Session 6

XML Intoduction

XML

- **Universal data language**
- **Plain-text data description**
- **More verbose than other formats (but very compressible)**
- **Enables Web Services, RSS, ATOM, etc…**
- **Consists of:**
  - Elements i.e. “nodes”
  - Attributes – attached to element, provide metadata
- **Characteristics:**
  - User defined
  - Case sensitive
  - Well formed
  - Transportable (crosses domain boundaries easily)

Sample XML Fragment

```
<Vehicle id='1234'>
  <Make>Jeep</Make>
  <Model>Wrangler</Model>
  <Color id='1'>Red</Color>
  <Year>1992</Year>
  <Price>12000</Price>
</Vehicle>
```

XML Uses in .NET

- **Config files**
- **Serialized representation of objects**
- **Typed DataSets**
- **Web Services**
- **Code Documentation**

Sending data across machine boundaries

- **Traditional Wire Format:**
  - Fixed-length buffer
    Reilly Douglas 14222345819560724Doug@me.com
  - Delimited
    Reilly,Douglas,142223458,19560724,Doug@me.com
    Lee,Frank,22321234,19920403,yellowfish@me.com

- **XML Format:**

```
<Customer id="142223458">
    <FirstName>Douglas</FirstName>
    <Last Name>Reilly</LastName>
    <DOB>19560724</DOB>
    <Email>Doug@me.com</Email>
</Customer>
```

Example: Write XML to a File

```
string xml = @"<?xml version='1.0' standalone='yes'?>
<Vehicles>
    <Vehicle id='1234'>
        <Make>Jeep</Make>
        <Model>Wrangler</Model>
        <Color id='1'>Red</Color>
        <Year>1992</Year>
        <Price>12000</Price>
    </Vehicle>
    <Vehicle id='2234'>
        <Make>Volkswagon</Make>
        <Model>Jetta</Model>
        <Color id='2'>Blue</Color>
        <Year>2002</Year>
        <Price>15500</Price>
    </Vehicle>
</Vehicles>";

XmlDocument doc = new XmlDocument();
doc.LoadXml(xml);
XmlTextWriter tw = new XmlTextWriter("vehicles.xml", null);
doc.WriteTo(tw);
tw.Close();
```

XML Rules

- **XML Declaration is required**
  `<?xml version="1.0 encoding="UTF-8" ?>`
- **One root element**
- **All opening tags must close**
- **Case sensitive**
- **Tags cannot overlap**
- **Attribute names in quotes**

XSL

- **XML Stylesheet Language**
- **Transform XML text to another format**
- **Declarative programming “language”**
- **Supported in .NET `System.Xml.Xsl` namespace**

XML in .NET

- **System.Xml namespace**
- **Xml characteristics are represented as classes:**
  - XmlDocument
  - XmlElement
  - XmlAttribute

Example Transforming XML

```
using System;
using System.IO;
using System.Text;
using System.Xml;
using System.Xml.Xsl;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            XmlDocument xdoc = new XmlDocument();
            xdoc.Load("Vehicles.xml");

            XslCompiledTransform xslt = new XslCompiledTransform();
            xslt.Load("Inventory.xsl");

            StringBuilder sb = new StringBuilder();
            StringWriter sw = new StringWriter(sb);
            xslt.Transform(xdoc, null, sw);

            Console.WriteLine(sb);

            Console.ReadKey();
        }
    }
}
```

Session 6

ADO.NET Data Access

Some Data Storage Techniques

- **Database**
  - SQL Server, Oracle, PostGRESQL, My SQL, DBase, IBM DB2, SQLite, MS-Access, SQL CE, etc…
- **File**
  - Text, CSV, Excel, XML
- **Cloud**
- **In Memory**

HOW CAN WE ACCESS ALL THESE DIFFERENT DATA STORES??

