



## 2. Platforma Microsoft .NET

Maciej Piechówka  
macpi@eti.pg.gda.pl

### Spis treści:

#### 2.1 Microsoft .NET Framework: CLR, podzespoły, biblioteka klas, elementy C#

#### 2.3 Dostęp do danych

- **ADO.NET: tryb dostępu, aktualizacja, procedury składowane, transakcje, integracja z XML**
- LINQ, ADO.NET Entity Framework, ADO.NET Data Services

#### 2.4 Wytwarzanie aplikacji .NET: Podstawy ASP.NET

- Model strony, WebForms,
- Model delegacyjny zdarzeń,
- Kontrolki serwerowe, użytkownika, sprawdzające, wiązanie danych,
- ASP.NET od wewnątrz
- Zarządzanie stanem

#### 2.5 Tworzenie i korzystanie z usług Web (*web services*), użycie protokołów SOAP i UDDI

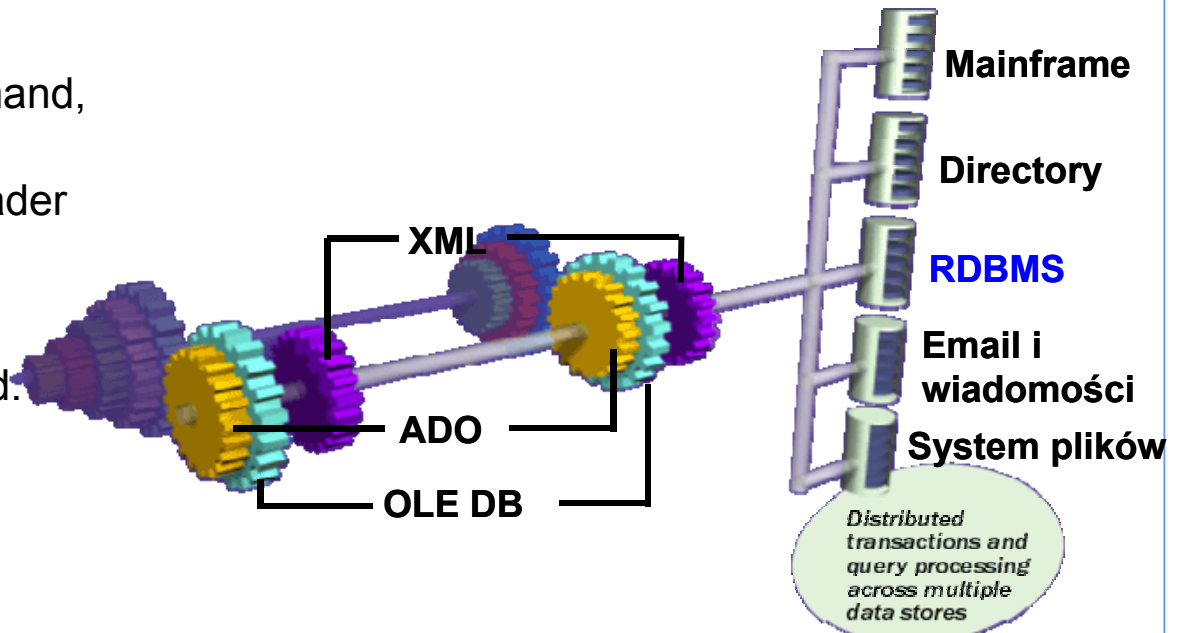
##### Literatura:

- Materiały firmy Microsoft, Piotr Bubacz *ITA-103 Aplikacje Internetowe*, zasoby Internetu...
- Platt D. *Podstawy Microsoft .Net*, RM, 2001
- Pery C. s. *Core C# i .NET*, Helion 2006
- Esposito D. *Tworzenia aplikacji za pomocą ASP.NET oraz ADO.NET*, RM 2002
- Connolly R. *ASP.NET 2.0. Projektowanie aplikacji internetowych*, Helion 2008
- Matulewski J., *C# 3.0 i .NET 3.5. Technologia LINQ*, Helion, 2008



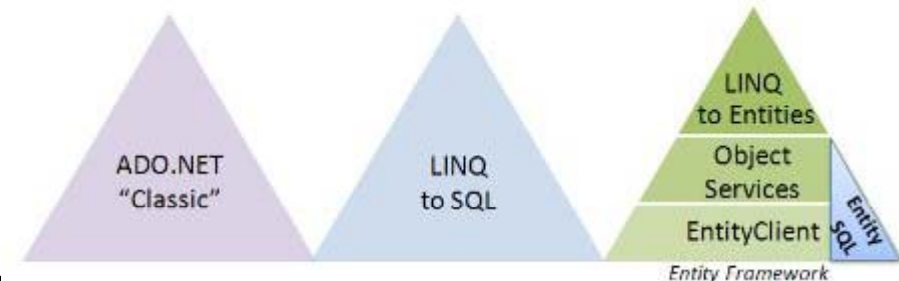
## 2.3 Warstwa dostępu do danych

- Architektura ADO.NET
  - Obiekty DataSet, Connection, Command, DataAdapter i ich współdziałanie
  - Praca w trybie połączonym: DataReader
  - Praca w trybie odłączonym: DataSet
  - Modyfikacje źródła danych
  - Obsługa procedur pamiętanych w b.d.
  - Integracja z XML
  - Transakcje
- LINQ, Entity Framework
- **ADO.NET Data Services**



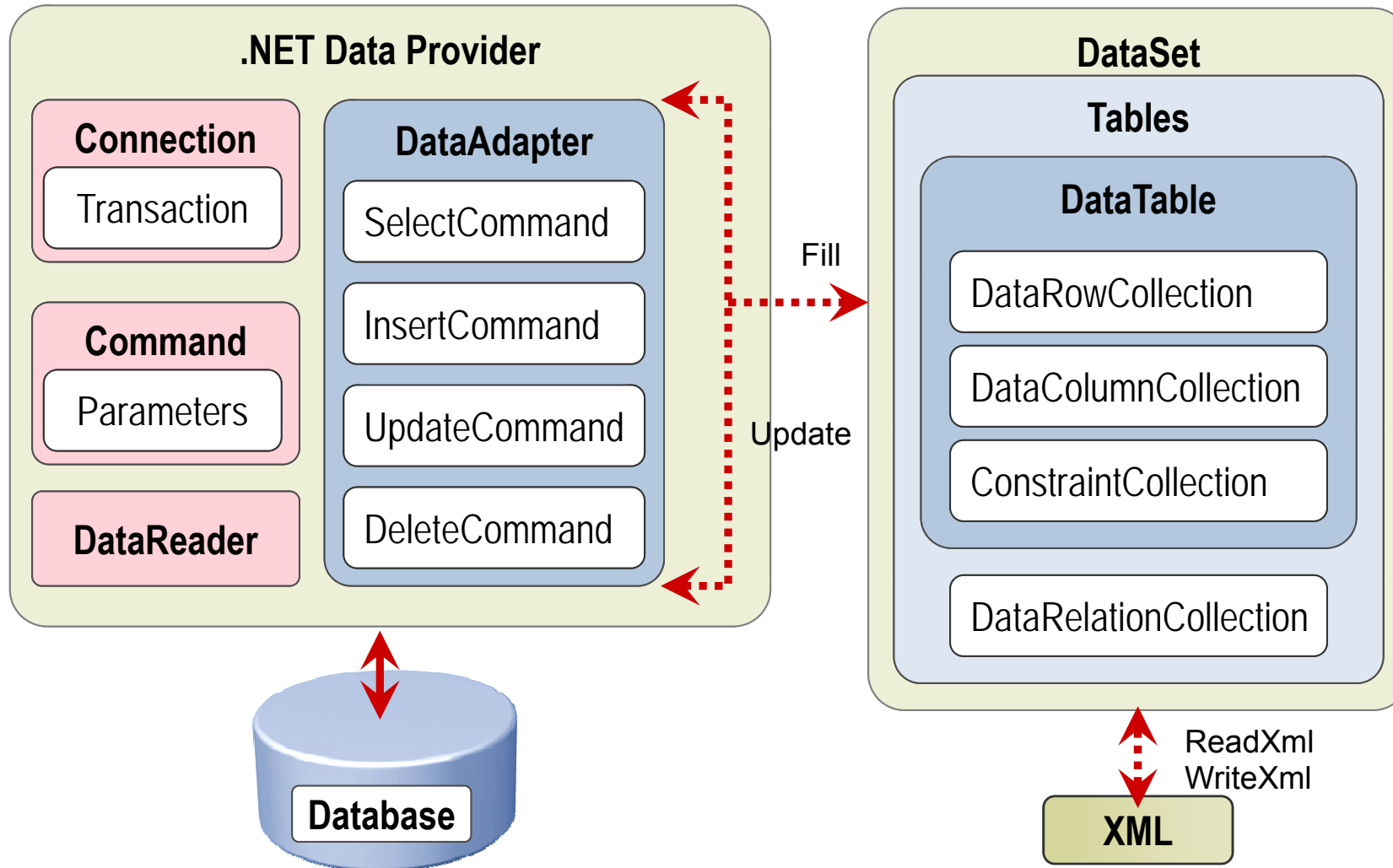
### Problemy

- Modele programistyczne
  - Połączeniowy (*connected*) – oparcie o otwarte połączenia do systemu bazy danych (OLE DB, ADO, ODBC)
  - Bezpołączeniowy, rozłączony (*disconnected*) – pytanie/odpowiedź, bez-stanowy między zapytaniami (HTTP)
- Obsługa stanu





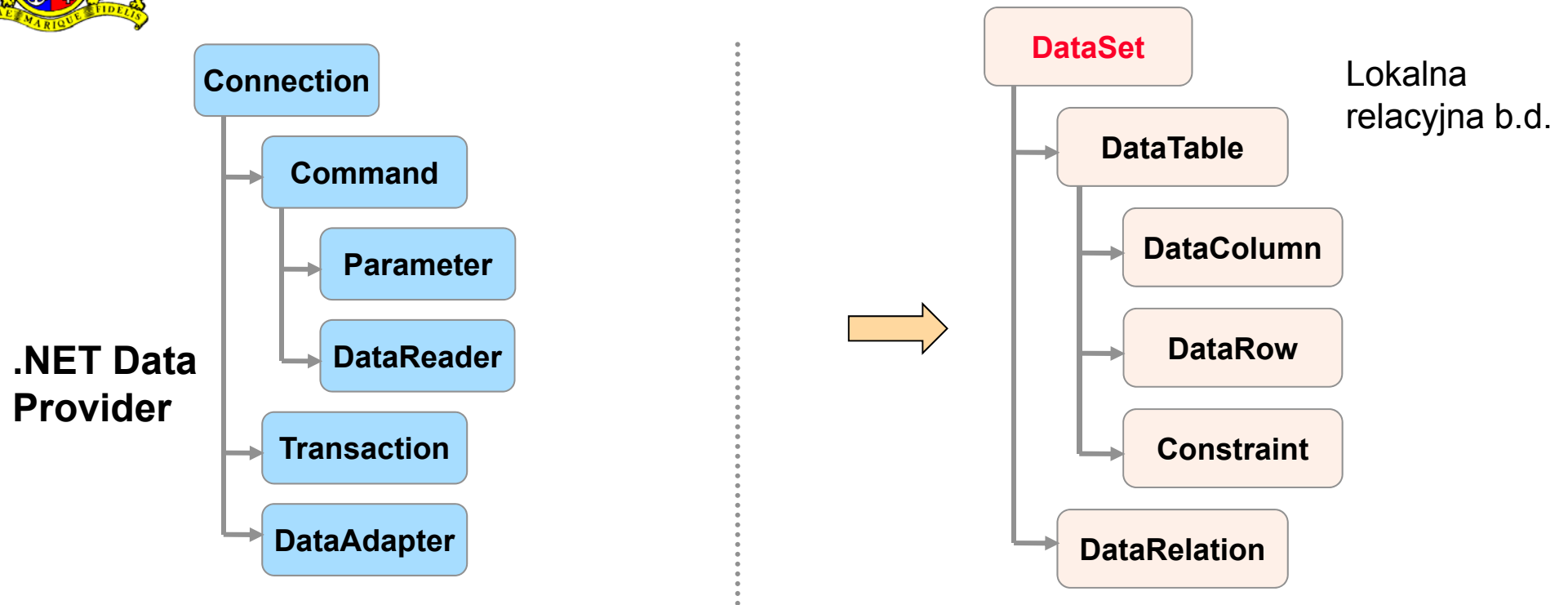
## 2.3.1 Architektura ADO.NET



A **.NET data provider** is a set of classes that you use to connect to a **data source**, and retrieve and update data



# ADO.NET Object Model (2)



Klasa	Opis
<b>Connection</b>	Umożliwia nawiązanie <b>połączenia</b> z określonym źródłem danych
<b>Command</b>	Wywołuje <b>polecenie</b> na źródle danych. Udostępnia kolekcję parametrów ( <b>Parameters</b> ) i zawsze działa w kontekście otwartego połączenia ( <b>Connection</b> )
<b>DataReader</b>	Udostępnia jednokierunkowy (rekord po rekordzie) strumień danych ze źródła, w trybie 'tylko do odczytu'
<b>DataAdapter</b>	Wypełnia <b>DataSet</b> danymi pochodzącymi ze źródła oraz umożliwia aktualizacje danych w źródle na podstawie DataSet-u (kanał łączący obiekt DataSet z dostawcą danych)



