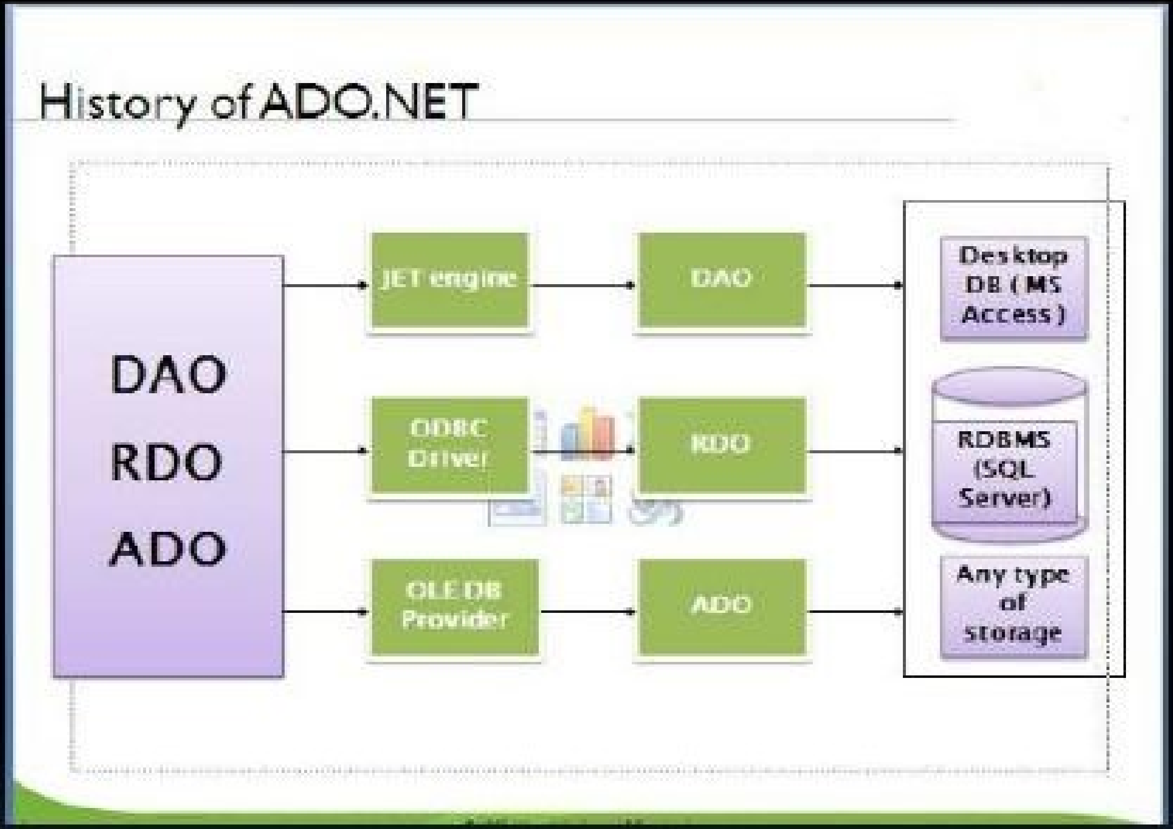
ADO.NET

BY
K. BHASKARA RAO
Asst. Prof.
DEPARTMENT OF IT
BEC

Short History of Data Access

- Data Access Objects (DAO/Jet)
- Open Database Connectivity (ODBC)
- OLE for Databases (OLE/DB)
- ActiveX Data Objects (ADO)



History of ADO.NET

DAo- Data Access Object
RDO- Remote Data Object
ADO ActiveX Data Object

ODBC

- ODBC is Open Data Base Connectivity, which is a connection method to data sources and other things.
- It requires that you set up a data source, or what's called a DSN using an SQLdriver or other driver if connecting to other database types.
- Most database systems support ODBC.

OLEDB

- OLEDB is the successor to ODBC
- A set of software components that allow a "front end" such as GUI based on VB, C++, Access or whatever to connect with a back end such as SQL Server, Oracle, DB2, mySQL …
- The OLEDB components offer much better performance than the older ODBC.
- It does not require that you set up a DSN

Why is ADO.NET Better?

- Disconnected by Design
- Relational by Nature
- Integration with XML
- Framework Supports Real Database Schema

Connectionless Data Access

- Motivation
  - Many parallel, long lasting access operations
  - Connection-oriented data access too costly