ADO.NET

- **API for accessing data**
- **Used to access/modify data stored in a data store**
- **Data store classified according to:**
  - The paradigm used to model the data
    - Columns, rows, flat hierarchy
  - The medium used to store the data
    - DB, Excel, CSV file
  - The mechanism used to query the data
    - SQL, text parsing

ADO.NET

- **OO set of libraries**
- **Allows access to database**
  - MS SQL Server
  - MS Access
  - Oracle
  - Borland Interbase
  - IBM DB2
  - MySQL
  - PostgreSQL
  - Excel
  - SqlAnywhere
  - Much more…

ADO.NET Namespaces

- **System.Data**
- **System.Data.Common**
- **Client specific:**
  - System.Data.SqlClient
  - MySql.Data.MySqlClient
  - Oracle.DataAccess.Client

ADO.NET Consists of

- Connected Layer
  - Data Provider - varies by database vendor (prefer vendor library over ODBC or OLEDB)
  - Connection
  - Command
  - DataReader
  - DataAdapter

- Disconnected Layer
  - Used with any provider
  - DataSet

Data Provider

- **Each provider publishes a common set of utility classes:**
  - Connection: connect to data source
  - Command: perform some action
  - Parameter: describe a single parameter to a command
  - DataAdapter: a bridge used to transfer data between a datasource and DataSet object.
  - DataReader: connection-oriented, used to process results one record at a time

```
System.Data.SqlClient.SqlConnection;
MySql.Data.MySqlConnection
System.Data.OleDb.OleDbConnection;
System.Data.Odbc.OdbcConnection
```

Connection Object

- Vendor Specific
- Requires a connection string
- Open a database
- Close after use
- “Open late, Close Early”

```
SqlConnection conn = new SqlConnection(@"Data Source=|DataDirectory|\Data
Directory\AutoSales.sdf");
```

Command Object

- **SQL Queries submitted to a DB**
  - Query string
  - Parameters
- **Returns results (DataSet or DataReader) or executes a command (such as delete)**
- **Vendor Specific**
- **Requires a connection**