# ADO.NET Class hierarchy

- **General interfaces**

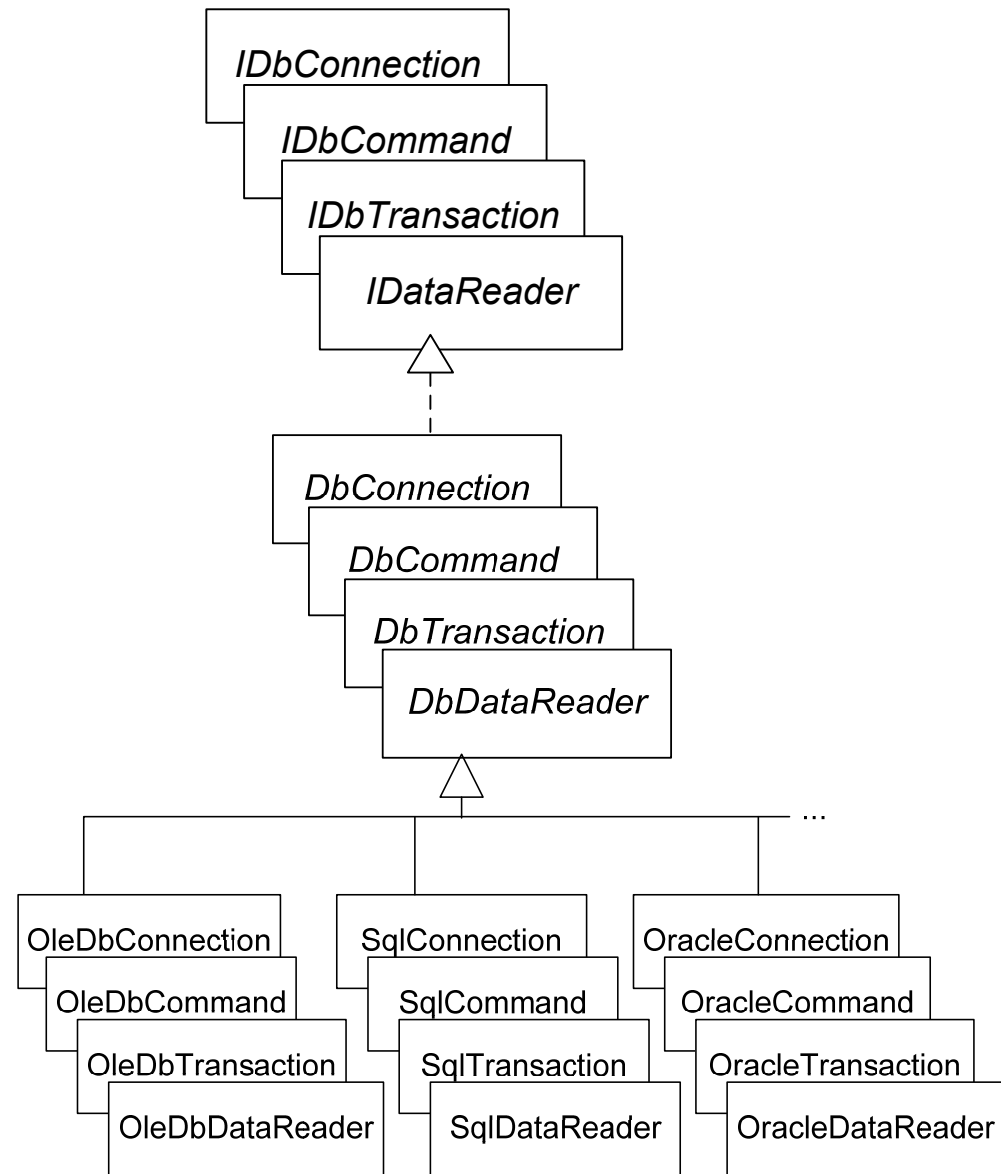
**IDbConnection**  
**IDbCommand**  
**IDbTransaction**  
**IDataReader**

- **Abstract base classes**

**DbConnection**  
**DbCommand**  
**DbTransaction**  
**DbDataReader**

- **Special implementations**

**OleDb:** implementation for **OLEDB**  
**Sql:** implementation for **SQL Server**  
**Oracle:** implementation for **Oracle**  
**Odbc:** implementation for **ODBC**  
**SqlCe:** implementation for **SQL Server CE data base**





# Połączenie: Creating the Connection

A **connection string** defines the parameters required to make a connection to a data source

## • Using SqlConnection

```
string strConn = "data source=localhost; " +  
    "initial catalog=northwind; integrated security=true";  
SqlConnection conn = new SqlConnection(strConn);
```

## • Setting connection string parameters

- **Connection timeout:** dopuszczalny czas uzyskania połączenia
- **Data source:** nazwa instancji SQL Server lub nazwa komputera
- **Initial catalog:** nazwa bazy danych
- **Integrated security;** gdy True połączenie z SQL serwerem na podstawie tożsamości konta procesu ASP.NET
- **User ID:** konto logowania SQL Server
- **Password:**
- ...

<http://www.connectionstrings.com/>



# Połączenie: Creating the Connection

```
public interface IDbConnection
{
    string ConnectionString {get; set;}
    int ConnectionTimeout {get;}
    string Database {get;}
    ConnectionState State {get;}

    IDbTransaction BeginTransaction();
    IDbTransaction BeginTransaction(IsolationLevel il);
    void ChangeDatabase(string db);
    void Close();
    IDbCommand CreateCommand();
    void Open();
}
```

**BeginTransaction** - Begins a database transaction

**ChangeDatabase** - Changes the current database for an open connection. The new database name is passed as string to the method.

**ConnectionTimeout** - The number of seconds to wait while trying to establish a connection to a data source before timing out.

**Database** - Name of the database associated with the current connection.

**State** - Gets the current state of the connection. Returns a ConnectionState enumeration name: Broken, Closed, Connecting, Executing, Fetching, or Open.

**Open, Close** - Opens a connection. Rolls back any pending operations and closes the connection—returning it to a connection pool, if one is used.

**CreateCommand** - Creates and returns a Command object associated with the connection.



# DbProviderFactory

- Writing database-independent programs with *DbProviderFactory*
- *DbProviderFactory* generates set of provider-specific components
- Provider can be configured in an easy way (e.g. in configuration file)

```
DbProviderFactory factory =  
    DbProviderFactories.GetFactory("System.Data.SqlClient");
```

create DbProviderFactory for SqlClient

```
IDbConnection conn = factory.CreateConnection();  
IDbCommand cmd = factory.CreateCommand();  
cmd.Connection = conn;  
IDataParameter param = factory.CreateParameter();
```

create provider specific data access components  
DBConnection  
DbCommand  
DataParameter

**Each data provider registers a ProviderFactory class and a provider string in the *machine.config* file. The available providers can be listed using the static *GetFactoryClasses* method of the *DbProviderFactories* class.**

```
DataTable tb = DbProviderFactories.GetFactoryClasses();  
foreach (DataRow drow in tb.Rows )  
{  
    StringBuilder sb = new StringBuilder("");  
    for (int i=0; i<tb.Columns.Count; i++)  
    {  
        sb.Append((i+1).ToString()).Append(drow[i].ToString());  
        sb.Append("\n");  
    }  
    Console.WriteLine(sb.ToString());  
}
```



# Managing Connection Strings

- use the `< connectionStrings >` section of the configuration file

```
< configuration >
...
  < connectionStrings >
    < add name="Northwind" providerName="System.Data.SqlClient"
      connectionString="server=(local);
      integrated security=SSPI;database=Northwind" / >
  < /connectionStrings >
< /configuration >
```

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



# Handling Errors

- **Connection will not open**
  - Connection string is invalid
  - Server or database not found
  - Login failed
- **DataAdapter cannot create a DataSet**
  - Invalid SQL syntax
  - Invalid table or field name

```
try
{
    SqlConnection conn = new SqlConnection("...");
    SqlDataAdapter da = new SqlDataAdapter("...",conn);
    DataSet ds = new DataSet();
    da.Fill(ds);
}
catch (System.Data.SqlClient.SqlException ex1)
{
    switch(ex1.Number)
    {
        case 17:
            lblErrors.Text = lblErrors.Text + ("invalid server name");
            break;
        ....
        case 18456:
            lblErrors.Text = lblErrors.Text + ("invalid password");
            break;
    }
}
catch (System.Exception ex2)
{
    lblErrors.Text = lblErrors.Text + ("Unexpected exception: " + ex2.Message + ". ");
}
```



## Pula połączeń

- **Connection pooling** is the process of keeping connections active and pooled so that they can be efficiently reused.
- Connection pooling is the ability of SQL Server or an OLE DB data source to reuse connections for a particular user or security context.
- **Connection string parameters for connection pooling**
  - **Connection Lifetime:**
  - **Connection Reset:** Determines whether the database connection is reset when the connection is removed from the pool
  - **Max Pool Size:** The maximum number of connections allowed in the pool
  - **Min Pool Size:** The minimum ...
  - **Pooling:** True/False
  - ...

```
cnNorthwind.ConnectionString = _  
    "Integrated Security=True;" & _  
    "Initial Catalog=Northwind;" & _  
    "Data Source=London;" & _  
    "Pooling=True;" & _  
    "Min Pool Size=5;" & _  
    "Connection Lifetime=120;"
```



# ADO.NET Programming Models

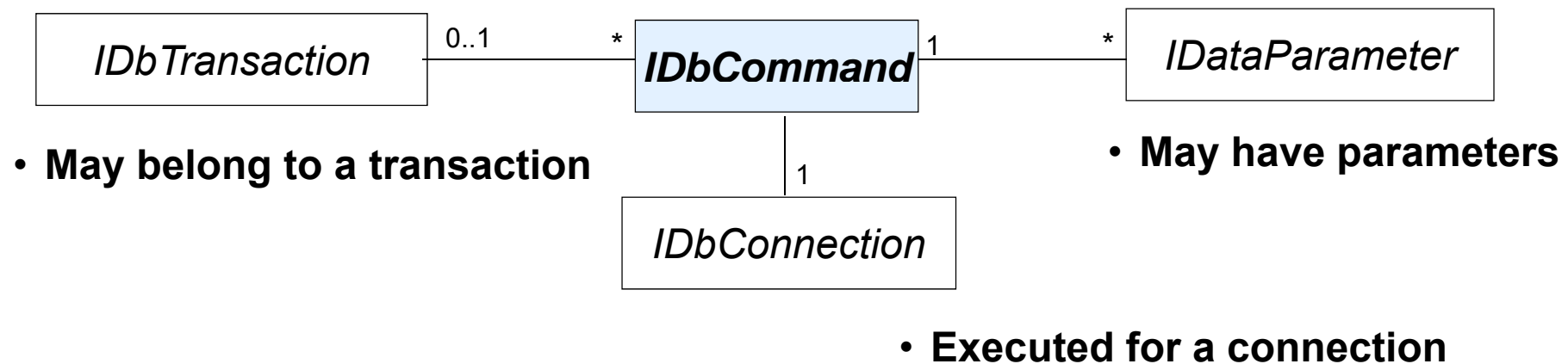
- ADO.NET provides support for two distinctly different programming models
  - **Connected data:** an active connection is maintained between an application's DataReader object and a data source.
    - Use connected **Command** and **DataReader** objects
    - **DataReader** is read-only, forward only
    - Updates performed through **Command** object
  - **Disconnected data:** data is loaded - using a SQL command - from an external source into a memory cache on the client's machine; the result set is manipulated on the local machine; and any updates are passed from data in-memory back to the data source. The connection is **only open** long enough to **read data** from the source and make **updates**.
    - Use **DataSets**
    - **DataAdapters load DataSets** with data and **write DataSet changes back** to the server
    - DataSets are Provider independent
    - DataSets are stored or transmitted as XML
    - DataSet is a RDOM\* for XML

\*RDOM – Relational Document Object Model



## 2.3.2 Model połączeniowy Polecenie: Command Objects

A **command object** is a reference to a SQL statement or stored procedure.  
*IDbCommand* models database commands.



Retrieving and Modifying Data in ADO.NET

<http://msdn.microsoft.com/en-us/library/ms254937.aspx>



# What Is a Command Object?

```
public interface ICommand
{
    string      CommandText      {get; set; }
    int         CommandTimeout   {get; set; }
    CommandType CommandType     {get; set; }
    IDbConnection Connection     {get; set; }
    IDbTransaction Transaction   {get; set; }
    UpdateRowSource UpdatedRowSource {get; set; }
    IDataParameterCollection Parameters {get; }

    void        Cancel ();
    IDataParameter CreateParameter ();
    int         ExecuteNonQuery ();
    IDataReader ExecuteReader ();
    IDataReader ExecuteReader (CommandBehavior cb);
    object      ExecuteScalar ();
    void        Prepare ();
    // Note ExecuteXmlReader (SqlCommand only)
}
```

- **Connection** - referencja do obiektu połączenia
- **CommandType** - typ polecenia
  - **Text** – wyrażenie SQL
  - **StoredProcedure**
- **CommandText** - w zależności od wyboru typu polecenia:
  - **Text** – treść polecenia SQL
  - **StoredProcedure** – nazwa procedury
- **Parameters**
  - **SQL statements and stored procedures can have input and output parameters, and a return value**
  - **Command parameters allow these parameters to be set and retrieved**



# Command Object Methods

- Executing Commands Objects That **Do Not Return Rows**
  - Call **ExecuteNonQuery** method
    - Returns count of rows affected
  - Automate database administration tasks: DDL and DCL statements
    - CREATE, ALTER, DROP, GRANT, DENY, REVOKE
  - **Modify data** in the database: DML Statements
    - INSERT, UPDATE, DELETE
- Executing Command That **Return a Single Value**
  - Call the **ExecuteScalar** method
    - ExecuteScalar returns a value of the type Object, Use CType or a cast, to convert into appropriate type
- Executing Commands That **Return Rows**
  - Call the **ExecuteReader** method
    - Returns a **DataReader**
    - For example, SqlDataReader, OleDbDataReader
  - DataReader
    - Read-only, forward-only, stream of rows
- Executing Command that **Return an XmlReader** object that is used to access the resultset
  - **ExecuteXmlReader**. Available for SQL Server data provider only.