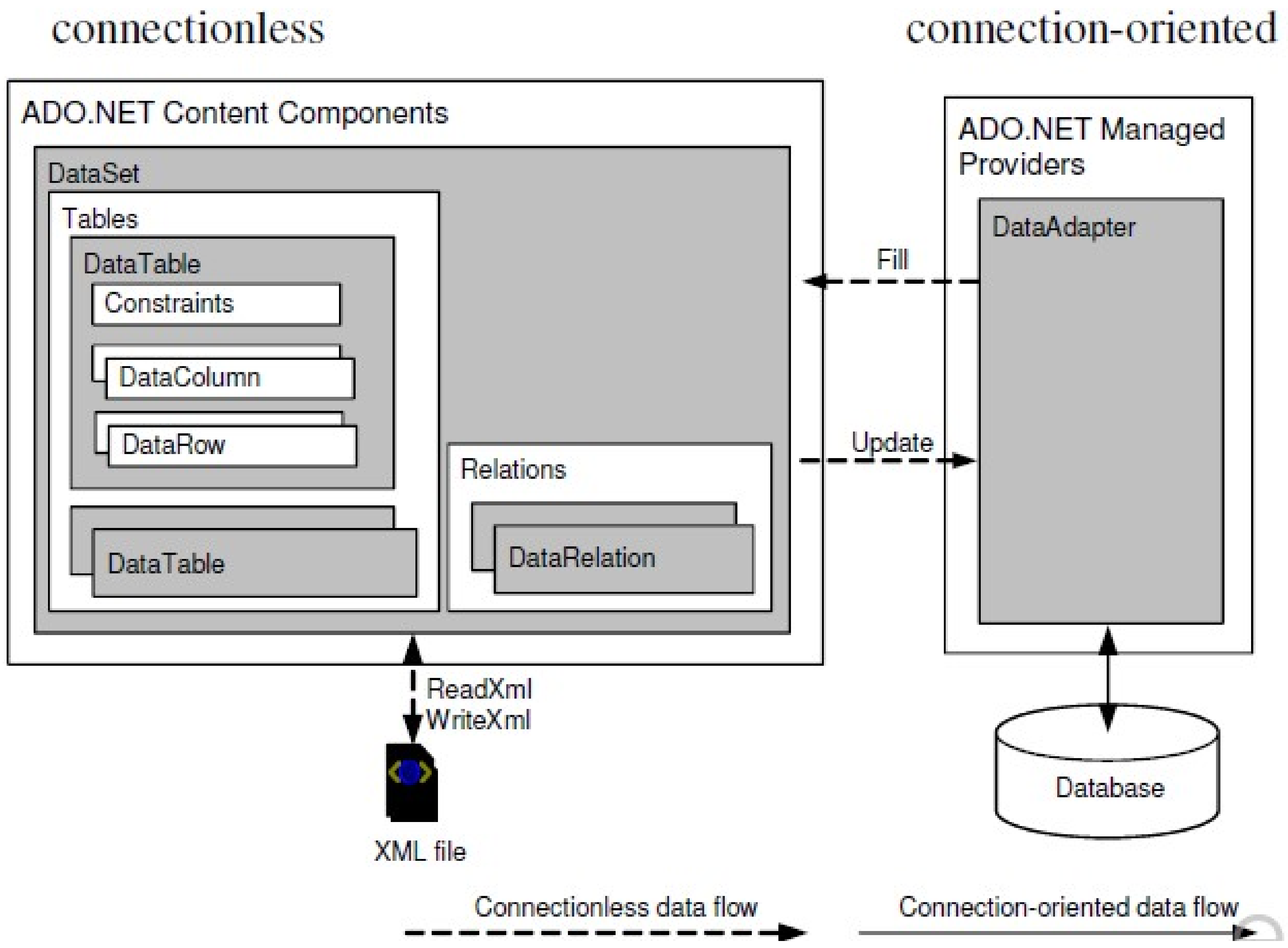
- Idea
  - Caching data in main memory

    ➔ “main memory data base“
  - Only short connections for reading and updates

    ➔ DataAdapter
  - Main memory data base independent from data source

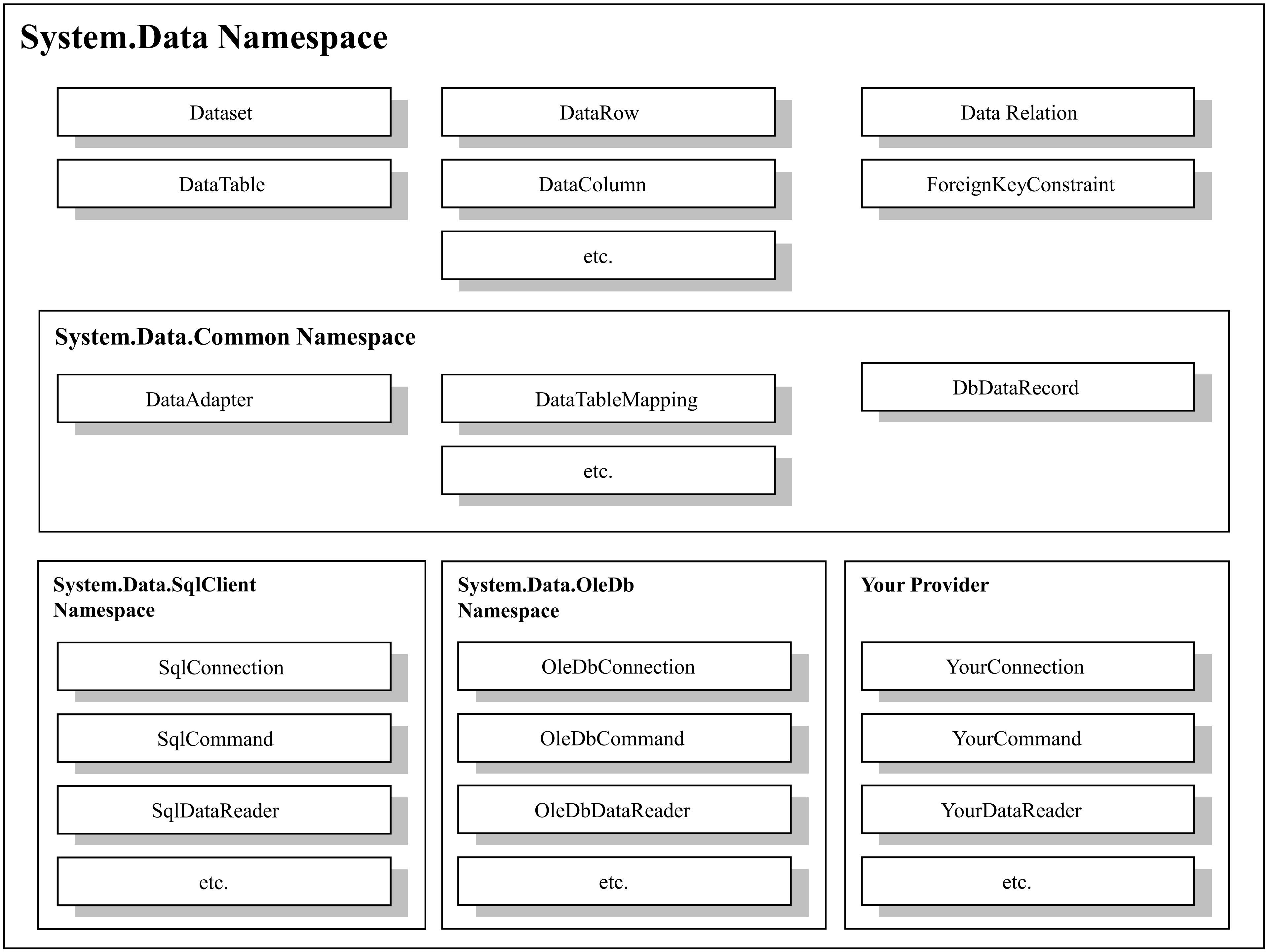
    ➔conflicting changes are possible

Architecture of Connectionless Data Access

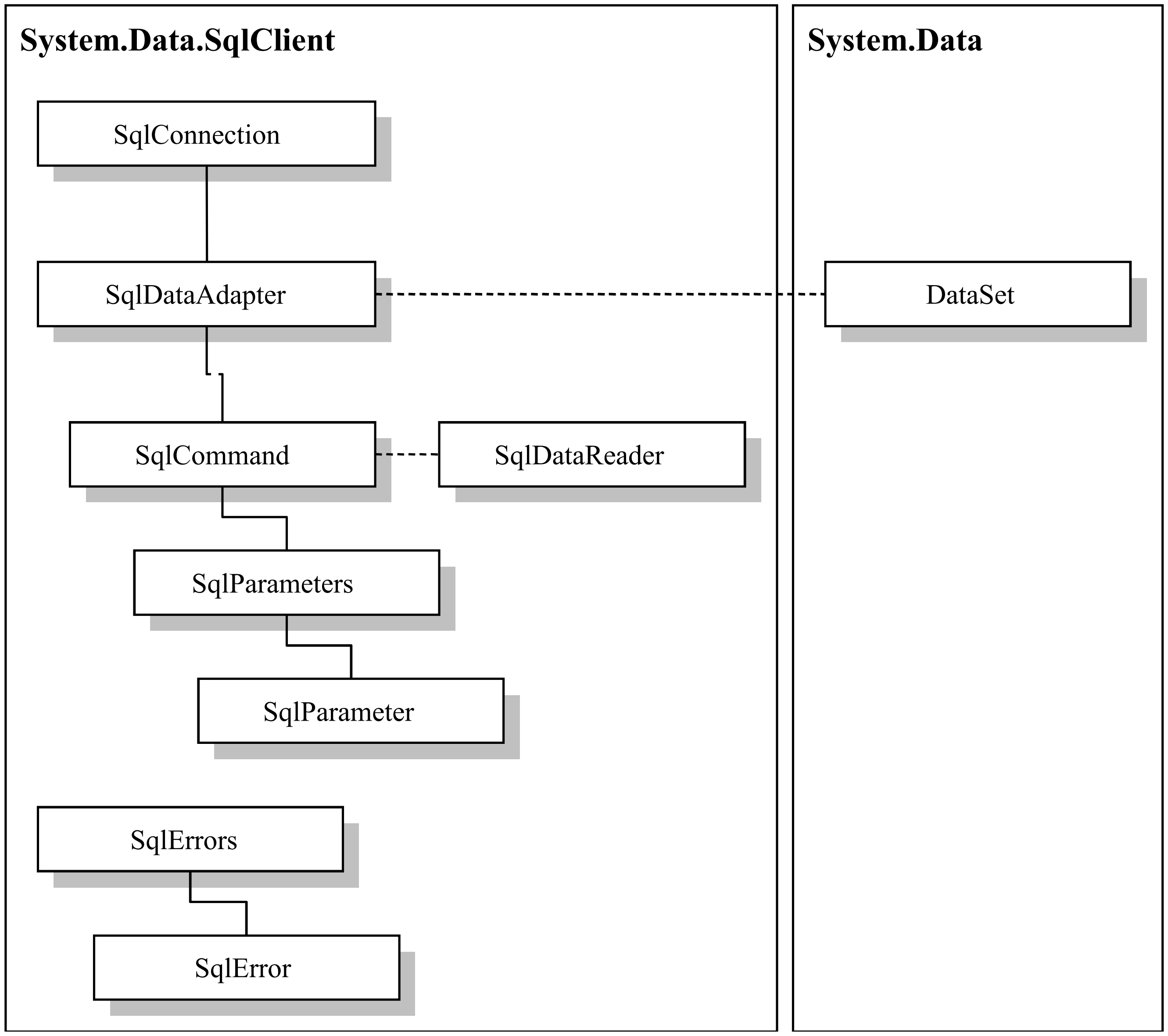


Introducing ADO.NET

- Managed Providers
- DataSet
- DataBinding in ASP.NET


System.Data Namespace
Dataset
DataRow
Data Relation
DataTable
DataColumn
ForeignKeyConstraint
etc.
System.Data.Common Namespace
DataAdapter
DataTableMapping
DbDataRecord
etc.
System.Data.SqlClient Namespace
SqlConnection
SqlCommand
SqlDataReader
etc.
System.Data.OleDb Namespace
OleDbConnection
OleDbCommand
OleDbDataReader
etc.
Your Provider
YourConnection
YourCommand
YourDataReader
etc.