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Vehicle", conn);
```

DataReader Object

- **Vendor specific**
- **Connection-oriented**
- **Use with small results**
- **Like a read-only, forward-only cursor**
- **Use Read method to advance cursor**

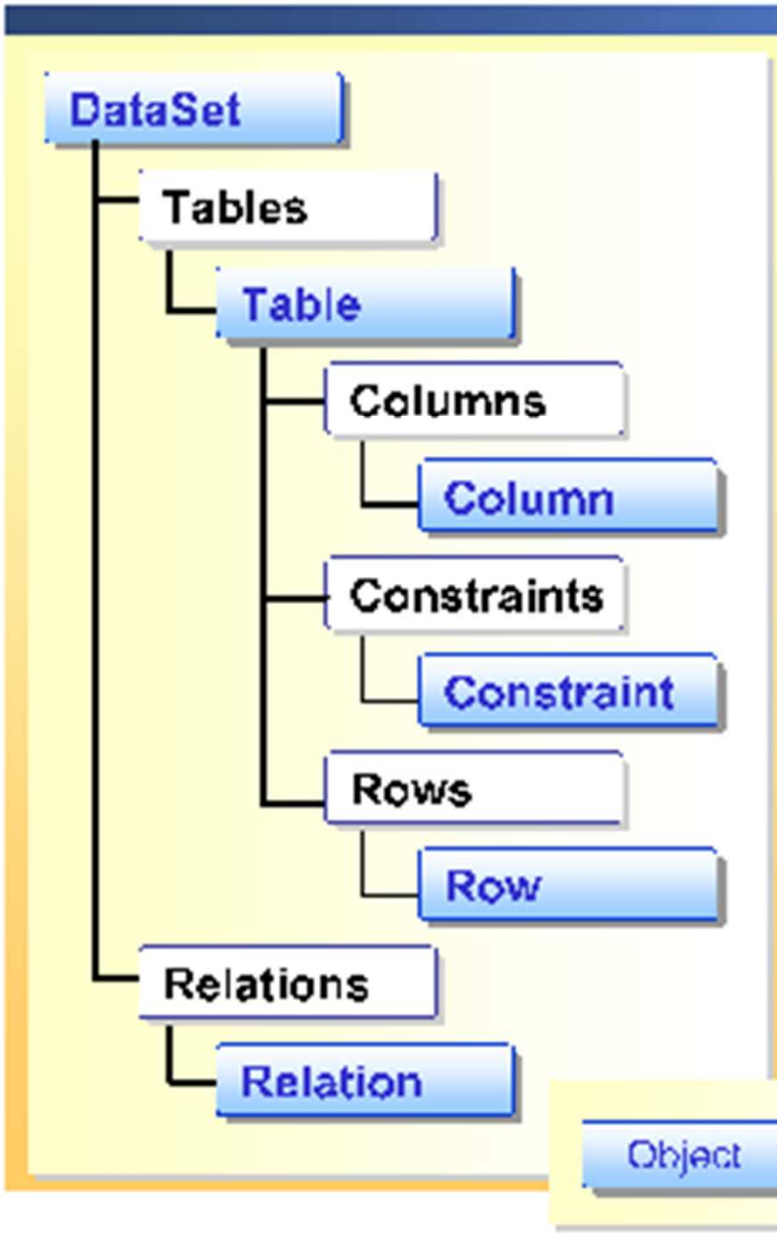
```
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
while (rdr.Read())
{
  Console.Write(rdr["id"]);
  Console.Write(rdr["Make"]);
  Console.WriteLine(rdr["Model"]);
}
conn.Close();
```

DataReader Example

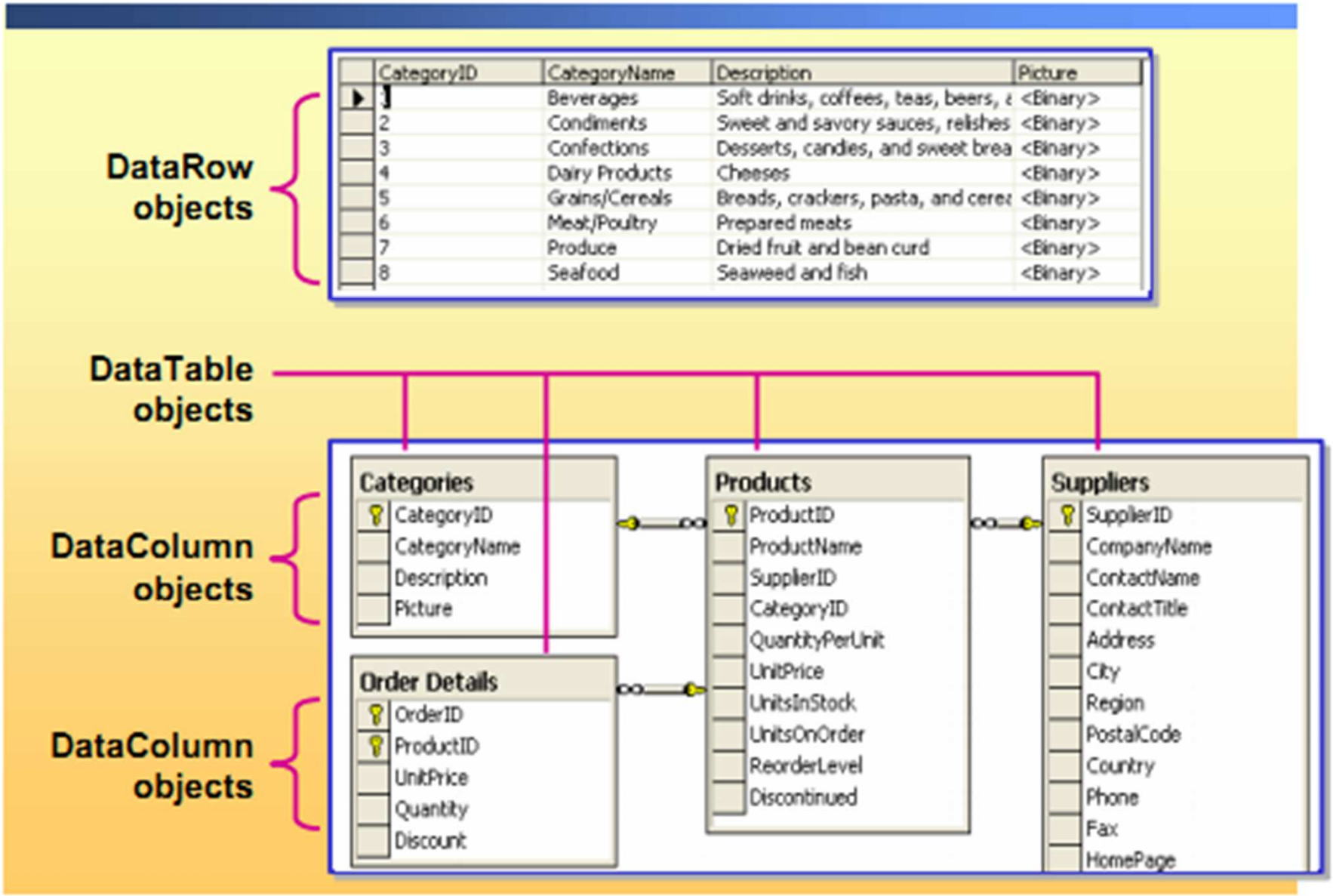
```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);Initial Catalog=AutoSales;Integrated Security=True";

SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.CommandText = "SELECT * FROM Vehicle";
cmd.CommandType = System.Data.CommandType.Text;

conn.Open();
IDataReader dr = cmd.ExecuteReader();

Console.WriteLine("Id\tMake\t\t\tModel\t\tColor\tYear\tPrice");
while (dr.Read())
{
    Console.WriteLine("{0}\t{1}\t\t\t{2}\t\t{3}\t{4}\t{5:c}"
        , dr.GetInt32(dr.GetOrdinal("Id"))
        , dr["Make"]
        , dr["Model"]
        , dr.GetInt32(dr.GetOrdinal("Color"))
        , dr.GetInt32(dr.GetOrdinal("Year"))
        , dr.GetInt32(dr.GetOrdinal("Price"))
        );
}

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

Data Sets

- **In-Memory DB**
- **Classes form a hierarchy**
- **Contains tables and relationships**
  - DataTable: a single table with rows and columns
    - DataView: sorting and filtering
    - DataRelation: relationship between tables
    - Constraint: describes an enforced property
- **Populated using a DataAdapter**
- **May be serialized and persisted (saved) to a file using the WriteXml() method**
- **Resurrect from file using ReadXml method()**



- Use **Fill** method of DataAdapter
- Populate programmatically by creating a table structure and filling it
- Read an XML document or stream into a DataSet
- Use **Merge** method to copy the contents of another **DataSet** object


DataRow objects
CategoryID
CategoryName
Description
Picture
1 Beverages Soft drinks, coffees, teas, beers, a <Binary>
2 Condiments Sweet and savory sauces, relishes <Binary>
3 Confections Desserts, candies, and sweet brea <Binary>
4 Dairy Products Cheeses <Binary>
5 Grains/Cereals Breads, crackers, pasta, and cerez <Binary>
6 Meat/Poultry Prepared meats <Binary>
7 Produce Dried fruit and bean curd <Binary>
8 Seafood Seaweed and fish <Binary>
DataTable objects
DataColumn objects
DataColumn objects
Categories
CategoryID
CategoryName
Description
Picture
Order Details
OrderID
ProductID
UnitPrice
Quantity
Discount
Products
ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued
Suppliers
SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
HomePage