## Przykład - Command Class

```
private void Demo()  
{  
    SqlConnection con = new SqlConnection(  
        "Server=localhost; Database=Pubs; Integrated Security=SSPI" );  
    SqlCommand cmd = new SqlCommand(  
        "SELECT COUNT( * ) FROM Authors", con );  
    con.Open();  
    Console.WriteLine( cmd.ExecuteNonQuery() ); // Writes '23'  
    con.Close(); // Important!  
}
```



## Przykład - Command Class : ExecuteReader

```
rdr = cmd.ExecuteReader(sql, CommandBehavior.SingleResult);
```

**CommandBehavior** enumeration includes the following:

- **SingleRow**. Indicates that the query should return one row. Default behavior is to return multiple result sets.
- **SingleResult**. The query is expected to return a single scalar value.
- **KeyInfo**. Returns column and primary key information. It is used with a data reader's *GetSchema* method to fetch column schema information.
- **SchemaOnly**. Used to retrieve column names for the resultset. Example:

```
dr=cmd.ExecuteReader(CommandBehavior.SchemaOnly);  
string col1= dr.GetName(0); // First column name
```

- **SequentialAccess**. Permits data in the returned row to be accessed sequentially by column. This is used with large binary (BLOB) or text fields.
- **CloseConnection**. Close connection when reader is closed.

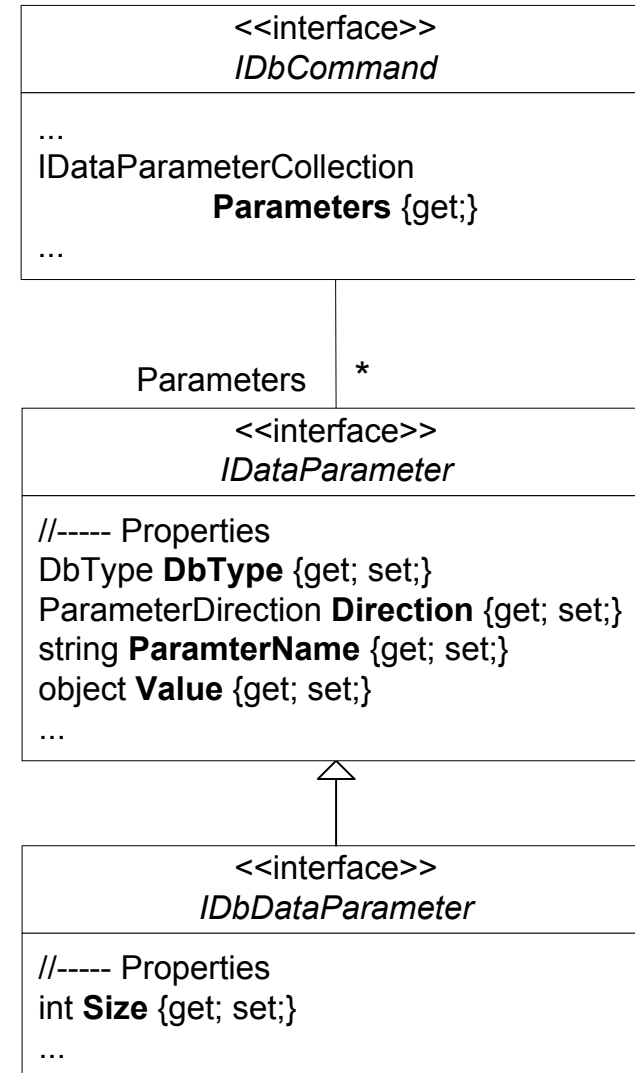
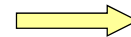


# Zapytania parametryzowane

- Command objects allow input and output parameters

`IDataParameterCollection Parameters {get;}`

- Parameter objects specify
  - Name: name of the parameter
  - Value: value of the parameter
  - DbType: data type of the parameter
  - Direction: direction of the parameter
    - Input
    - Output
    - InputOutput
    - ReturnValue





# Zapytania parametryzowane (2)

## 1. Define SQL command with place holders

**OLEDB: Identification of parameters by position (notation: "?")**

```
OleDbCommand cmd = new OleDbCommand();  
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = ?";
```

**SQL Server: Identification of parameters by name (notation: "@name")**

```
SqlCommand cmd = new SqlCommand();  
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = @ID";
```

## 2. Create and add parameter

```
cmd.Parameters.Add( new OleDbParameter("@ID", OleDbType.BigInt));
```

## 3. Assign values and execute command

```
cmd.Parameters["@ID"].Value = 1234;  
cmd.ExecuteNonQuery();
```



# Tryb połączeniowy: DataReader

**DataReader** - provides a means of reading one or more forward-only streams of result sets obtained by executing a command at a data source

- Forward-only, read-only
- Fast access to data
- Connected to a data source
- Manage the connection yourself
- Manage the data yourself, or bind it to a list-bound control
- Uses fewer server resources

## To use a DataReader:

1. Create and open the database connection
2. Create a **Command** object
3. Create a **DataReader** from the **Command** object  
Call the **ExecuteReader** method
4. Use the **DataReader** object
5. Close the **DataReader** object
6. Close the **Connection** object

Use Try...Catch...Finally error handling

```
public interface IDataReader
{
    int Depth {get;}
    bool IsClosed {get;}
    int RecordsAffected {get;}
    ...
    void Close();
    DataTable GetSchemaTable();
    bool NextResult();
    bool Read();
    ...
}
```

**NextResult** - Advances the data reader to the next result, when reading the results of batch SQL statements

**Read** - Advances the IDataReader to the next record.



# Reading Data from a DataReader - przykład

- **Call Read for each record**
  - Returns false when there are no more records
- **Access fields**
  - Parameter is the ordinal position or name of the field
  - **Get functions give best performance**

```
while (myReader.Read())
{
    str += myReader[1];
    str += myReader["field"];
    str += myReader.GetDateTime(2);
}
```

- **Close the DataReader**
- **Close the connection**

```
// Open Connection and create command
SqlConnection conn = new
SqlConnection("data source=localhost;
initial catalog=pubs; integrated
security=true");

SqlCommand cmdAuthors = new
SqlCommand("select * from Authors", conn);
conn.Open();

// Create DataReader and read data
SqlDataReader dr;
dr = cmdAuthors.ExecuteReader();
while (dr.Read())
{
    lstBuiltNames.Items.Add(dr["au_lname"] + ",
" + dr["au_fname"]);
}

// Close DataReader and Connection
dr.Close();
conn.Close();
```



# Binding a DataReader to a List-Bound Control

- **Create the Control**

```
<asp:DataGrid id="dgAuthors" runat="server" />
```

- **Bind to a DataReader**

```
dgAuthors.DataSource = dr;  
dgAuthors.DataBind();
```

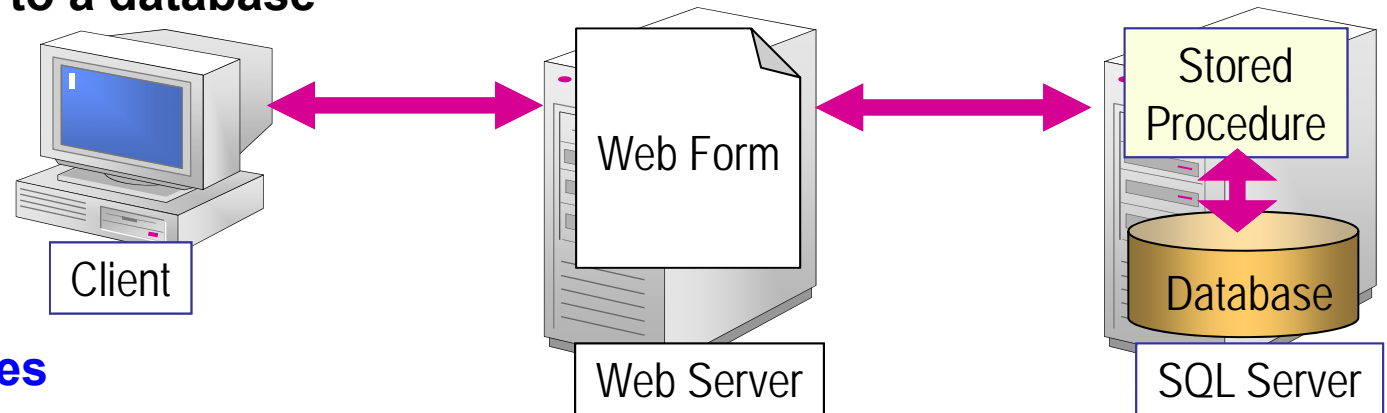
The screenshot shows a web browser window with the address bar containing `http://localhost/2063/module3/grid.aspx`. The page displays a table with the following data:

au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	True
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	True
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	True
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	True
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	True
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	True
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	True



# Procedury składowane (Stored Procedures)

- A common data procedures that can be called by many Web applications
- Programmatic access to a database
  - Return records
  - Return value
  - Perform action



## Calling Stored Procedures

- Identify the stored procedure
- Set up the **SelectCommand** property of the DataAdapter

```
SqlDataAdapter daCategory = new SqlDataAdapter();  
daCategory.SelectCommand = new SqlCommand();  
daCategory.SelectCommand.Connection = conn;  
daCategory.SelectCommand.CommandText = "ProductCategoryList";  
daCategory.SelectCommand.CommandType = CommandType.StoredProcedure;
```

- Run the stored procedure and store returned records

```
daCategory.Fill(ds, "Categories");
```



## Passing Input Parameters

- **Create parameter, set direction and value, add to the Parameters collection**

```
SqlParameter param = new SqlParameter  
    ("@Beginning_Date", SqlDbType.DateTime);  
param.Direction = ParameterDirection.Input;  
param.Value = Convert.ToDateTime(txtStartDate.Text);  
da.SelectCommand.Parameters.Add(param);
```

- **Run stored procedure and store returned records**

```
ds = New DataSet();  
da.Fill(ds, "Products");
```



# Using Output and Return Parameters

- **Create parameter, set direction, add to the Parameters collection**

```
param = new SqlParameter("@ItemCount", SqlDbType.Int);  
param.Direction = ParameterDirection.Output;  
da.SelectCommand.Parameters.Add(param);
```

- **Run stored procedure and store returned records**

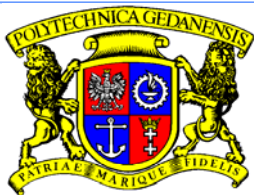
```
ds = new DataSet();  
da.Fill(ds);
```

- **Read output parameters**

```
iTotal = da.Parameters("@ItemCount").Value;
```

**How to get the return value from a stored procedure**

```
iRet = da.Parameters("@RETURN_VALUE").Value;
```



# Transakcje

## Modele transakcji

- RDBMS
- local transactions:
  - transactions for one connection
  - provided by ADO.NET
- distributed transactions:
  - transactions for several connections
  - usage of *Microsoft Distributed Transaction Component (MSDTC)*
  - namespace *System.Transaction*

[Transactions and Concurrency \(ADO.NET\)](http://msdn.microsoft.com/en-us/library/777e5ebh.aspx)  
<http://msdn.microsoft.com/en-us/library/777e5ebh.aspx>



# Obsługa transakcji

- **A transaction** is a set of related tasks that either succeed or fail as a unit
- **ADO.NET** supports transactions  
Commands are assigned to transactions  
Execution of commands are
  - committed with Commit
  - aborted with Rollback

## SZBD

- **SQL transaction statements**
  - BEGIN TRANS, COMMIT TRANS, ROLLBACK TRANS
- **Code example**

```
BEGIN TRANS
```

```
DECLARE @orderDetailsError int, @productError int
```

```
DELETE FROM "Order Details" WHERE ProductID=42
```

```
SELECT @orderDetailsError = @@ERROR
```

```
DELETE FROM Products WHERE ProductID=42
```

```
SELECT @productError = @@ERROR
```

```
IF @orderDetailsError = 0 AND @productError = 0
```

```
    COMMIT TRANS
```

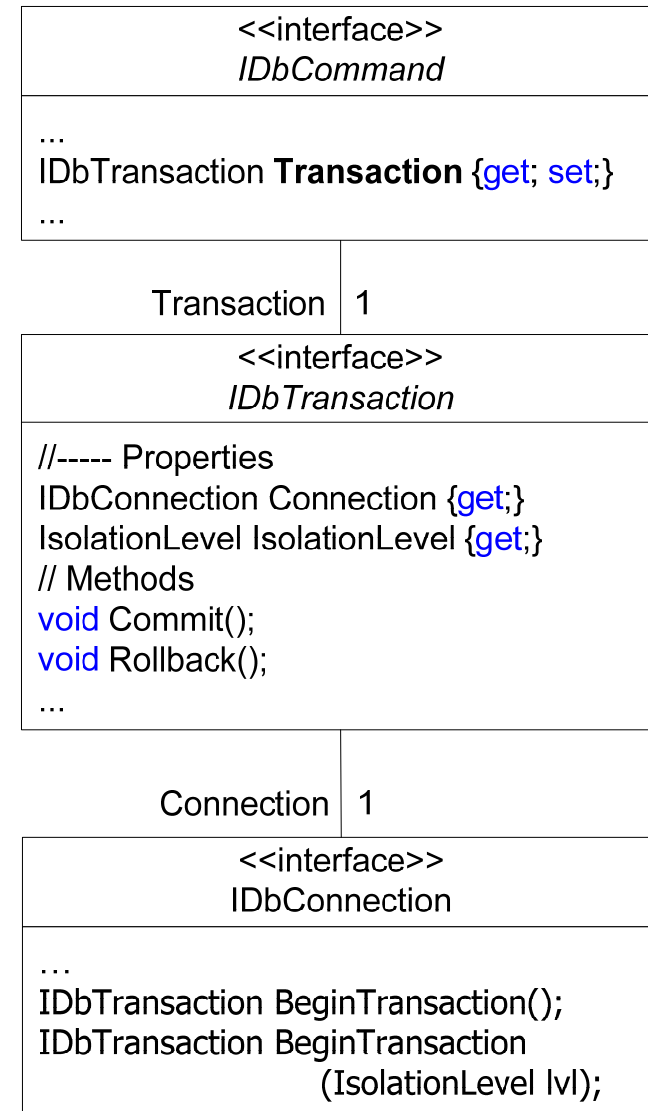
```
ELSE
```

```
    ROLLBACK TRANS
```