Managed Provider Abstraction



ADO.NET - Database Technology

This technology takes its name from ADO(ActiveX Data Objects)

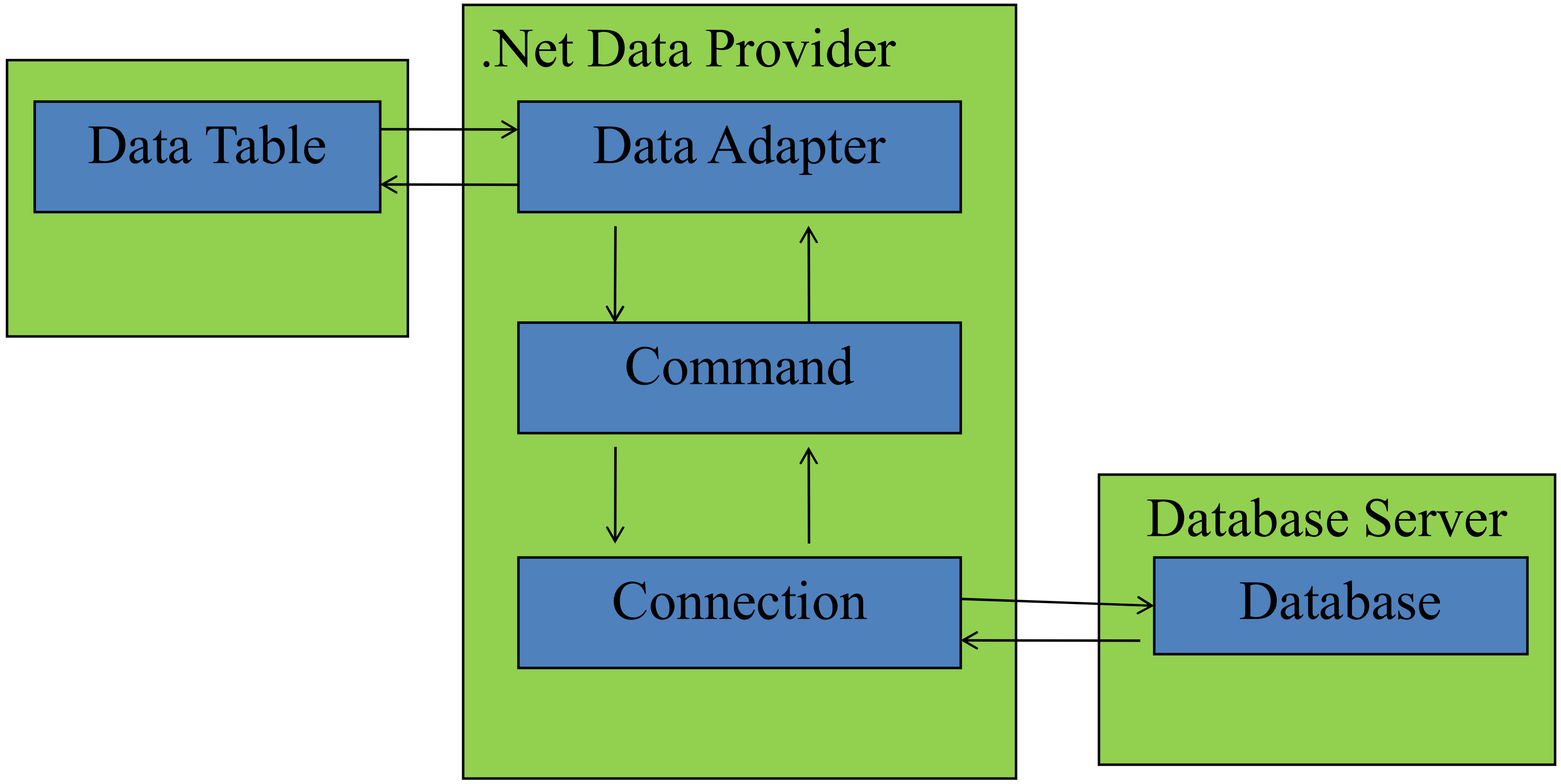
New ADO.Net – is designed to overcome the limitation of ADO technology. It deals with accessing and manipulating databases. It comprises of many namespaces and classes to do so. ADO.Net provides accesses to data source such as Microsoft SQL Server, OLE-DB and XML etc.

ADO technology supports only Connected approach. But new ADO.Net supports both a connected and disconnected approach.

I. Connected approach
II. Disconnected approach

ADO.NET

Basic ADO.NET Objects :



Direct data access with ADO.NET

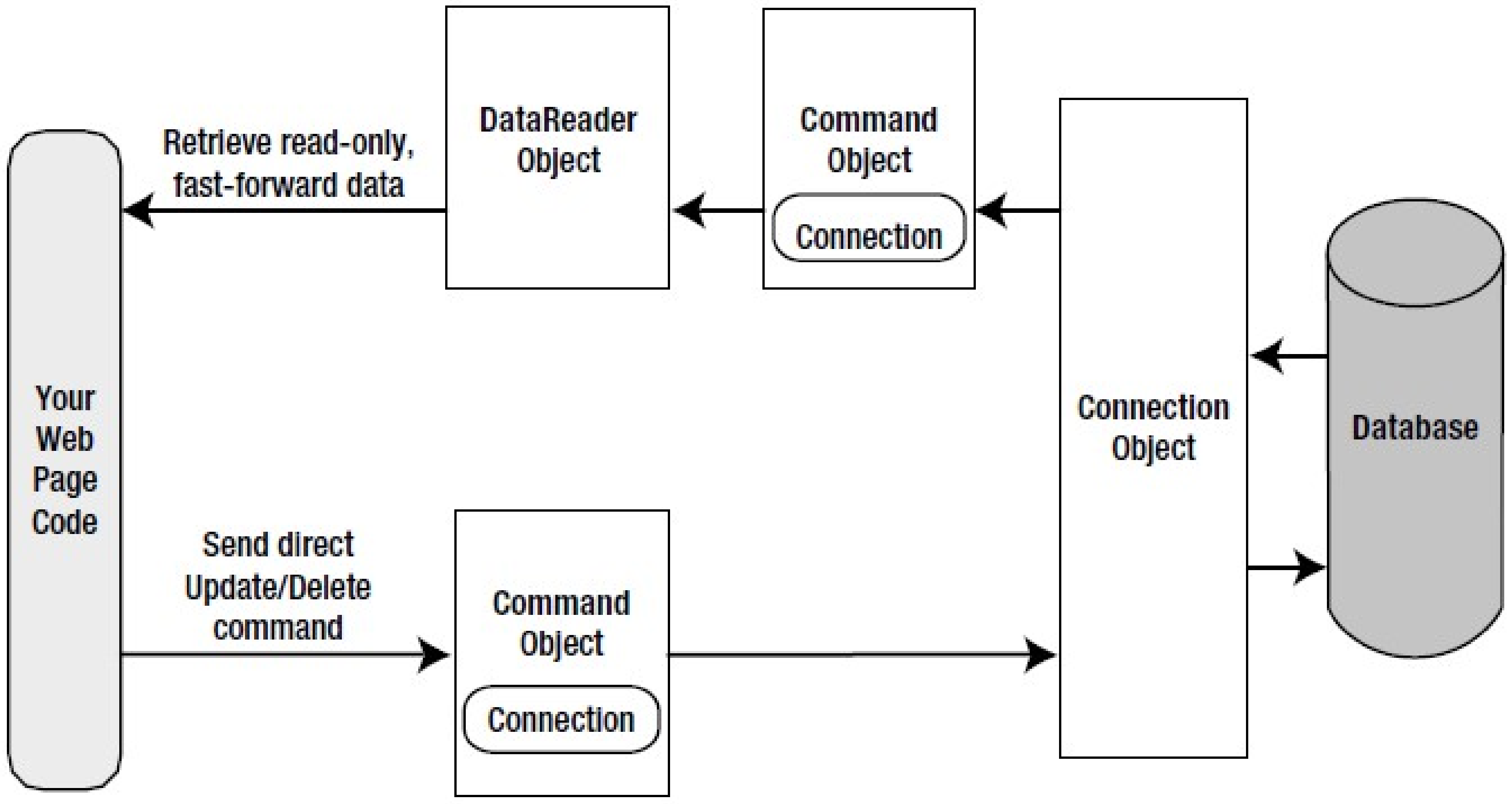


Figure 14-8. *Direct data access with ADO.NET*

ADO.Net has two major components

1. The .Net Data providers
2. The DataSet

A .Net Data Provider is used for connecting to a database, executing commands, and retrieving results. Using the .Net Data Providers, we can either access database directly or use the disconnected approach, in disconnected approach we use DataSet class.