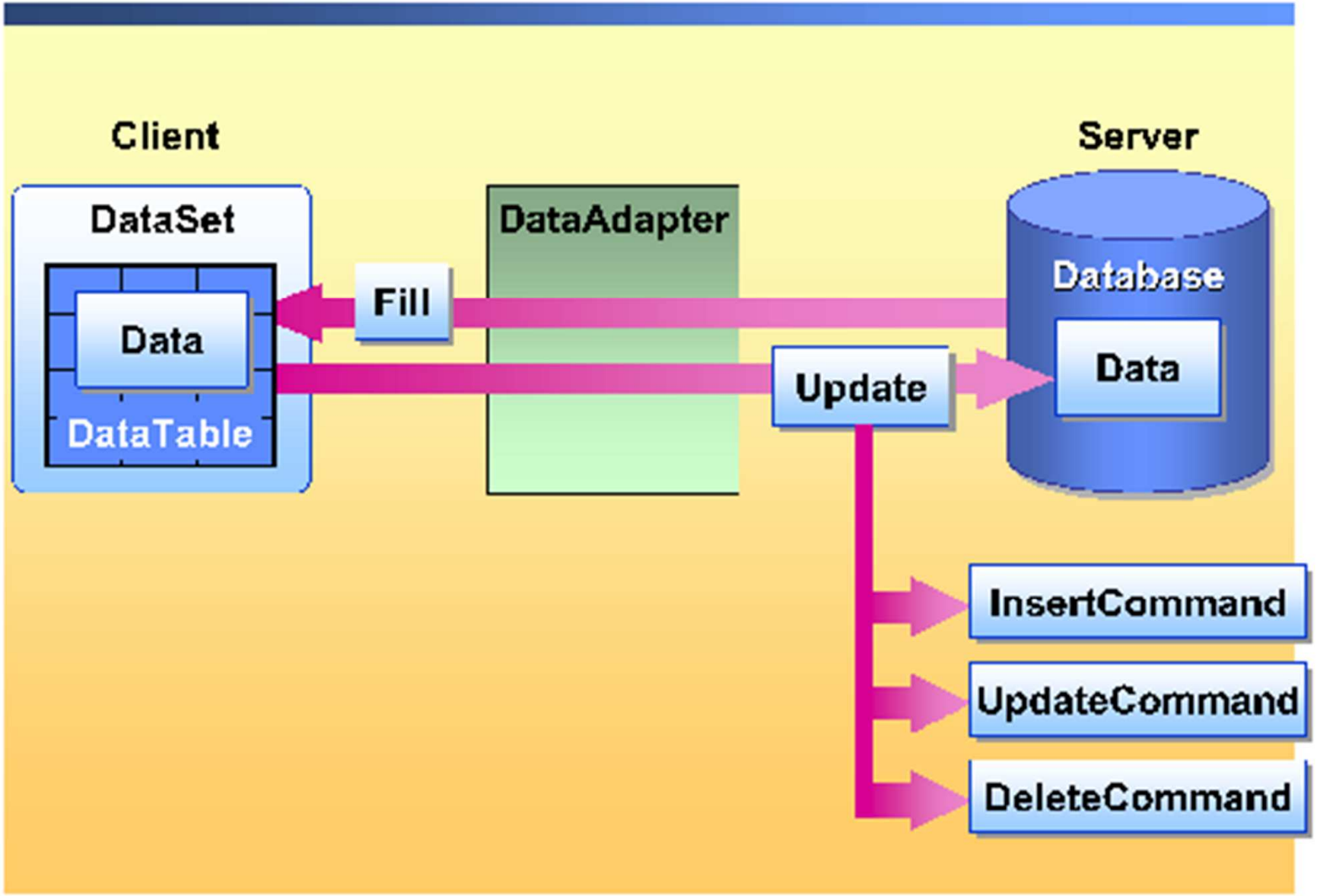
Typed Data Sets

- Strongly typed metaphore
- Objects represent columns, rows, relationsh
- Created from XML schema
- Visual Studio auto generates
- Generate from schema via xsd.exe utility
- More convenient, fewer errors

```
DataSet ds1 = new DataSet();
int id1 = (int)ds1.Tables["Orders"].Rows[0]["EmployeeID"];
// VS
NwindDataSet ds2 = new NwindDataSet();
int id2 = ds2.Orders[0].EmployeeID;
```