# Local Transactions with ADO.NET

- ADO.NET supports transactions
  - Connection objects support transactions with a *BeginTransaction* method that creates a transaction object .
  - The transaction object in turn supports methods that allow you to commit or roll back the transactions
- Commands can be executed within transactions
- Execution of commands are
  - committed with *Commit*
  - aborted with *Rollback*





# Obsługa transakcji (1) - przykład

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    // Start a local transaction.
    SqlTransaction sqlTran = connection.BeginTransaction(); Define connection and create Transaction object
    // Enlist a command in the current transaction.
    SqlCommand command = connection.CreateCommand(); Create Command object, assign it to Transaction object, and execute it
    command.Transaction = sqlTran;
    try
    { // Execute two separate commands.
        command.CommandText =
            "INSERT INTO Production.ScrapReason(Name) VALUES('Wrong size')";
        command.ExecuteNonQuery();
        command.CommandText =
            "INSERT INTO Production.ScrapReason(Name) VALUES('Wrong color')";
        command.ExecuteNonQuery();
        // Commit the transaction.
        sqlTran.Commit();
        Console.WriteLine("Both records were written to database."); Commit or abort transaction
    }
    catch (Exception ex)
    { // Handle the exception if the transaction fails to commit.
        Console.WriteLine(ex.Message);
        try
        { // Attempt to roll back the transaction.
            sqlTran.Rollback();
        }
    }
}
```



# Distributed Transactions

- Transaction over several connection and several data bases
- Usage of *Microsoft Distributed Transaction Component (MSDTC)*
  - MSDTC must be installed and started
- Namespace *System.Transaction*
- ADO.NET 2.0 introduced support for enlisting in a distributed transaction using the *EnlistTransaction* method, which enlists a connection in a Transaction instance.

## Implicit Model with *TransactionScope*

- Creating and managing of transaction objects automatically using transaction scope (class *TransactionScope*)

### Approach:

- code block is marked to participate in transaction
- all connections opened within block run within same transaction
- finalize transaction with *Complete*
- leaving block without *Complete* means rollback



# Example *TransactionScope*

*System.Transaction*

Create *TransactionScope* object for a local block (using statement)

```
//---- same connections as in example with CommitableTransaction  
try {  
    using (TransactionScope transScope =  
        new TransactionScope(TransactionScopeOption.RequiresNew)) {
```

Open connection and execute commands ; all connection opened in block are assigned to same transaction

```
        con1.Open();  
        con2.Open();  
        IDbCommand cmd1 = con1.CreateCommand();  
        cmd1.CommandText = "UPDATE Employees SET Extension=1234 WHERE LastName = 'King'";  
        cmd1.ExecuteNonQuery();  
        IDbCommand cmd2 = con2.CreateCommand();  
        cmd2.CommandText = "UPDATE Contact SET Phone=1234 WHERE Name = 'King'";  
        cmd2.ExecuteNonQuery();
```

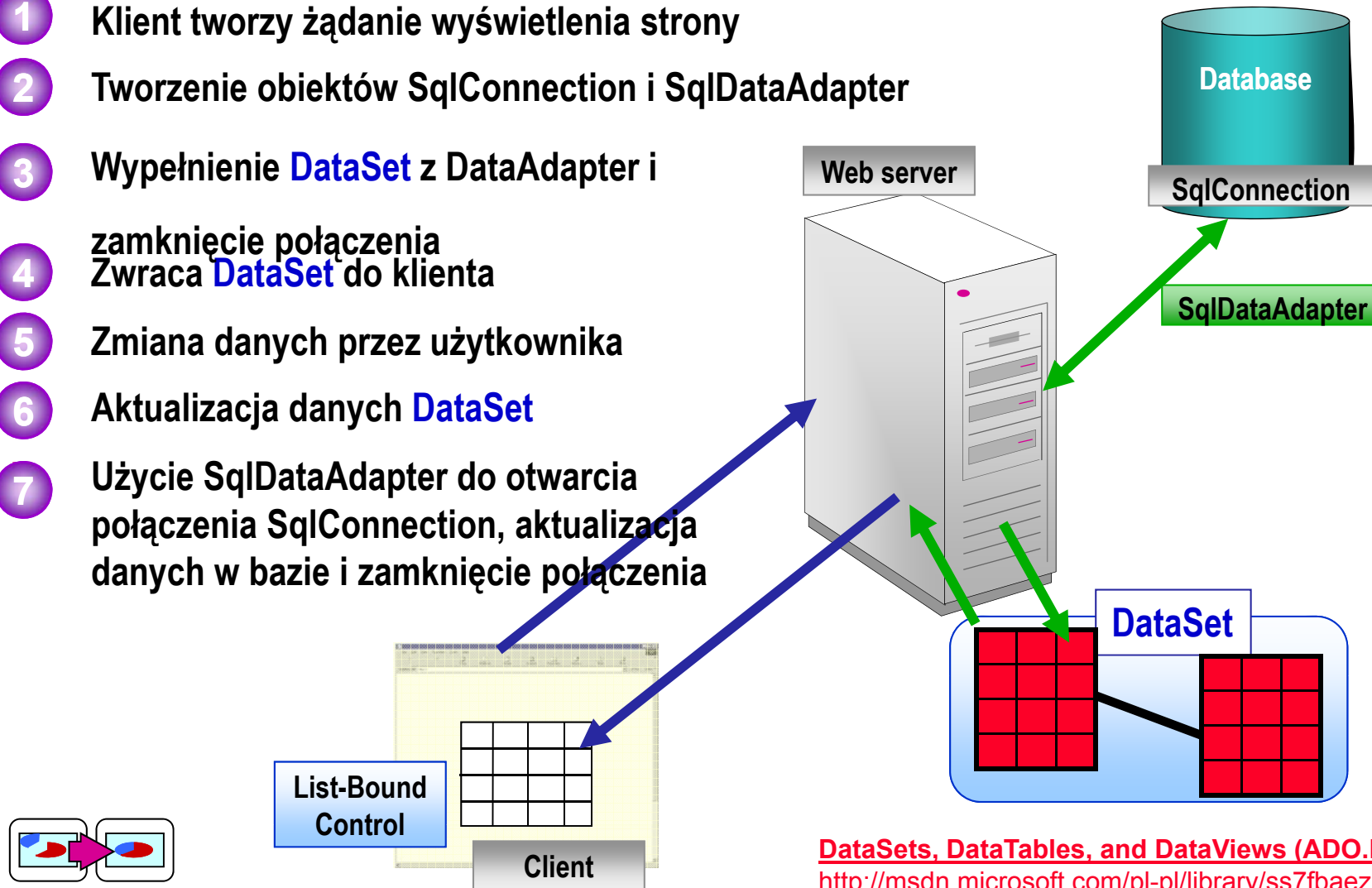
When *Complete* is executed, changes are committed, otherwise discarded