The .Net Data Providers - It has following basic objects.

1. A Connection Object
2. A Command Object
3. A DataReader Object
4. A DataAdapter Object

The .Net Data Provider objects

The Connection Object is used to connected to the data source. Data source can be any database file. The Connection object contains information like the provider name, server name, data source name, user name, password and so on.

A Command Object is used to connect the Connection Object to a dataReader or DataAdaptor Object. The Command Object allows us to execute an SQL statement or a stored procedure in a data source.

The DataReader Object is used to read the data in a first an efficient manner from the database. It is generally used to extract one or a few records or specific field values, or to execute simple SQL statements.

A DataAdaptor Object is used to fill data from the database into the DataSet Object. The DataSet is used in the disconnected approach.

Abstract Base Classes of a Data Provider

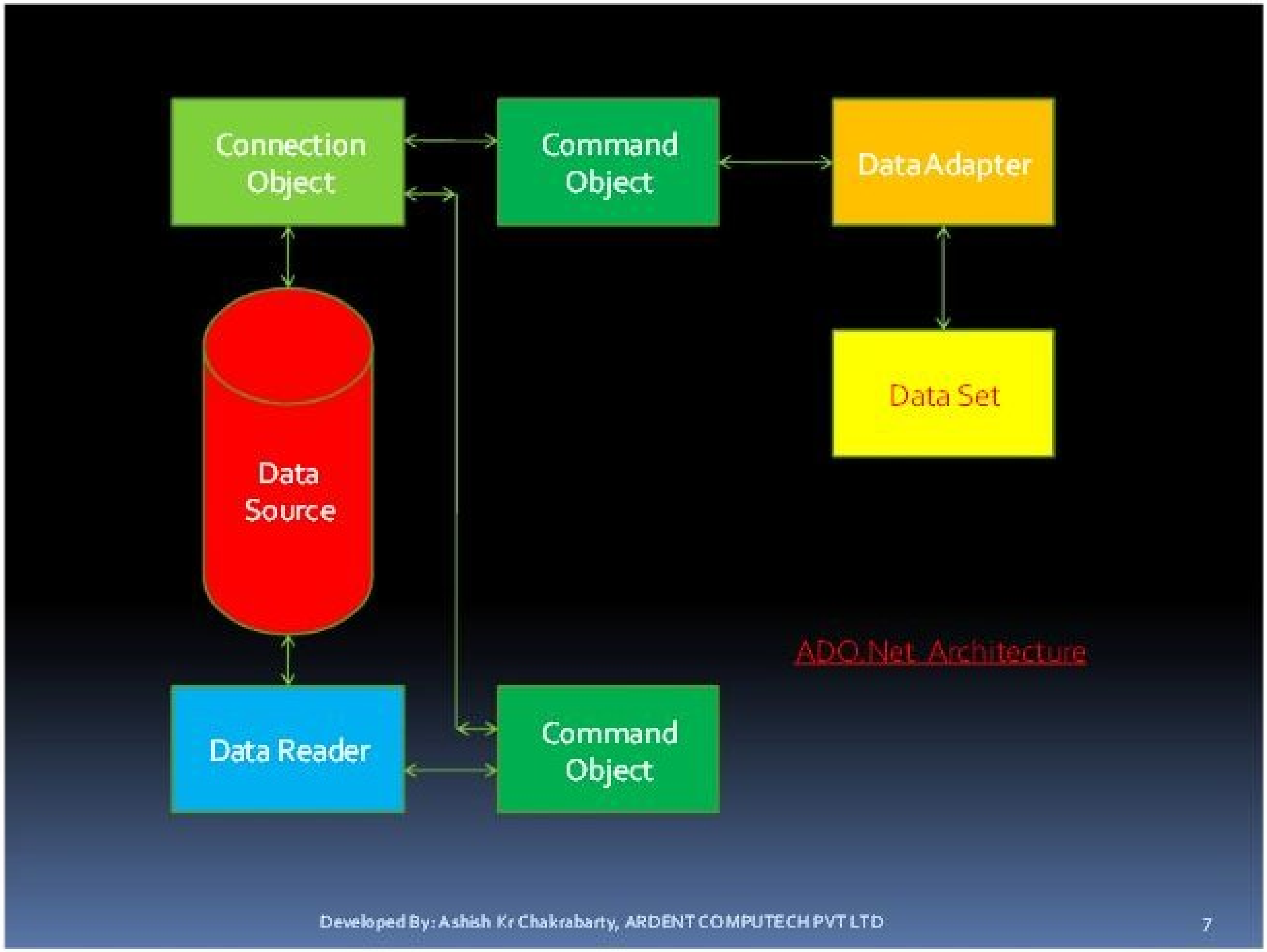
Class	Description
`DbCommand`	Executes a data command, such as a SQL statement or a stored procedure.
`DbConnection`	Establishes a connection to a data source.
`DbDataAdapter`	Populates a `DataSet` from a data source.
`DbDataReader`	Represents a read-only, forward-only stream of data from a data source.

Data Providers in the .NET

Provider	Namespace	Description
SQL Server	System.Data.SqlClient	Lets you access SQL Server database.
OLE DB	System.Data.OldDb	Lets you access any database that supports OLE DB.
ODBC	System.Data.Odbc	Lets you access any database that supports ODBC
Oracle	System.Data.OracleClient	Lets you access Oracle database

Class Names for the data Providers

Object	SQL Server	OLE DB	ODBC	Oracle
Connection	SqlConnection	OleDbConnection	OdbcConnection	OracleConnection
Command	SqlCommand	OldDbCommand	OdbcCommand	OracleCommand
Data reader	SqlDataReader	OleDbDataReader	OdbcDataReader	OracleDataReader
Data adapter	SqlDataAdapter	OleDbDataAdapter	OdbcDataAdapter	OracleDataAdapter



ADO.Net Architecture

Using Data Provider Classes

- The process for using these classes is as follows:
  1. Create a `DbConnection` to the database via a connection string.
  2. Create and execute a `DbCommand` for the database.
  3. [Optional] Fill a `DataSet` from the database using a `DbDataAdapter`, or use a `DbDataReader` to retrieve data from the database.

DBConnection

- Represents a connection to a data source through which commands are passed to the data source and through which data is returned.
- Before you can access a database, you must first create a connection to it.
- Database commands then “travel” across the connection to the database, as does any data returned from a database.

- Each `DbConnection` class has members for:
  - opening and closing a connection,
  - setting and retrieving properties of a connection,
  - handling connection-related events.

DbCommand

- Represents an SQL statement or a stored procedure that is executed by the data source.
- Each `DbCommand` class has members for:
  - representing an SQL statement,
  - creating data parameters,
  - executing SQL commands that either return data (e.g., SELECT) or do not return data (e.g., INSERT, DELETE, or UPDATE).
  - can also be used to run stored procedures if the database supports them.

Executing a DbCommand

- Once a `DbCommand` object has been instantiated and its command text, command type, and connection set, the command can be run by calling one of the `Execute` methods of the `DbCommand` class:
  - `ExecuteNonQuery`
    - for commands that do not return any record data, such as an SQL INSERT or DELETE.
  - `ExecuteScalar`
    - for SELECT commands that return a single value.
  - `ExecuteReader`
    - for SELECT commands that return multiple results.
    - How are these results returned?
      - returns a `DbDataReader.`

Using DbParameters

- Represent a parameter to a command.
- Parameters are used to specify criteria to a query
  - that is, they are used to construct **parameterized queries**.
  - They are a more secure alternative to simply building a query with criteria via string building.

Using a DbParameter

- Creating a command using a `DbParameter` involves three steps:
  - Modify the SQL WHERE clause so it uses parameter names instead of values.

```
string s = "select * from Users where UserId=@user";
```

  - Create the appropriate `DbParameter` objects and assign them the appropriate names and values.

```
SqlParameter param = new SqlParameter("@user",txtUser.Tex
```

  - Add the created `DbParameter` objects to the `DbCommand` object's Parameters collection.
    - This step must be done before calling any of the `DbCommand` object's `Execute` methods.

```
SqlCommand cmd;
…
cmd.Parameters.Add(param);
```

Transactions

- Transactions provide a way to gracefully handle errors and keep your data properly consistent when errors do occur.
- Transactions can be implemented both in ADO.NET as well as within the DBMS.
- Transactions can be **local** or **distributed**.

