northwind");
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter("Select * from Customers",
        myConnection);
DataSet myDataSet = new DataSet();
DataRow myDataRow;

mySqlDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
// mySqlDataAdapter.FillSchema(myDataSet, SchemaType.Mapped);
mySqlDataAdapter.Fill(myDataSet, "Customers");

myDataRow = myDataSet.Tables["Customers"].NewRow();
myDataRow["CustomerId"] = "NewID";
```

```
myDataRow["ContactName"] = "New Name";
myDataRow["CompanyName"] = "New Company Name";

myDataSet.Tables["Customers"].Rows.Add(myDataRow);

myDataSet.Tables["Customers"].Rows[0]["ContactName"]="Peach";

DataRow myDataRow1 = myDataSet.Tables["Customers"].Rows.Find("ALFKI");
myDataRow1["ContactName"]="Peach";

myDataSet.Tables["Customers"].Rows[0].Delete();

SqlCommandBuilder mySqlCommandBuilder = new
SqlCommandBuilder(mySqlDataAdapter);

//myConnection.Open();
SqlTransaction sqlTS;
myConnection.BeginTransaction();

mySqlDataAdapter.DeleteCommand = mySqlCommandBuilder.GetDeleteCommand();
mySqlDataAdapter.InsertCommand = mySqlCommandBuilder.GetInsertCommand();
mySqlDataAdapter.UpdateCommand = mySqlCommandBuilder.GetUpdateCommand();

Response.Write("Lệnh Delete" + mySqlDataAdapter.DeleteCommand.CommandText);
Response.Write("Lệnh Insert" + mySqlDataAdapter.InsertCommand.CommandText);
Response.Write("Lệnh UpDate" + mySqlDataAdapter.UpdateCommand.CommandText);

mySqlDataAdapter.DeleteCommand.Transaction = sqlTS;
mySqlDataAdapter.InsertCommand.Transaction = sqlTS;
mySqlDataAdapter.UpdateCommand.Transaction = sqlTS;
try{
        mySqlDataAdapter.Update(myDataSet, "Customers");
        sqlTS.Commit();
}
catch (SqlException sqlEX) {
        sqlTS.RollBack();
}
finally {
        myConnection.Close();
}
```

Tierd Development (next lect.)