```
            transScope.Complete();  
        }  
    }  
catch (Exception e) { ... }
```



## 2.3.3 Tryb bezpołączeniowy: use DataSet

- 1 Klient tworzy żądanie wyświetlenia strony
- 2 Tworzenie obiektów SqlConnection i SqlDataAdapter
- 3 Wypełnienie DataSet z DataAdapter i
- 4 zamknięcie połączenia  
Zwraca DataSet do klienta
- 5 Zmiana danych przez użytkownika
- 6 Aktualizacja danych DataSet
- 7 Użycie SqlDataAdapter do otwarcia połączenia SqlConnection, aktualizacja danych w bazie i zamknięcie połączenia

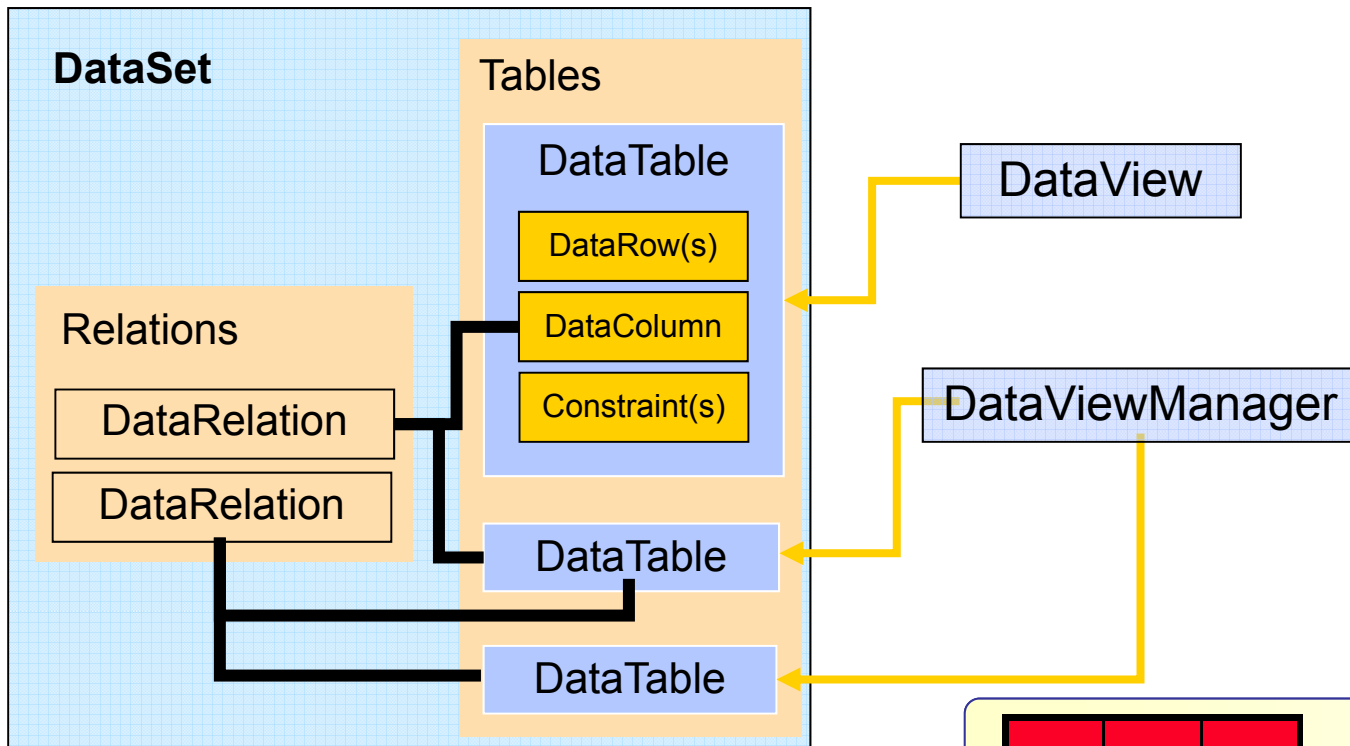


[DataSets, DataTables, and DataViews \(ADO.NET\)](http://msdn.microsoft.com/pl-pl/library/ss7fbaez(en-us).aspx)  
[http://msdn.microsoft.com/pl-pl/library/ss7fbaez\(en-us\).aspx](http://msdn.microsoft.com/pl-pl/library/ss7fbaez(en-us).aspx)

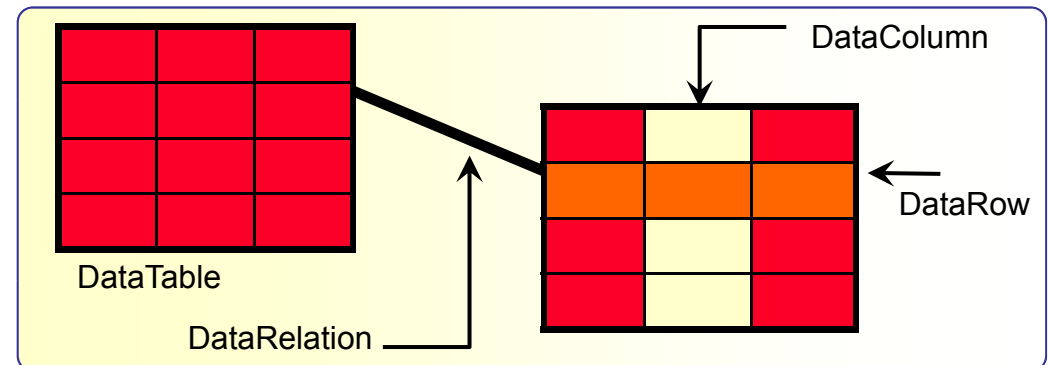


# What Is a DataSet?

**DataSet** - main memory data base (relational structure, object oriented interface)



- Datasets can include multiple DataTables
- Relationships between tables are represented using DataRelations
- Constraints enforce primary and foreign keys
- Use the DataRow and DataColumn to access values in Tables

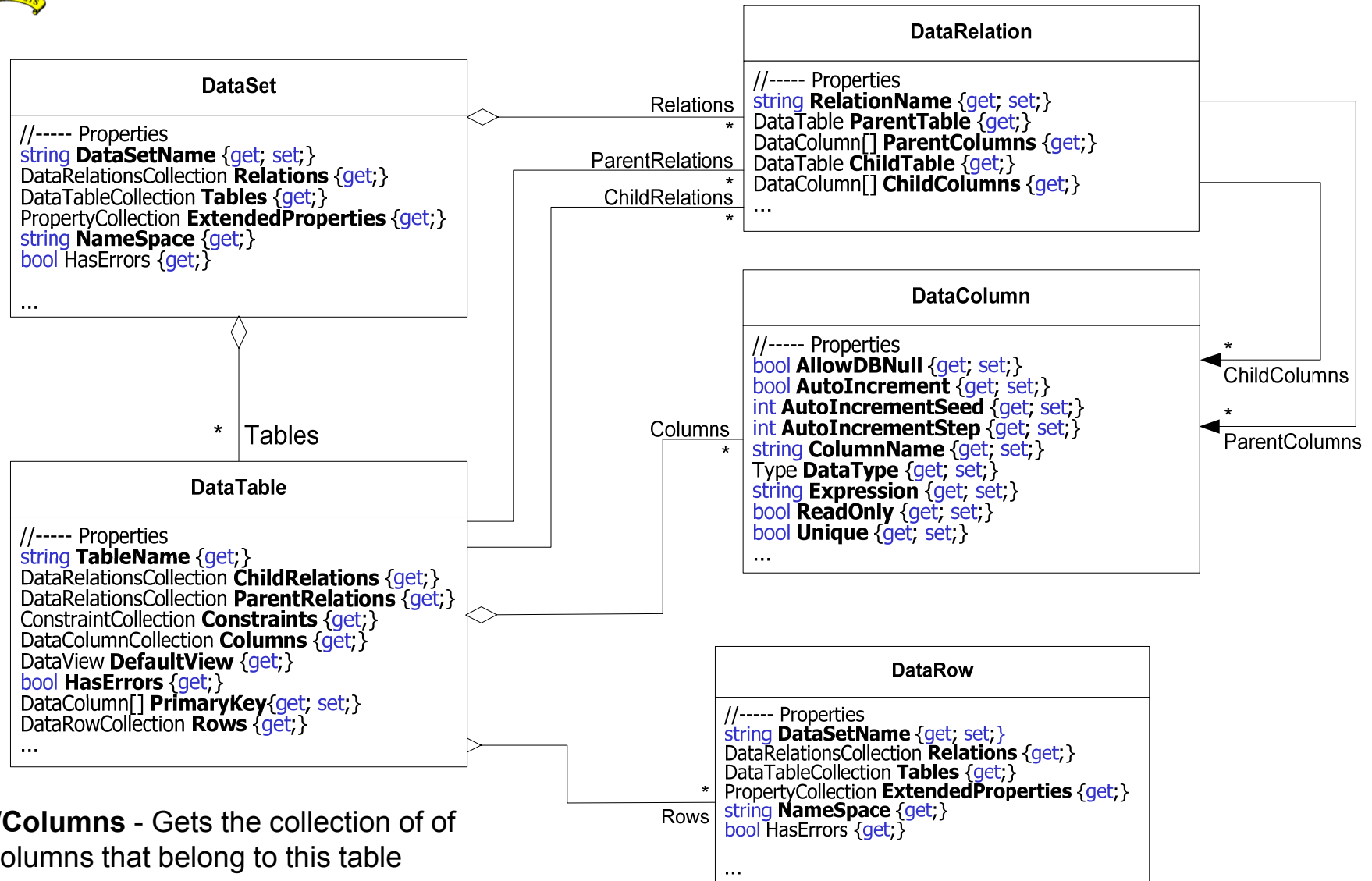


**Using DataSets in**

[http://msdn.microsoft.com/en-us/library/ss7fbaez\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/ss7fbaez(VS.80).aspx) ADO.NET



# Diagram klasy DataSet



**Rows/Columns** - Gets the collection of rows/columns that belong to this table

**ChildRelations** - Gets the collection of child relations for this DataTable



# Klasa DataSet

- **DataSet** consists of
    - collection of DataTables
    - collection of DataRelations
  - **DataTables** consists of
    - collection of DataColumnColumns  
(= **schema definition**)
    - collection of DataTableRows  
(= **data**)
    - DefaultView (DataTableView)
  - **DataRelations**
    - associate two DataTable objects
    - define ParentTable and ParentColumns and ChildTable and ChildColumns
- To access**
- an individual DataTable use
    - DataSet.Tables[0] by index
    - DataSet.Tables[“tablename”] by table name
  - an individual DataColumn use
    - DataTable.Columns[0] by index
    - DataTable.Columns[“columnname”] by columnname
  - an individual DataRow use
    - DataTable.Rows[0] by index
  - an individual field as object use
    - DataRow[0] by index
    - DataRow[“columnname”] by column name



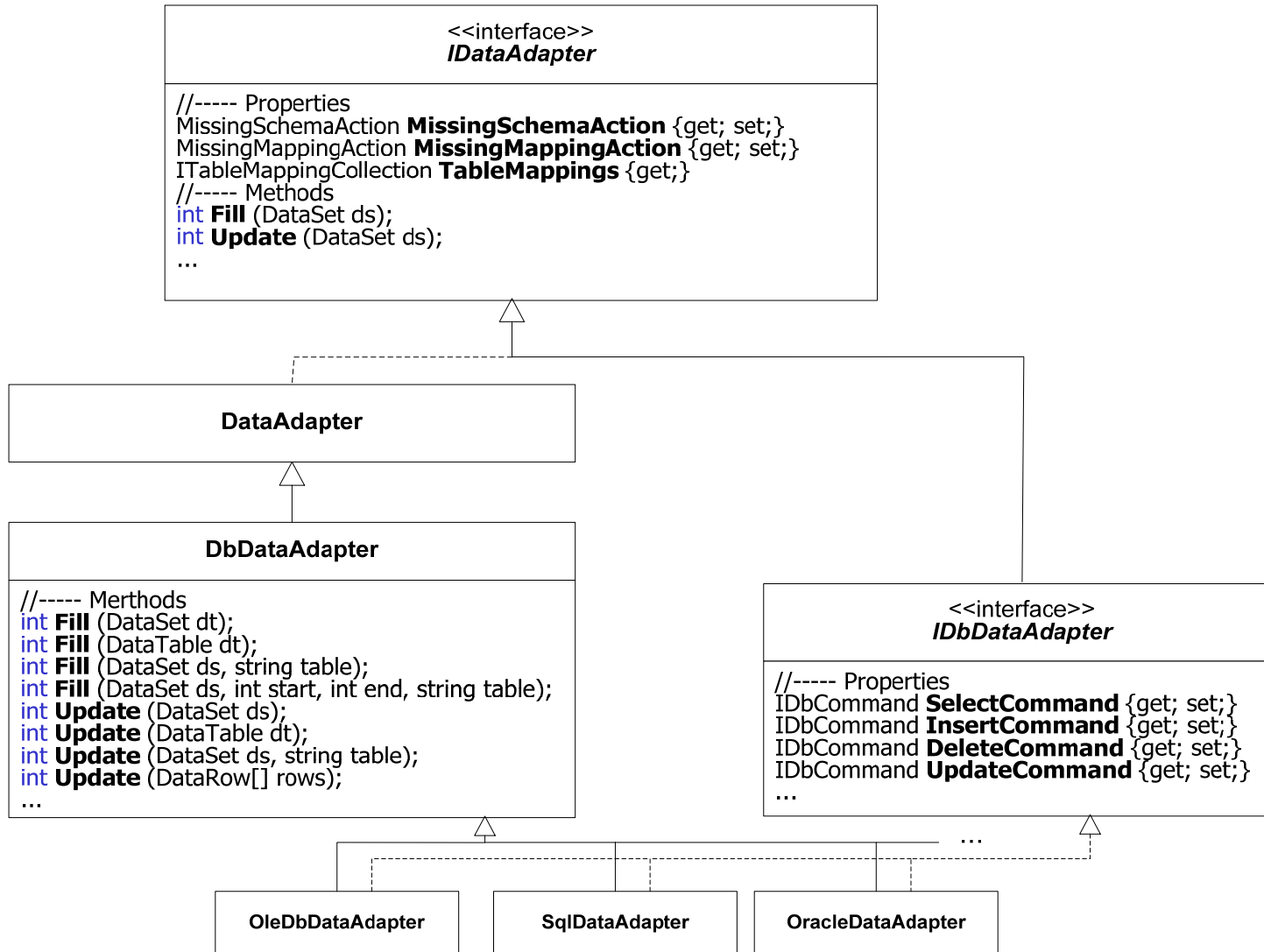
## Wypełnianie DataSet: *DataAdapter*

A *DataAdapter* object serves as a **bridge** between a *DataSet* and a data source for retrieving and saving data. The *DataAdapter* class represents a set of database commands and a database connection that you use to **fill** a *DataSet* and **update** the data source. Each *DataAdapter* exchanges data between a single *DataTable* object in a *DataSet* and a single result set from a SQL statement or stored procedure

- *DataAdapter* **properties** refers to a **command object**:
  - *SelectCommand* - that retrieves rows from the data source
  - *InsertCommand* - that writes inserted rows from the *DataSet* into the data source
  - *UpdateCommand* - that writes modified rows from the *DataSet* into the data source
  - *DeleteCommand* - that deletes rows in the data source
- **Methods used by *DataAdapters***
  - **Fill** - to add or refresh rows from a data source and place them in a *DataSet* table (uses the SELECT statement)
  - **Update** - to transmit changes to a *DataSet* table to the corresponding data source (uses INSERT, UPDATE, or DELETE command for each specified row in a *DataSet DataTable*)



# DataAdapter Class Diagram





## Creating a DataSet (1)

- DataAdapters have a *SelectCommand* property
  - Specifies SQL command text or stored procedure name to be used to obtain result set
  - Assign Command object or use constructor argument

```
private void Demo()  
{  
    SqlDataAdapter da = new SqlDataAdapter(  
        "SELECT City FROM Authors",  
        "Server=localhost; Database=Pubs; Integrated Security=SSPI" );  
    DataSet ds = new DataSet();  
    da.Fill( ds, "Authors" ); // Opens and closes a connection  
    foreach ( DataRow dr in ds.Tables[ "Authors" ].Rows )  
        Console.WriteLine( dr[ "City" ] ); // Writes list of cities  
}
```



## Using a DataView

- A **DataView** can be customized to present a subset of data from a **DataTable**
- The **DefaultView** property returns the default **DataView** of the table

```
DataView dv = ds.Tables["Authors"].DefaultView;
```

- **Setting up a different view of a DataSet (by using filter expression)**

```
DataView dv = new DataView(ds.Tables["Authors"]);  
dv.RowFilter = "state = 'CA'";
```

### **DataTables** have a **Select** method

- Gets an array of **DataRow** objects that match the filter in the order of the sort, and that match the specified state

#### Three optional parameters

- **Filter expression**, for example, "City='London'"
- **Sort**, for example, "CompanyName ASC"
- **DataViewRowState**, for example, Deleted



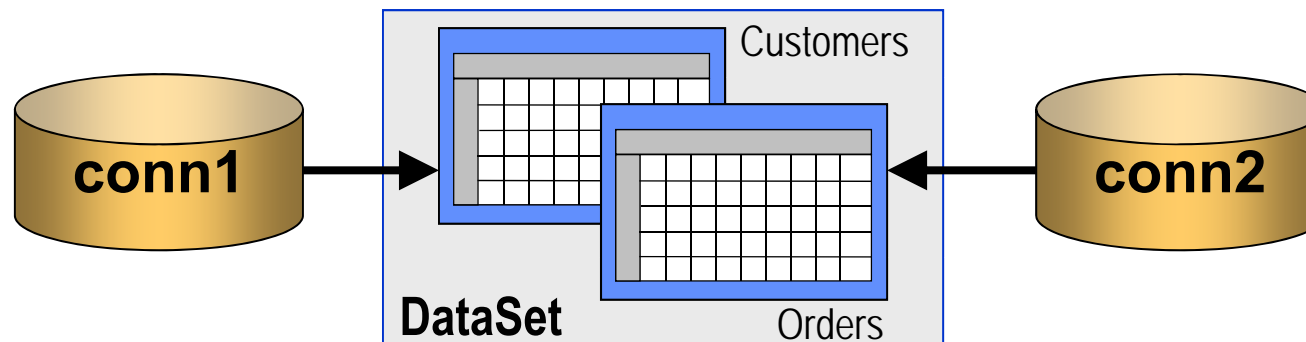
# Storing Multiple Tables

- Add the first table

```
daCustomers = New SqlDataAdapter _  
    ("select * from Customers", conn1)  
daCustomers.Fill(ds, "Customers")
```

- Add the subsequent table(s)

```
daOrders = New SqlDataAdapter _  
    ("select * from Orders", conn2)  
daOrders.Fill(ds, "Orders")
```





# Creating Relationships

- **Identify parent column**

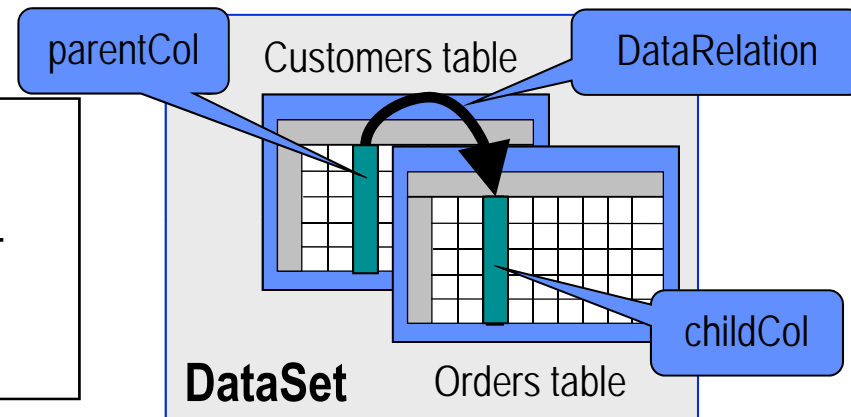
```
parentCol = ds.Tables["Customers"].Columns["CustomerID"]
```

- **Identify child column**

```
childCol = ds.Tables["Orders"].Columns["CustomerID"]
```

- **Create DataRelation**

```
dr = New DataRelation _  
    („CustOrders", parentCol, _  
    childCol)  
ds.DataRelations.Add(dr)
```



A DataRelation object defines a navigational relationship, NOT a constraint relationship



## 2.3.3.1 DataSet: Modifying Data in a DataTable

Praca w trybie odłączonym:

### Modifying Data in a Table

- The **BeginEdit** method of DataRow class
  - Disables the raising of events and exceptions
- **EndEdit** and **CancelEdit** methods of DataRow class
  - Enable the raising of events and exceptions