Local Transaction Steps

- Create a DbTransaction object by calling the BeginTransaction method of the DbConnection class.
- Assign the DbTransaction object to each DbCommand object being executed as part of the transaction via its Transaction property.
- Execute each DbCommand.
- **Commit** (i.e., save the database changes) if everything worked okay or **roll back** (i.e., undo any database changes) if an exception occurred.
  - If the connection is closed before a commit or a roll back (caused, for instance, by a crash in the DBMS or the web server), then the transaction would be rolled back.

DbDataReader

- The `DbDataReader` is optimized for the fast retrieval of a read-only stream of records and is thus ideal for web applications.
  - `DbDataReader` is not a data container like the `DataSet`, but a kind of pointer to a record in a result set (that is, a set of records returned from a database query).
- `DbDataReader` also implements the `IEnumerable` interface so multi-value web server controls can be data bound to it.

Programming a DbDataReader

- Field data can be retrieved from the current record in the reader since the reader object acts like a collection, with each element in the collection corresponding to a field in the record.

```
SELECT Id,ProductName,Price FROM Products
```

```
// retrieve using column name
int id = (int)reader["Id"];
string name = (string)reader["ProductName"];
double price = (double)reader["Price"];

// retrieve using a zero-based column ordinal
int id = (int)reader[0];
string name = (string)reader[1];
double price = (double)reader[2];

// retrieve a typed value using column ordinal
int id = reader.GetInt32(0);
string name = reader.GetString(1);
double price = reader.GetDouble(2);
```

DbDataAdapter

- These classes represent a bridge between the `DataSet` container and an underlying database.
- Each `DbDataAdapter` class provides a:
  - `Fill` method for filling a `DataSet` (or just a `DataTable`) with data from the database
  - `Update` method for outputting any changes made to the data in the `DataSet` back to the database.
- The `DbDataAdapter` can also persist changes made to the in-memory data by writing the changes back to the database.

Programming a DbDataAdapter

```
DataSet ds = new DataSet();

// create a connection
SqlConnection conn = new SqlConnection(connString);

string sql = "SELECT Isbn,Title,Price FROM Books";
SqlDataAdapter adapter = new SqlDataAdapter(sql, conn);

try
{
   // read data into DataSet
   adapter.Fill(ds);

   // use the filled DataSet
}
catch (Exception ex)
{
   // process exception
}
```

```
string connString = "…"
OleDbConnection conn = new OleDbConnection(connString);

string cmdString = "SELECT Id,ProductName,Price From Products";
OleDbCommand cmd = new OleDbCommand(cmdString, conn);

conn.Open();
OleDBDataReader reader = cmd.ExecuteReader();

someControl.DataSource = reader;
someControl.DataBind();

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

SqlConnection Class

- Before you can access the data in a database, you have to create a connection object that defines the connection to the database.

SqlConnection Properties and Methods :

Property	Description
ConnectionString	Contains information that lets you connect to the SQL Server database. It includes information such as name of the server, name of the database and login information.

Method	Description
Open	Opens a connection to a database.
Close	Closes a connection to a database.

SqlCommand Class

Property	Description
Connection	The SqlConnection object that is used by the Command to connect to the data base.
CommandText	SqlCommand text or name of the stored procedure
CommandType	A constant specifying whether the CommandText property contains SQL stmt (Text) or Stored Procedure (StoredProcedure).
Parameters	The collection of parameters used by the command

Method	Description
ExecuteReader	Executes a query and returns the result as a SqlDataReader object.
ExecuteNonQuery	Executes the command and returns an int representing the no.of rows affected.
ExecuteScalar	Executes a query and returns first column of the first row returned by query.

SqlDataAdapter Class

Property	Description
SelectCommand	Represents the Select statement or stored procedure used to query the database.
DeleteCommand	Represents the Delete statement or stored procedure used to delete a row from the database.
InsertCommand	Represents the Insert statement or stored procedure used to add a row to the database.
UpdateCommand	Represents the Update statement or stored procedure used to update a row in the database.

Method	Description
Fill	Executes the command identified by SelectCommand property and loads the result into a dataset object.
Update	Executes the commands identified by the DeleteCommand, InsertCommand and UpdateCommand properties for each row in the dataset that was deleted, added , or updated.

SqlDataReader

Property	Description
Indexer	Accesses the column with the specified index or name from the current row.
FieldCount	The number of columns in the current row.

Method	Description
Read	Reads the next row. Returns True if there are more rows. Otherwise, returns False.
Close	Closes the data reader.

DbAccess class – for Database Management(Ms SQL Server)

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

namespace <FileName>
{
   class DbAccess
   {

       static SqlConnection  con = new SqlConnection(@" Connection String");

      // Method use to execute SELECT command from the data source

      public static DataSet FetchData(string Query)
            {
              SqlDataAdapter  da = new SqlDataAdapter(Query, con);
              DataSet ds = new DataSet();
              da.Fill(ds);
              return ds;
                    }
```

```
//Method use to execute Non-Query(INSERT / UPDATE / DELETE )records
   public static bool SaveData(string Query)
    {
      try
      {
        SqlCommand cmd = new SqlCommand(Query, con);
        con.Open();
        cmd.ExecuteNonQuery();
        return true;
          }
      catch (Exception ex)
      {
        return false;
          }

      finally
      {
        con.Close();
          }
     }
```

```
// Method use to obtain a single value from the data source

    public static string FetchScalar(string Query)
    {
      SqlCommand  cmd = new SqlCommand(Query, con);
      string s;
      try
      {
        con.Open();
        s = Convert.ToString(cmd.ExecuteScalar());
        return s;         }
      catch (Exception ex)
      {
        return  " ";         }
      finally
      {
        con.Close();         }
    }
  }          //End of DbAccess class.
}            //End of the namespace.
```

OleDb – Data Provider

It help us to connect to Ms Access Database.

For using OleDb – Data Provider

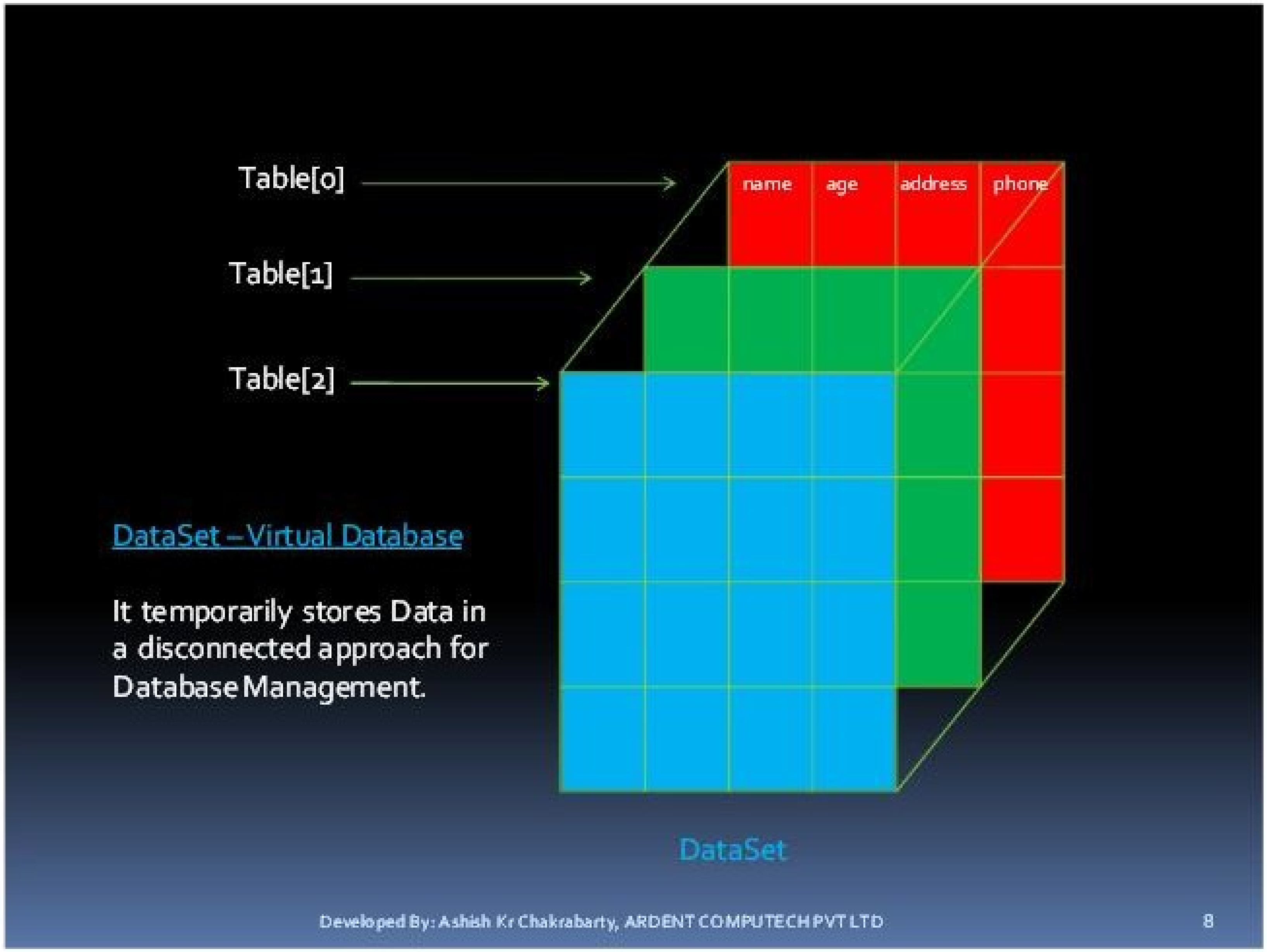
1. Change the namespace to - using System.Data.OleDb;