DataAdapter Object

- **Provider specific**
- **Used to load a dataset**
- **Use the Fill method**


Client
Server
DataSet
Data
DataTable
DataAdapter
Fill
Update
Database
Data
InsertCommand
UpdateCommand
DeleteCommand

DataAdapter Example

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);Initial Catalog=AutoSales;Integrated Security=True";

SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.CommandText = "SELECT * FROM Vehicle";
cmd.CommandType = System.Data.CommandType.Text;

conn.Open();
IDataAdapter da = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
da.Fill(ds);

Console.WriteLine("Id\tMake\t\t\tModel\t\tColor\tYear\tPrice");

foreach(DataRow row in ds.Tables[0].Rows)
{
    Console.WriteLine("{0}\t{1}\t\t\t{2}\t\t{3}\t{4}\t{5:c}"
        , row["Id"]
        , row["Make"]
        , row["Model"]
        , row["Color"]
        , row["Year"]
        , Convert.ToInt32(row["Price"])
        );
}

conn.Close();
```

Transaction

- **Atomic unit of work**
- **Several steps that must succeed of fail together**
- **Provider specific (sqlTransaction**

```
SqlConnection con = new SqlConnection(System.Configuration.ConfigurationManager.AppSettings["dbConnection"]);
SqlCommand cmd1 = new SqlCommand("UPDATE Savings SET balance = balance - 500");
SqlCommand cmd2 = new SqlCommand("UPDATE Checking SET balance = balance + 500");
SqlTransaction tran = null;
try
{
    con.Open();
    tran = con.BeginTransaction();
    cmd1.Transaction = tran;
    cmd2.Transaction = tran;
    cmd1.ExecuteNonQuery();
    cmd2.ExecuteNonQuery();
    tran.Commit();
}
catch (Exception ex)
{
    tran.Rollback();
    throw ex;
}
finally{ con.Close(); }
```

Best Practices

- **A db connection is a limited resources**
- **Open your database connection as late as possible**
- **Close your database connection as soon as possible**
- **If an object is disposible (implements Idisposible), then always call the Dispose() method**

Dispose Method

- **Releases unmanaged resources**
- **The GC will release memory allocated to the managed object**
- **Use try-finally statement or the using statement**

Using statement

- **Implements the dispose pattern**
- **Recommended whenever creating an object that implements the IDisposable interface**
- **Syntax is:**

```
using( <IDisposable Object> declaration)
{
        …
}
```

- **Same as:**

```
SomeType x;
try{
        x = new SomeType();
        …
}
finally{
        x.Dispose();
}
```

Using Example

```
using (SqlConnection conn = new SqlConnection())
{
    conn.ConnectionString = "Data Source=(local);Initial Catalog=AutoSales;Integrated Security=True";

    using (SqlCommand cmd = new SqlCommand("SELECT * FROM Vehicle",conn))
    {
        IDataAdapter da = new SqlDataAdapter(cmd);
        using (DataSet ds = new DataSet())
        {
            conn.Open();
            da.Fill(ds);

            Console.WriteLine("Id\tMake\t\t\tModel\t\tColor\tYear\tPrice");
            foreach (DataRow row in ds.Tables[0].Rows)
            {
                Console.WriteLine("{0}\t{1}\t\t\t{2}\t\t{3}\t{4}\t{5:c}"
                    , row["Id"], row["Make"], row["Model"]
                    , row["Color"], row["Year"]
                    , Convert.ToInt32(row["Price"]));
            }
            ds.WriteXml("AutoSales.xml");
            conn.Close();
        }
    }
}
```