```
drEmployee DataRow =  
dtEmployees.Rows(3)  
drEmployee.BeginEdit()  
drEmployee("FirstName") = "John"  
drEmployee("LastName") = "Smith"  
drEmployee.EndEdit()
```

### Insert a New Row

- Creating a new row  
`DataRow workRow = workTable.NewRow();`
- Filling the new row  
`workRow[0] = „1”;`  
`workRow["CustLName"] = "Smi th";`
- Appending the row to a DataTable  
`workTable.Rows.Add(workRow);`
- Creating, filling, and appending a row simultaneously  
`workTable.Rows.Add(new Object[] {1, "Smi th"});`

[Manipulating Data in a DataTable](http://msdn.microsoft.com/en-us/library/tzwewss0(VS.80).aspx)

[http://msdn.microsoft.com/en-us/library/tzwewss0\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/tzwewss0(VS.80).aspx)



# Modifying Data in a DataTable: Adding & Deleting Rows

## How to Delete a Record

- The ***Remove*** method of the DataRowCollection class
  - Completely removes the row from the collection
  - Example:  
`dtEmployees.Rows.Remove(drEmployee)`
- The ***Delete*** method of the DataRow class
  - Marks the row as deleted
  - Hidden, but still accessible if necessary
  - Example:  
`drEmployee.Delete`



## Modifying Data in a DataTable: Editing Data

```
DataTable workTable = new DataTable();
workTable.Columns.Add("LastName", typeof(String));

workTable.ColumnChanged += new DataColumnChangeEventHandler(OnColumnChanged);
DataRow workRow = workTable.NewRow();
workRow[0] = "Smith";
workTable.Rows.Add(workRow);

workRow.BeginEdit();
// Causes the ColumnChanged event to write a message and cancel the edit.
workRow[0] = "";
workRow.EndEdit();

// Displays "Smith, New".
Console.WriteLine("{0}, {1}", workRow[0], workRow.RowState);

protected static void OnColumnChanged(
    Object sender, DataColumnChangeEventArgs args)
{
    if (args.Column.ColumnName == "LastName")
        if (args.ProposedValue.ToString() == "")
        {
            Console.WriteLine("Last Name cannot be blank. Edit canceled.");
            args.Row.CancelEdit();
        }
}
```

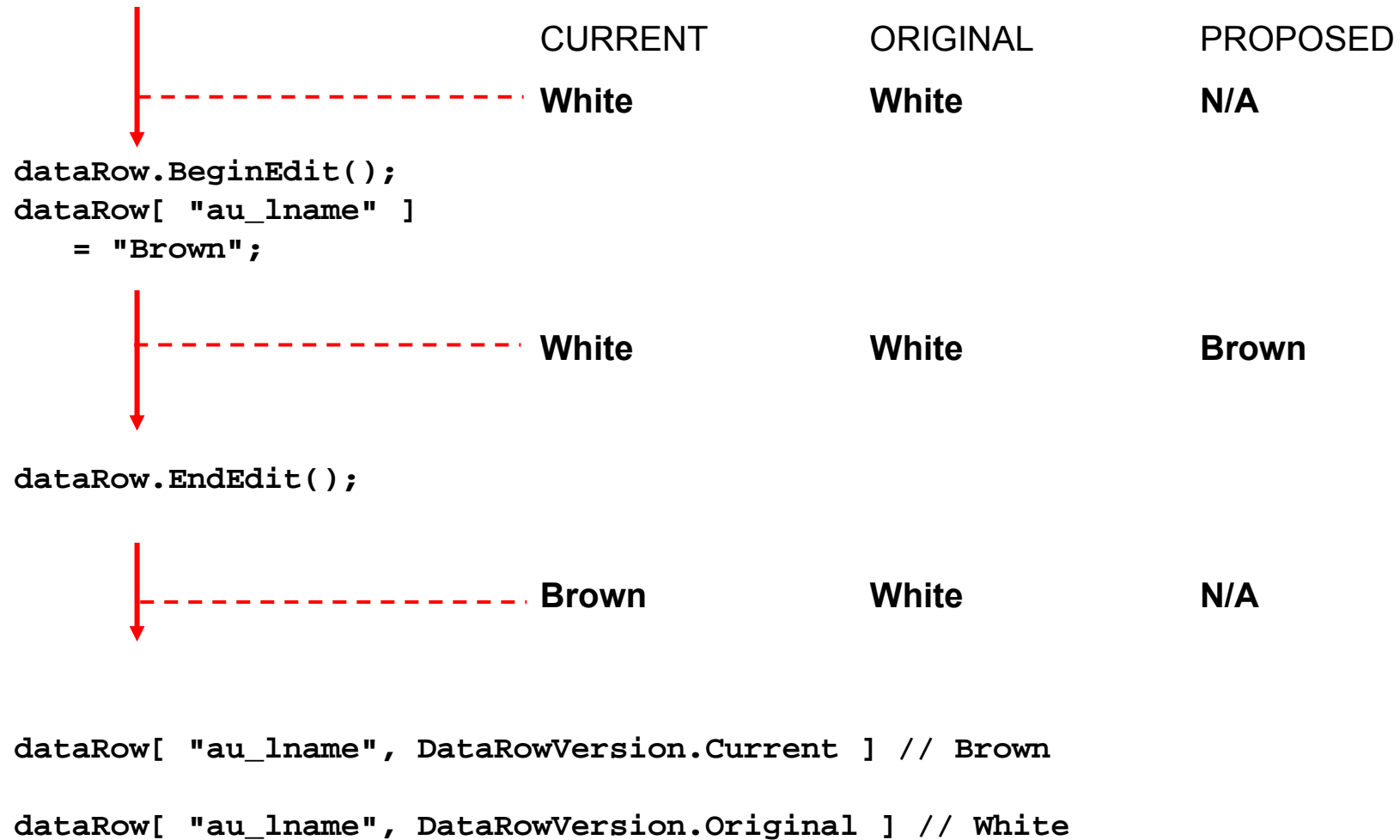


# Tracking DataSet Changes

- A DataRow can hold multiple *versions of a value* for each column:
  - DataRowVersion.**Current**
    - The current value of the column
  - DataRowVersion.**Original**
    - The value of the column before any changes were made
  - DataRowVersion.**Proposed**
    - The value of the column during a *BeginEdit / EndEdit* cycle
  - DataRowVersion.**Default**
    - The default value for the row's *state* (e.g. Original for deleted rows as deleted rows do not have Current values)



# Row Version Processing



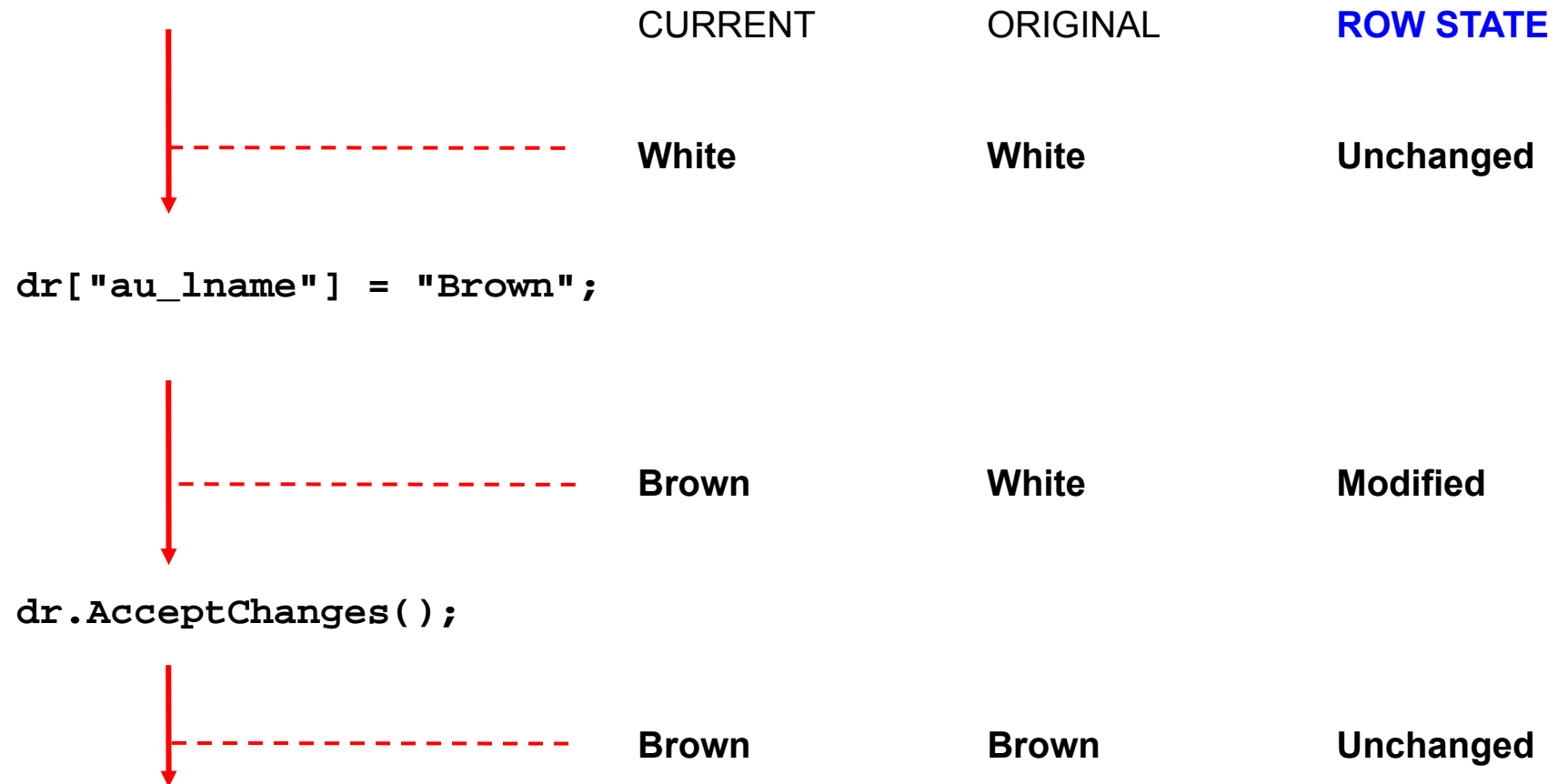


## Row State

- Each row has a *RowState* property
  - Added, Deleted, Detached, Modified, Unchanged
- ***AcceptChanges* method** finalises changes
  - RowState is set to Unchanged
  - Current row version copied to Original row version
- ***RejectChanges* method** restores original values
  - RowState is set to Unchanged
  - Original row version copied to Current row version



# Row State Processing





## 2.3.3.2 Modyfikacja źródła danych

- A *SqlDataAdapter* or *OleDbDataAdapter* object has command properties that are themselves command objects that you can use to **modify data at the data source**
  - *InsertCommand*, *UpdateCommand*, *DeleteCommand*
- Updates are written to a database using the *DataAdapter*'s **Update** method
  - *DataAdapter* looks at the *RowState* for each row
  - Performs appropriate action (insert, update or delete) according to the row's state
    - Uses the *Command* objects assigned to its *InsertCommand*, *UpdateCommand* and *DeleteCommand* properties

### DataRows in DataTable

### DataAdapter Action

RowState = Modified	Use UPDATE command
RowState = Unchanged	Ignore
RowState = Added	Use INSERT command
RowState = Modified	Use UPDATE command
RowState = Deleted	Use DELETE command



# Obsługa współbieżnego dostępu do danych

- **Disconnected applications use optimistic concurrency**
  - Releases database locks between data operations
- **Data conflicts can occur when you update the database**
  - Another application or service might have already changed the data
    - Deleting a previously deleted row
    - Changing a previously changed column

## How to Detect Conflicts

- The *Data Adapter Configuration Wizard* can generate SQL statements to detect conflicts
- When you update the database:
  - Data modification commands **compare the current data** in the database against your **original values** (where ...)
  - Any discrepancies cause a conflict error
- **Add a timestamp column** to the table - updates itself with a new value every time a value in its row is modified.

[Optimistic Concurrency \(ADO.NET\)](#)

<http://msdn.microsoft.com/en-us/library/aa0416cz.aspx>

<http://msdn.microsoft.com/msdnmag/issues/04/09/DataPoints>



# How to Resolve Conflicts

- Use the **HasErrors** property to test for errors
  - Test a DataSet, DataTable, or DataRow
- Choose one of these strategies to resolve conflicts:
  - “Last in wins” - that data changes made by your application overwrite any database changes made by other applications.
  - Retain conflicting rows in your DataSet so you can update the database again later
  - Reject conflicting rows and revert to the original values in your DataSet
  - Reject conflicting rows and reload the latest data from the database



## Stosowanie obiektu DataSet

- **gdy dane muszą być edytowane lub gdy do bazy trzeba dodawać i usuwać rekordy.**
- **gdy zachodzi potrzeba organizowania danych - filtrowania, sortowania czy wyszukiwania**
- **gdy rekordy pobrane z bazy danych będą przetwarzane w wielu iteracjach**
- **gdy wynikowy zbiór danych pomiędzy kolejnymi odwołaniami do tej samej strony musi zostać zachowany w obiekcie Session lub Cache.**
- **do przekazywania wyników działania obiektów warstwy biznesowej i usług Web Service**
  - **odłączony obiekt DataSet może być serializowany do postaci XML i przesyłany z wykorzystaniem protokołu HTTP**



# DataBinding

**Data binding** is the process that establishes a connection between the application **UI controls** and business logic in such a way that changes to the **data source** („intermediate“) initiate similar changes in the data contained in the UI controls.

## Tasks of a data binding:

- Retrieving the data from a data source
- Gluing the data to a user interface control
- Allowing manipulation, including some sorting and paging
- Optionally changing the data and synchronization of such changes with the original data source