2. Change the DataProvider Object name with - OleDbConnection, OleDbAdapter , OleDbCommand etc.

DataSet

- Main memory data base
  - relational structure
  - object oriented interface
- DataSet consists of
  - collection of DataTables
  - collection of DataRelations
- DataTables consists of
  - collection of DataColumns (= schema definition)
  - collection of DataRows (= data)
  - DefaultView (DataView, see later)
- DataRelations
  - associate two DataTable objects
  - define ParentTable and ParentColumns
    and ChildTable and ChildColumns

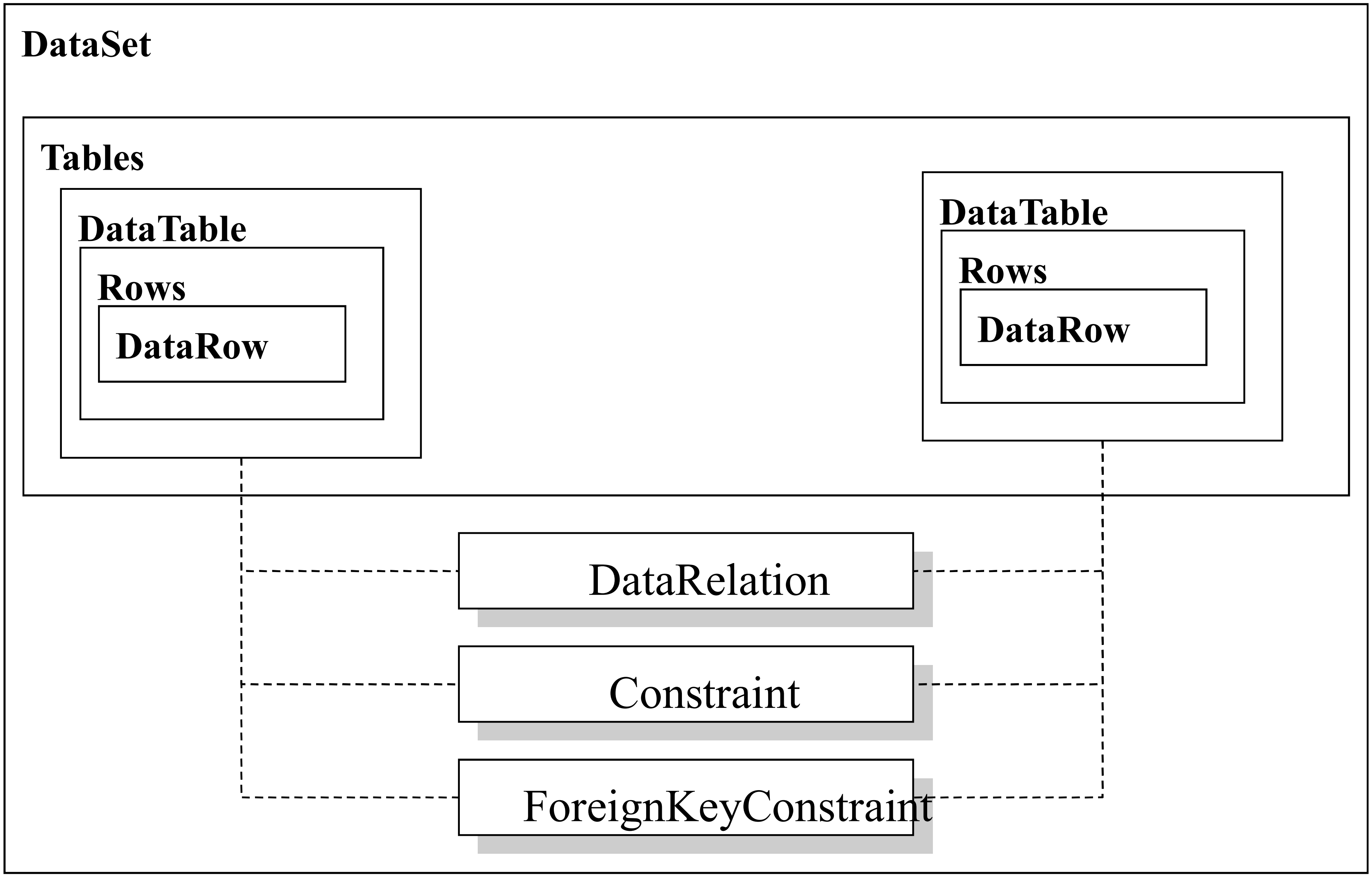
Dataset



DataSet – Virtual Database

It temporarily stores Data in a disconnected approach for Database Management.

The DataSet


DataSet
Tables
DataTable
Rows
DataRow
DataTable
Rows
DataRow
DataRelation
Constraint
ForeignKeyConstraint

Dataset

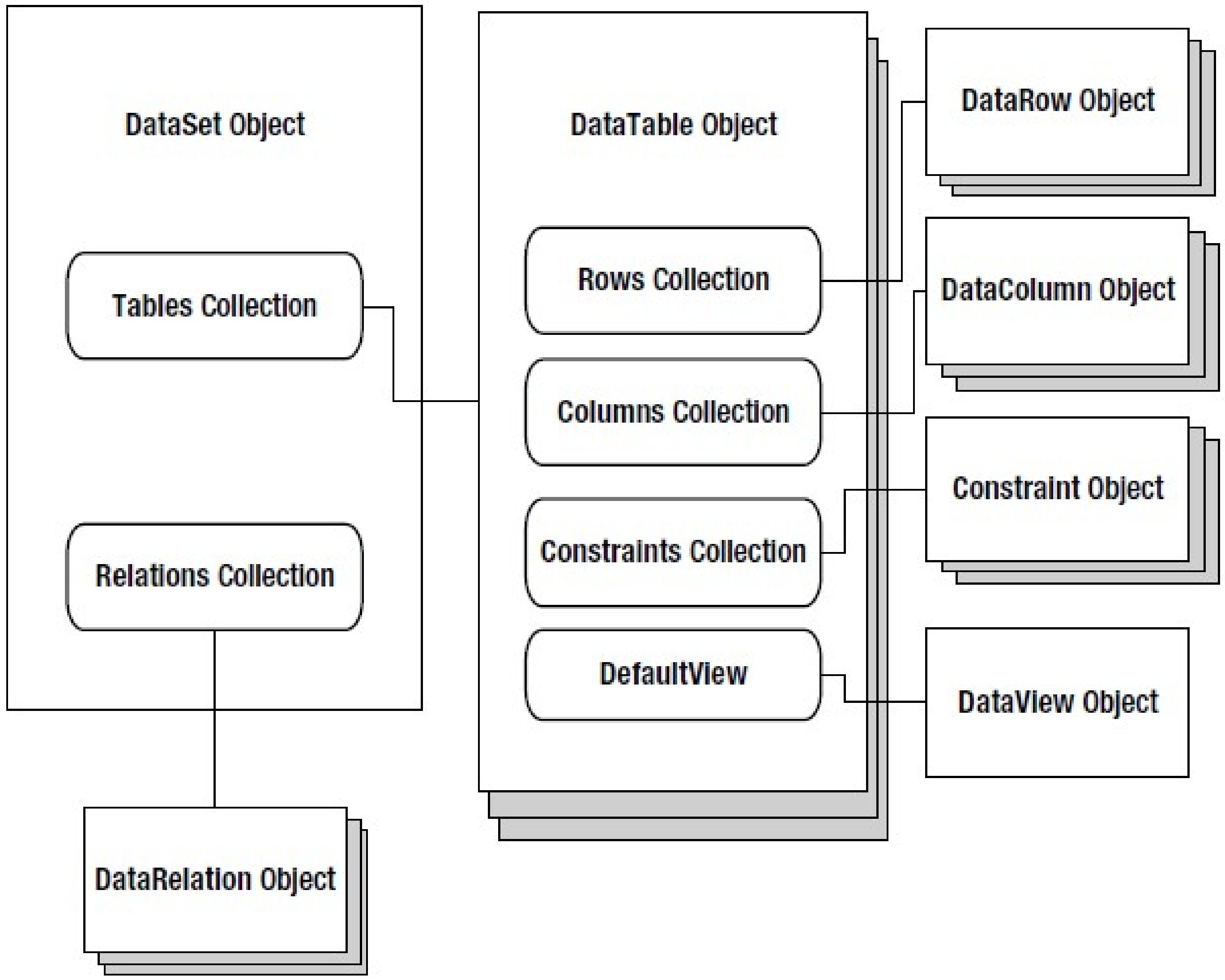
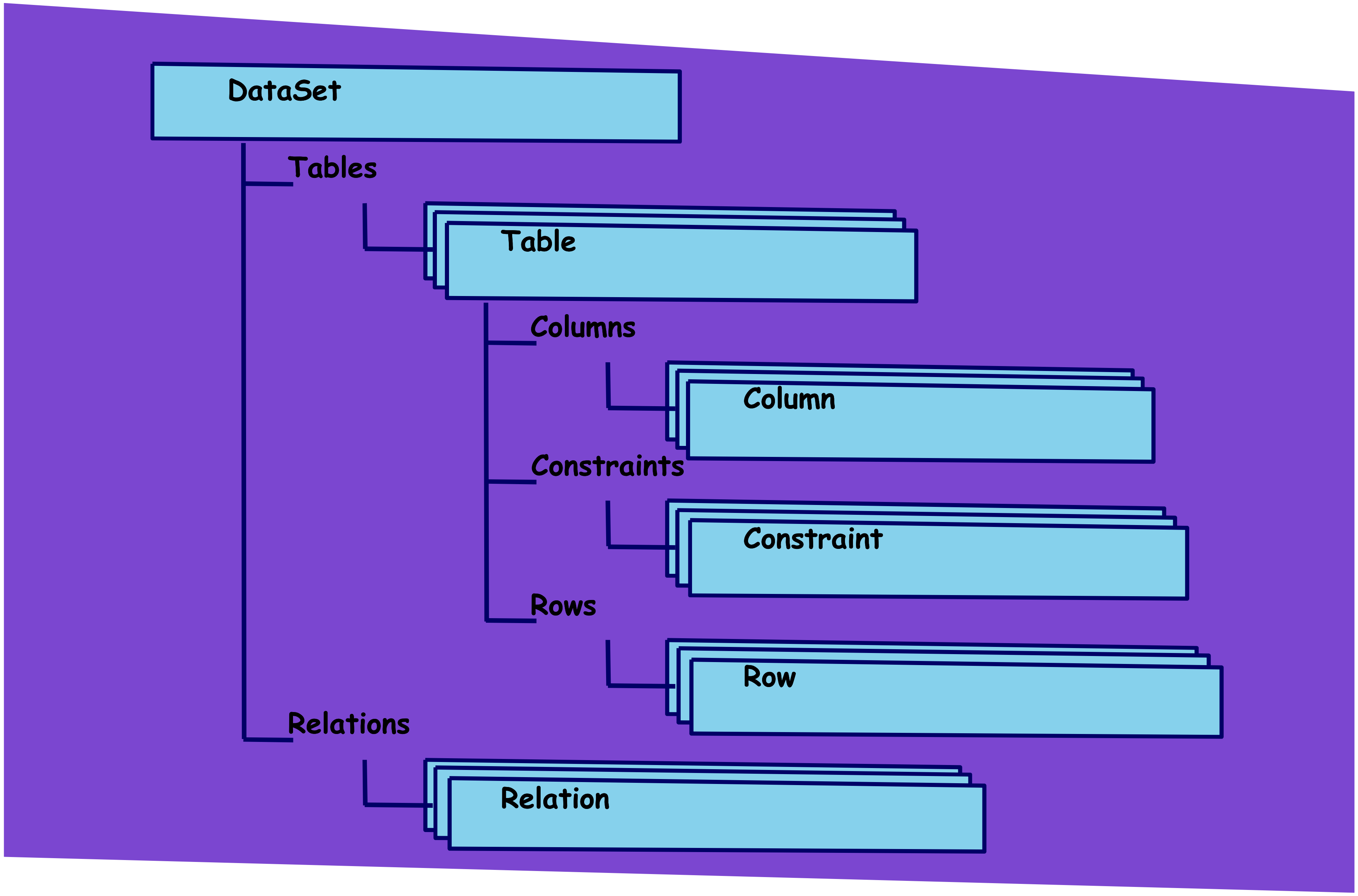
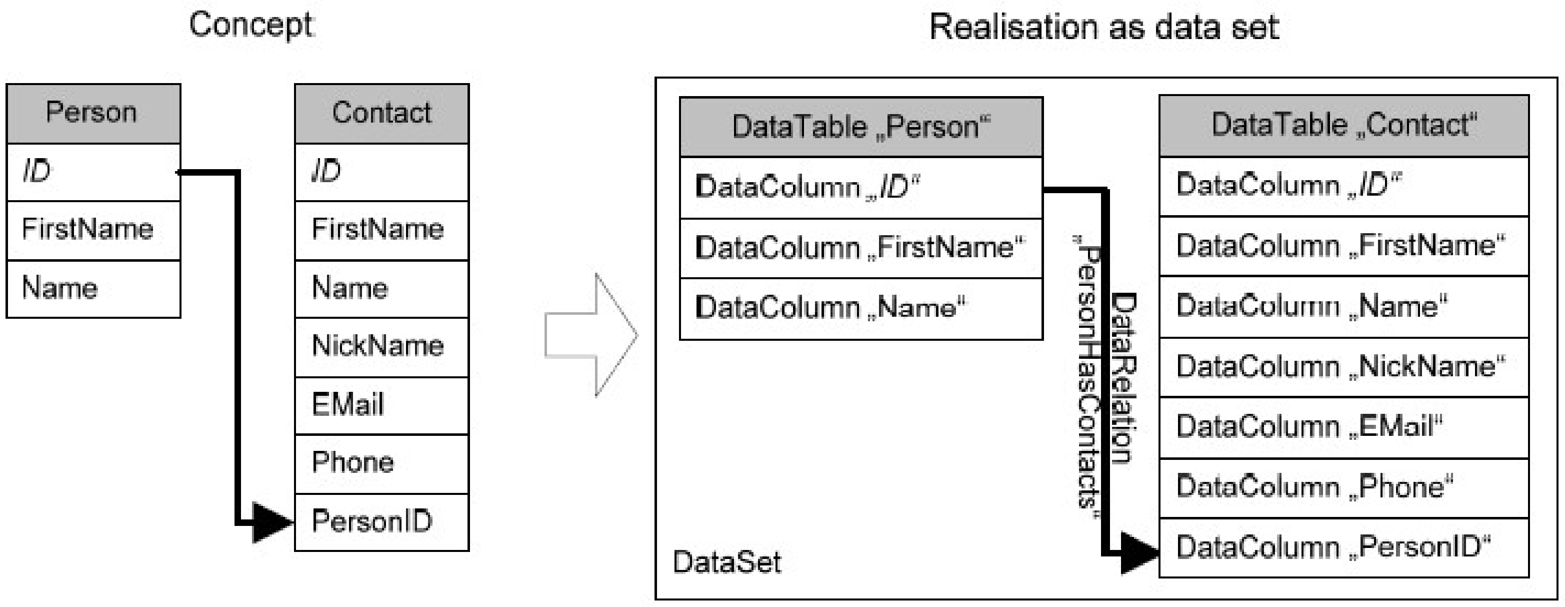


Figure 14-15. *The DataSet family of objects*

Data set


DataSet
Tables
Table
Columns
Column
Constraints
Constraint
Rows
Row
Relations
Relation