```
private void InitializeSortedFilteredBindingSource()
{
    // Create the connection string, data adapter and data table.
    SqlConnection connectionString =
        new SqlConnection("Initial Catalog=Northwind;" +
            "Data Source=localhost;Integrated Security=SSPI;");
    SqlDataAdapter customersTableAdapter =
        new SqlDataAdapter("Select * from Customers", connectionString);
    DataTable customerTable = new DataTable();
    // Fill the the adapter with the contents of the customer table.
    customersTableAdapter.Fill(customerTable);
    // Set data source for BindingSource1.
    BindingSource1.DataSource = customerTable;
    // Filter the items to show contacts who are owners.
    BindingSource1.Filter = "ContactTitle='Owner'";
    // Sort the items on the company name in descending order.
    BindingSource1.Sort = "Country DESC, Address ASC";
    // Set the data source for dataGridView1 to BindingSource1.
    dataGridView1.DataSource = BindingSource1;
}
```



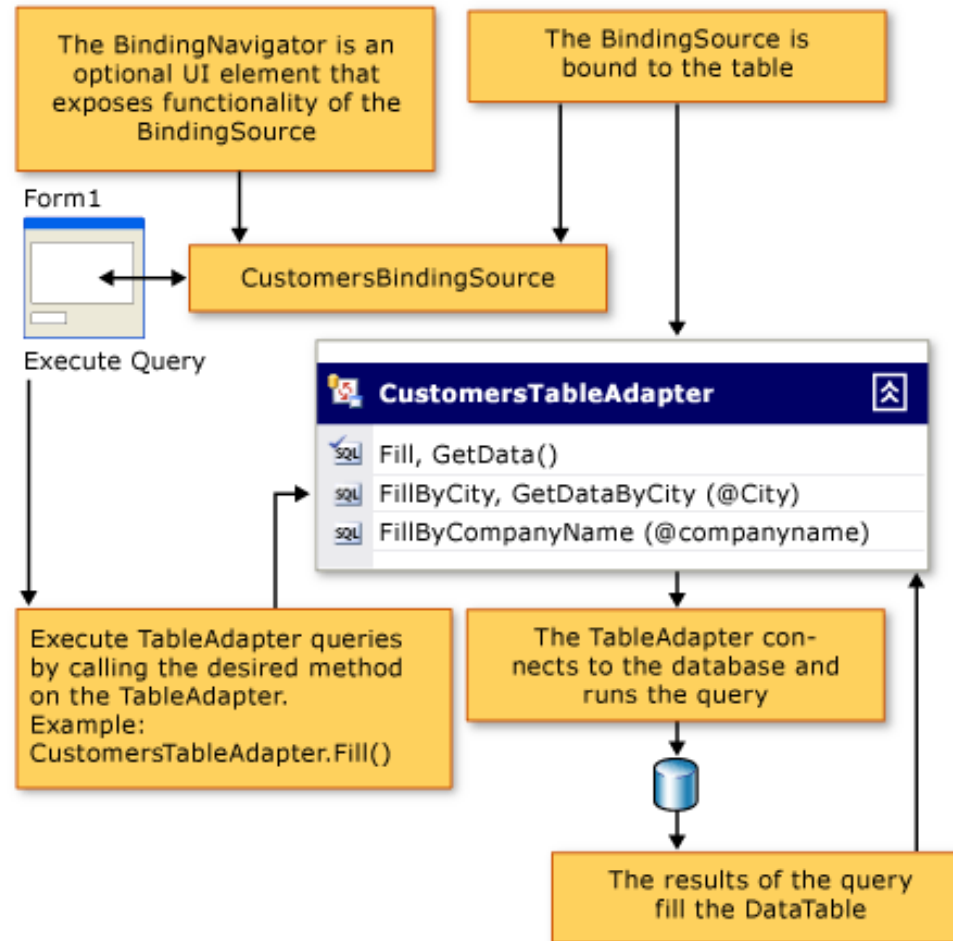
# Data Binding and Windows Forms

Type of data binding	Description
Simple data binding	The ability of a control to bind to a single data element, such as a value in a column in a dataset table. This is the type of binding typical for controls such as a <a href="#">TextBox</a> control or <a href="#">Label</a> control, which are controls that typically only displays a single value
Complex data binding	The ability of a control to bind to more than one data element, typically more than one record in a database. Complex binding is also called list-based binding. Examples of controls that support complex binding are the <a href="#">DataGridView</a> , <a href="#">ListBox</a> , and <a href="#">ComboBox</a> controls.

- A ***BindingSource*** is the most common Windows Forms data source and acts a proxy between a **data source** (for example, an ADO.NET data table or a business object) and **Windows Forms controls**.



# Connecting to Data in Visual Studio



The standard flow of operations when connecting to data by executing a **TableAdapter** query to fetch data and display it on a form in a Windows application

[Controls to Use on Windows Forms](http://msdn.microsoft.com/en-us/library/3xdhey7w.aspx)

<http://msdn.microsoft.com/en-us/library/3xdhey7w.aspx>



# Data Containers and Data Source Controls

## Kontrolki źródeł danych:

- **SqlDataSource:** umożliwia odczytywanie i zmianę informacji w bazie danych MS SQL
- **XMLDataSource:** odczyt i zmiana informacji w plikach XML
- **ObjectDataSource:** odczyt i zmiana danych zawartych w niestandardowych obiektach
- **SiteMapDataSource:** odczyt informacji z pliku mapy witryny
- **LinqDataSource:** odczyt i modyfikacja danych z użyciem języka LINQ

## Dostęp do pojemników danych:

- **Data Connection:** umożliwia połączenie z bazą danych
- **Data Source:** źródło danych umożliwia odczyt i aktualizację danych
- **Data-bound Control:** kontrolka wiążąca dane wyświetla informacje przekazywane przez źródło danych lub przekazuje dane do źródła

## Kontroli prezentacji informacji:

- **DataGridView/GridView:** wyświetlanie rekordów w formie arkusza, umożliwia modyfikację i usuwanie danych
- **DetailsView:** wyświetla jeden rekord; wstawianie, modyfikacja, usuwanie
- **FormView:** wyświetla jeden rekord w sformatowanej postaci; wstawianie, modyfikacja, usuwanie
- ...



## Kontrolki źródła danych

ObjectDataSource	Umożliwia połączenie z obiektami logiki biznesowej i innych klas i służy do tworzenia aplikacji webowych które bazują na obiektach warstwy pośredniej do zarządzania danymi. Wspiera zaawansowane sortowania i dzielenie na strony niedostępne w innych kontrolkach źródeł danych.
SqlDataSource	Umożliwia połączenie do serwerów baz danych takich jak Microsoft SQL Server czy Oracle. We współpracy z serwerem SQL Server wspiera zaawansowane możliwości buforowania. Kontrolka wspiera również sortowanie, filtrowanie i dzielenie na strony, jeśli dane są zwracane jako obiekt DataSet.
AccessDataSource	Umożliwia współpracę z bazami danych zapisanymi w Microsoft Access. Kontrolka wspiera sortowanie, filtrowanie i dzielenie na strony, jeśli dane są zwracane jako obiekt DataSet
XmlDataSource	Umożliwia pobieranie danych zapisanych w plikach XML, szczególnie dla hierarchicznych kontrolek serwerowych ASP.NET takich jak TreeView. Wspiera filtrowanie przy użyciu wyrażeń XPath i umożliwia stosowanie transformacji danych przy użyciu XSLT. Umożliwia również aktualizację danych przez zapisanie całego dokumentu XML ze zmianami.
SiteMapDataSource	Używana w ASP.NET do nawigacji na stronie
LinqDataSource	Umożliwia użycie języka LINQ na stronach ASP.NET poprzez model deklaratywny do pobrania i modyfikowania danych z obiektów danych takich jak tabele w bazie czy kolekcje w pamięci serwera. Wspiera automatyczne generowanie poleceń wybierania, aktualizacji, dodawania i usuwania danych. Kontrolka wspiera sortowanie, filtrowanie i dzielenie na strony
EntityDataSource	Umożliwia połączenie z danymi pochodzącymi z modelu Entity Data Model (EDM). Wspiera automatyczne generowanie poleceń wybierania, aktualizacji, dodawania i usuwania danych. Kontrolka wspiera sortowanie, filtrowanie i dzielenie na strony.



## Kontrolki prezentacji danych (Form, Web)

BulletedList, CheckBoxList, DropDownList, ListBox, RadioButtonList	Kontrolki listy – wyświetlają dane w różnych formatach
AdRotator	wyświetla ogłoszenia na stronie – dane o lokalizacji mogą być umieszczone w źródle danych
DataList	wyświetla dane w tabeli, każdy element jest wyświetlany zgodnie z zdefiniowanym szablonem
DetailsView	wyświetla jeden rekord w układzie tabelarycznym i umożliwia edycję, usuwanie i dodawanie rekordów.
FormView	kontrolka podobna do kontrolki DetailsView, ale umożliwia definiowanie wyglądu dla każdego rekordu. Kontrolka FormView jest podobna do kontrolki DataList, ale dla pojedynczego rekordu
GridView	wyświetla dane w tabeli i umożliwia edycję, zmianę, usuwanie, sortowanie, dzielenie na strony bez konieczności pisania kodu.
ListView	wyświetla dane zgodnie z definicją umieszczoną w szablonie. Umożliwia automatyczne sortowanie, edycję, dodawanie i usuwanie rekordów. Możliwe jest również dzielenie na strony przy pomocy połączonej kontrolki DataPager
Repeater	wyświetla dane w postaci listy. Każdy rekord jest wyświetlany przy użyciu szablonu
TreeView	wyświetla dane w postaci drzewa hierarchicznego z rozwijanymi węzłami
Menu	wyświetla dane w postaci hierarchicznego, dynamicznego menu, który może zawierać podmenu
DataGridView	wyświetla dane w tabeli i umożliwia edycję, zmianę, usuwanie, sortowanie, ...



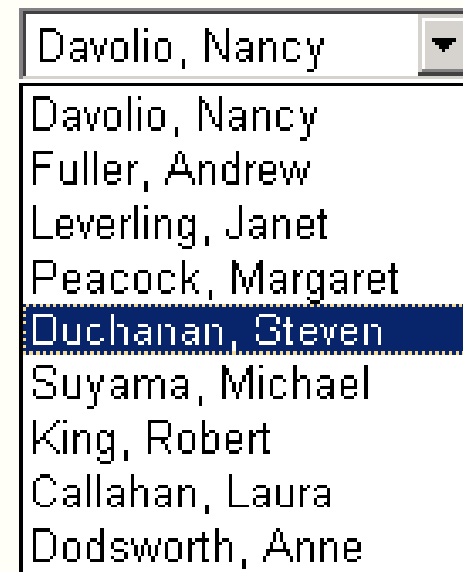
# Wiązanie danych złożonych Binding data to controls

**Wiązanie danych złożonych -** gdy wiążemy kontrolkę listy (*DropDownList, CheckBoxLayout, RadioButtonList, ListBox*) lub kontrolkę iteracyjną (*DataGrid*) z jedną lub kilkoma kolumnami danych

- **DataSource** property
- **DataBind** method

```
// data loading
DataTable _dataTable;
_dataTable = _dataHandler.Load();

// data binding
DDLst.DataSource = _dataTable;
DDLst.DataTextField = "Name";
DDLst.DataValueField = "empID";
DDLst.DataBind();
```



Property	Description
<b>DataSource</b>	▪ The <b>DataSet</b> containing the data
<b>DataMember</b>	▪ The <b>DataTable</b> in the <b>DataSet</b>
<b>DataTextField</b>	▪ The field in the <b>DataTable</b> that is displayed
<b>DataValueField</b>	▪ The field in the <b>DataTable</b> that becomes the value of the selected item in the list

```
np.:
DataGrid1.DataSource=productsDataSet;
DataGrid1.DataMember="Products";
```



## DataBinding with a DataGridView - przykład

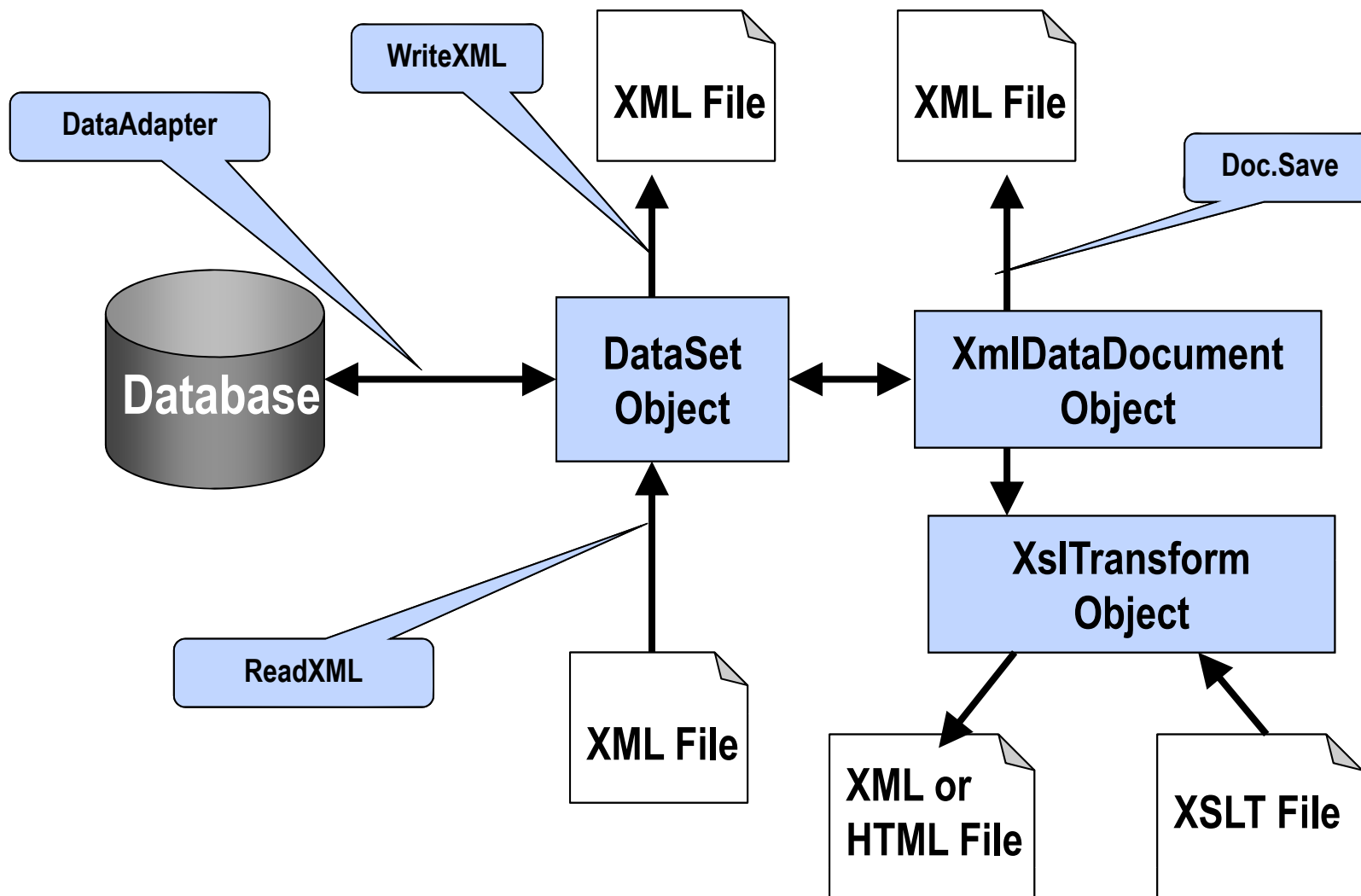
```
// Turn this off so column names do not come from data source
dataGridView1.AutoGenerateColumns = false;
// Specify table as data
source dataGridView1.DataSource = ds; // Dataset
dataGridView1.DataMember = "movies"; // Table in dataset
// Tie the columns in the grid to column names in the data table
dataGridView1.Columns[0].DataPropertyName = "Title";
dataGridView1.Columns[1].DataPropertyName = "Year";
dataGridView1.Columns[2].DataPropertyName = "director";
```

**Two-way data binding: Changes made to the grid are reflected in the underlying table, and changes made to the table are reflected in the grid**

```
private void buttonAdd_Click(object sender, EventArgs e)
{
// Adding to underlying table is okay
r[0] = "TAXI";
r[1] = "1976"; r[2] = "Martin Scorsese";
dt.Rows.Add(r);
}
```



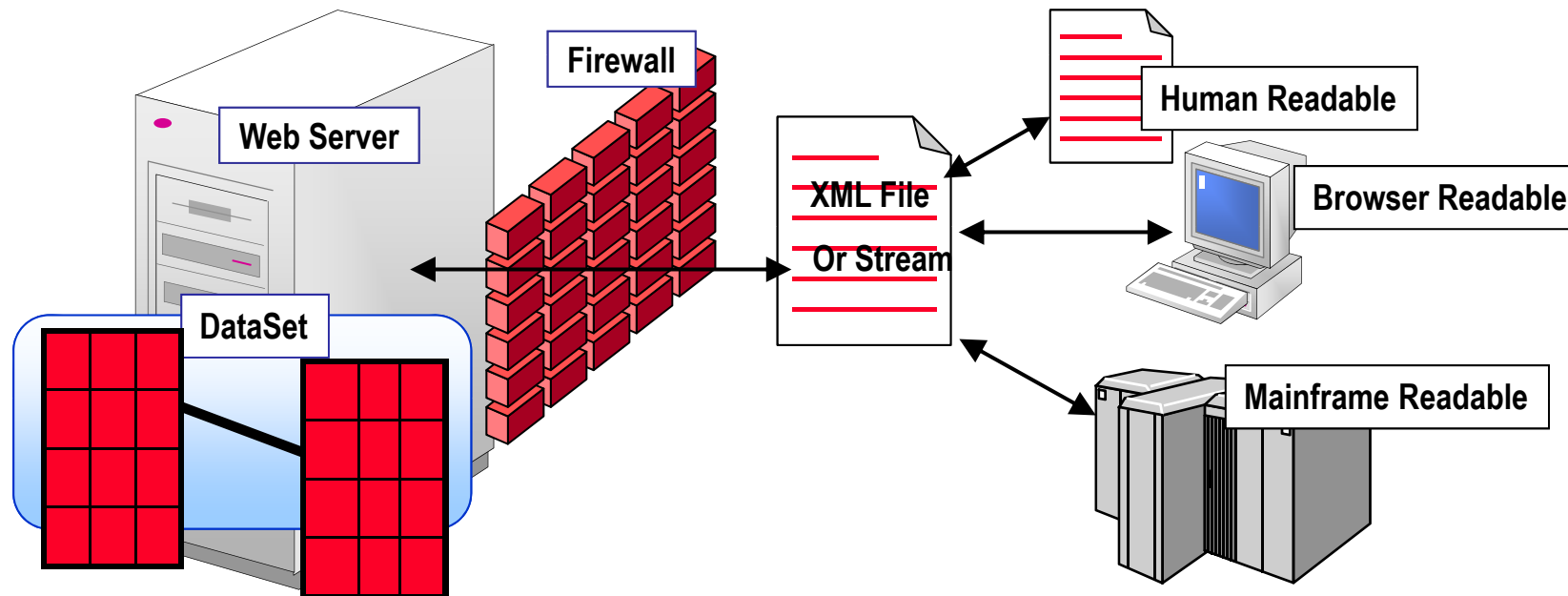
## 2.3.4 XML i DataSets





# Why Use XML with Datasets?

- XML is the universal format for exchanging data on the Internet
- XML provides a convenient format for transferring the contents of a dataset to and from remote clients
- XML objects synchronize and transform data



- **DataSets and XML are highly integrated**
  - serializing DataSets as XML data
  - XML documents as data sources for DataSets
  - schemas for DataSets defined as XML schemas

- *strongly typed* DataSets generated from XML schemas
- access to DataSets using XML-DOM interface

- **Integration of DataSets and XML used in distributed systems, e.g., web services**



## Metody klasy DataSet do obsługi danych i schematów XML

- **ReadXML** - loads XML data into a DataSet.
- **WriteXml** - writes the DataSet's contents to an XML formatted stream.
- **WriteXmlSchema** - generates an XML Schema from the **DataSet** schema.
- **ReadXmlSchema** - reads an XML Schema file and creates a database schema.
- **InferXmlSchema** - creates a DataSet schema from XML **data**.
- **GetXml** and **GetXmlSchema** - returns a string containing the XML representation of the data or the XSD schema for XML representation

[Using XML in a DataSet \(ADO.NET\)](http://msdn.microsoft.com/pl-pl/library/84sxtbxh(en-us).aspx)

[http://msdn.microsoft.com/pl-pl/library/84sxtbxh\(en-us\).aspx](http://msdn.microsoft.com/pl-pl/library/84sxtbxh(en-us).aspx)



## Loading a DataSet from XML

- Use *ReadXml* to load data from a file or stream. It will also create the relational schema of the *DataSet* depending on the *XmlReadMode* specified and whether or not a relational schema already exists.

```
DataSet.ReadXML(FileName | Stream, XMLReadMode)
```

```
DataSet ds = new DataSet();  
ds.ReadXml(Server.MapPath("filename.xml"));
```

### *XmlReadMode* argument

- ReadSchema
- IgnoreSchema
- InferSchema
- DiffGram
- ...



# Writing DataSet Contents as XML Data

- Use `GetXml` to write data to a string variable

```
string strXmlDS = ds.GetXml();
```

- Use `WriteXml` to write XML data to a file or stream.

```
DataSet.WriteXml (FileName|Stream, XmlWriteMode)
```

```
DataSet ds = new DataSet();  
SqlDataAdapter da = new SqlDataAdapter("select * from  
    Authors", conn);  
da.Fill(ds);  
ds.WriteXml(Server.MapPath("filename.xml"));
```

## *XmlWriteMode* argument

- IgnoreSchema
- WriteSchema
- DiffGram
- ...



# DiffGrams

- **What is a DiffGram?**
  - A file or a stream in XML format that represents **changes** made to a DataSet
- **Creating a DiffGram**
  - Set the *WriteXml* method of the *DataSet* object with the *XmlWriteMode* parameter set to *DiffGram*.

- **DiffGram Format**

```
<?xml version="1.0"?>
```

```
<diffgr:diffgram
```

```
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
```

```
  xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<DataInstance>
```

contains the current data

```
</DataInstance>
```

```
<diffgr:before>
```

contains the original version of a row

```
</diffgr:before>
```

```
<diffgr:errors>
```

```
</diffgr:errors>
```

```
</diffgr:diffgram>
```



# Loading/Writing DataSet Schema Information from XML/as XML

## Loading XSD Information from a File

- Use *ReadXMLSchema* to load an existing XSD schema into a DataSet

```
DataSet.ReadXMLSchema (FileName | Stream);
```

```
System.IO.StreamReader xmlStream = new System.IO.StreamReader("schema.xsd");  
DataSet dataSet = new DataSet();  
dataSet.ReadXmlSchema(xmlStream);  
xmlStream.Close();
```

## Loading XSD Information from a File

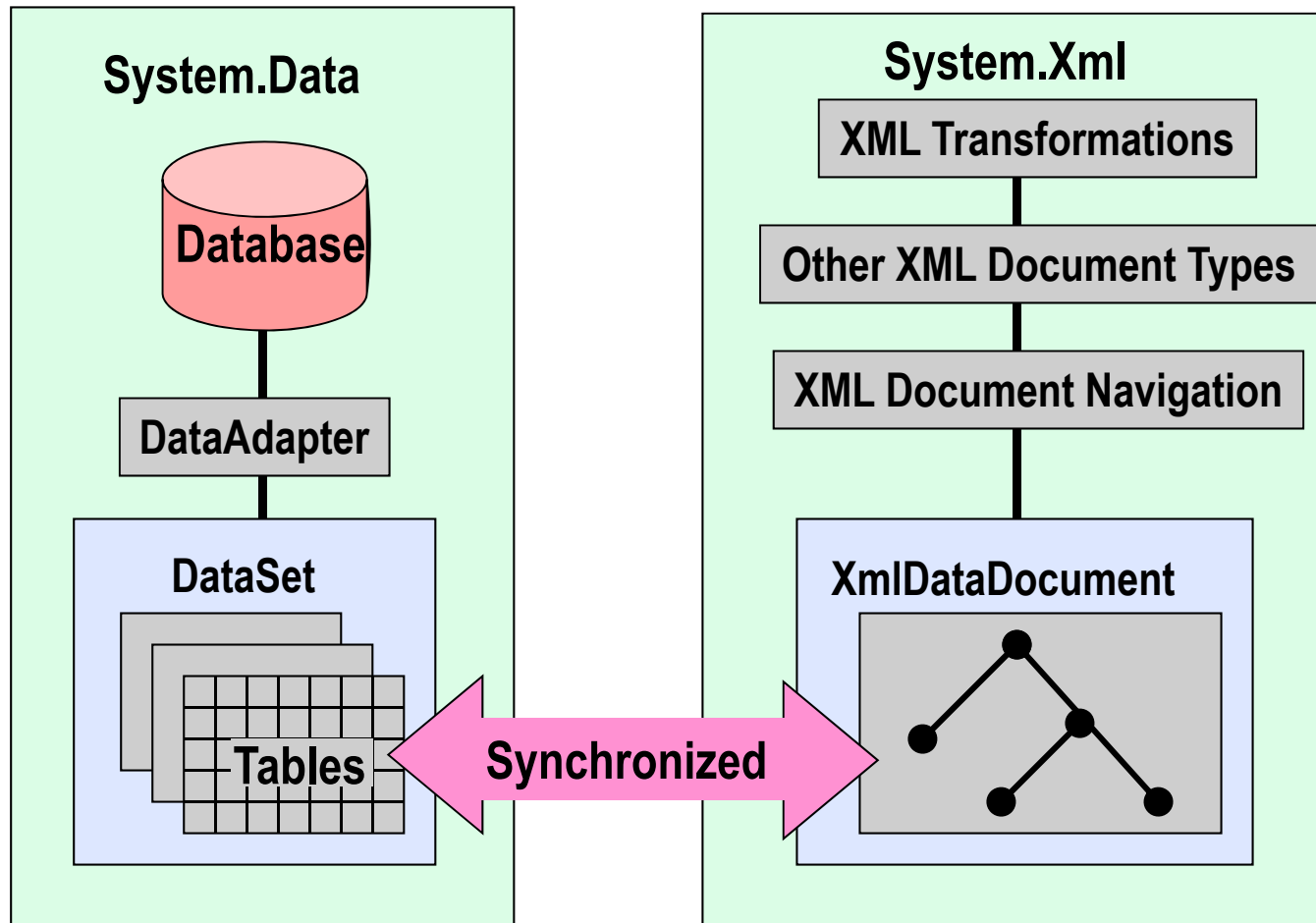
- Use *WriteXMLSchema* to write the schema of a DataSet as XML Schema to a file, stream

```
DataSet.WriteXMLSchema (FileName | Stream);
```

```
System.IO.StreamWriter writer = new System.IO.StreamWriter("Customers.xsd");  
dataSet.WriteXmlSchema(writer);  
writer.Close();
```



# Synchronizing a DataSet with an XmlDocument





## How to Synchronize a DataSet with an XmlDocument

- Store XML Data into an XmlDocument

```
XmlDataDocument objXmlDataDoc = new XmlDocument();  
objXmlDataDoc.Load(Server.MapPath ("file.xml"));
```

- Store a DataSet in an XmlDocument

```
DataSet ds = new DataSet();  
//fill in ds  
objXmlDataDoc = new XmlDocument(ds);
```

- Use XML DOM methods
  - *XmlDataDocument* inherits from *XmlDocument*
- Apply an XSLT transformation
  - *XsltTransform* object