Example: Person Contacts



Implementation steps:

- Define schema
- Define data
- Access data

Building A DataSet – Create Table

- Create DataSet and DataTable "Person"

```
DataSet ds = new DataSet("PersonContacts");
DataTable personTable = new DataTable("Person");
```

- Define column "ID" and set properties

```
DataColumn col = new DataColumn();
col.DataType = typeof(System.Int64);
col.ColumnName = "ID";
col.ReadOnly = true;
col.Unique = true;                    // values must be unique
col.AutoIncrement = true;             // keys are assigned automatically
col.AutoIncrementSeed = -1;           // first key starts with -1
col.AutoIncrementStep = -1;           // next key = prev. key - 1
```

- Add column to table and set as primary key

```
personTable.Columns.Add(col);
personTable.PrimaryKey = new DataColumn[ ] { col };
```

Building A DataSet – Add Table

- Add table to DataSet

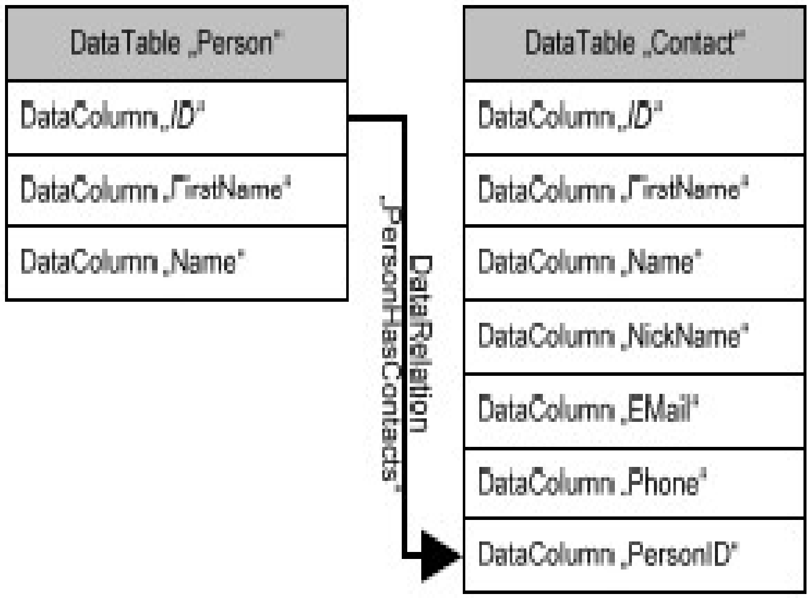
```
ds.Tables.Add(personTable);
```

- Create table `"Contact"` in similar way

```
DataTable contactTable = new DataTable("Contact");
...
ds.Tables.Add(contactTable);
```

Adding Relations

- Create relation PersonHasContacts
- and add it to the DataSet



```
DataColumn parentCol = ds.Tables["Person"].Columns["ID"];
DataColumn childCol = ds.Tables["Contact"].Columns["PersonID"];

DataRelation rel = new DataRelation("PersonHasContacts", parentCol, childCol);
ds.Relations.Add(rel);
```

Working with DataRows - Filling Data

- Create new row and assign column values

```
DataRow personRow = personTable.NewRow();
personRow[1] = "Wolfgang";
personRow["Name"] = "Beer";
```

- Add row to table "Person"

```
personTable.Rows.Add(personRow);
```

- Create and add row to table "Contact"

```
DataRow contactRow = contactTable.NewRow ();
contactRow[0] = "Wolfgang";
...
contactRow["PersonID"] = (long)personRow["ID"]; // defines relation
contactTable.Rows.Add (contactRow);
```

- Commit changes

```
ds.AcceptChanges();
```

Working with DataRows: Reading Data

- Iterate over all persons of personTable and put out the names

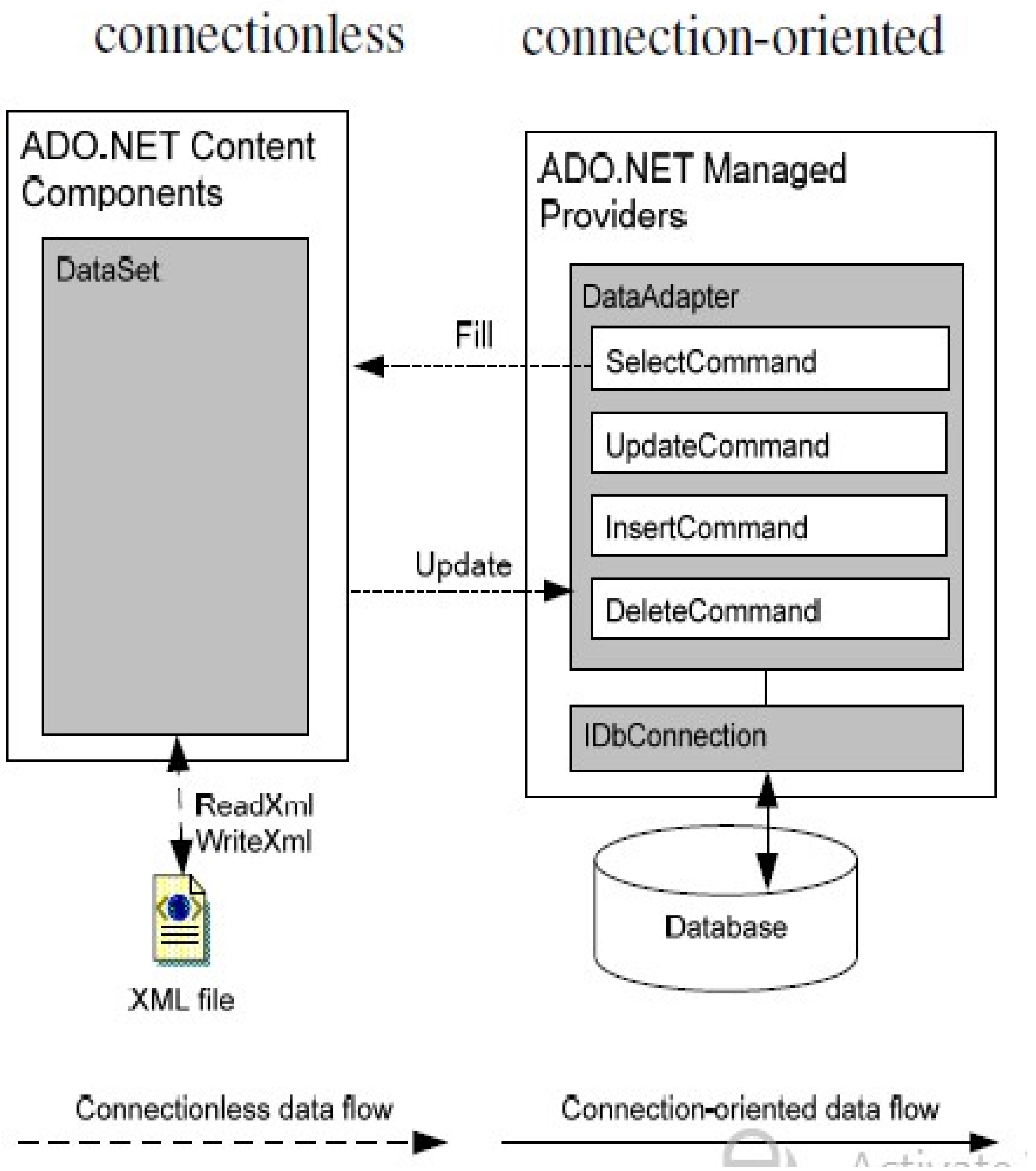
```
foreach (DataRow person in personTable.Rows) {
    Console.WriteLine("Contacts of {0}:", person["Name"]);
```

- Access contacts through relation "PersonHasContacts" and print out contacts

```
foreach (DataRow contact in person.GetChildRows("PersonHasContacts")) {
    Console.WriteLine("{0}, {1}: {2}", contact[0], contact["Name"], contact["Phone"] );
}
```

Filling DataSets with Data Adapters

- DataAdapter for connection to data source
  - Fill: Filling the DataSet
  - Update: Writing back changes
- DataAdapters use Command objects
  - SelectCommand
  - InsertCommand
  - DeleteCommand
  - UpdateCommand



DataAdapter: Loading Data

- Create DataAdapter object and set SelectCommand

```
IDbDataAdapter adapter = new OleDbDataAdapter();
OleDbCommand cmd = new OleDbCommand();
cmd.Connection = new OleDbConnection ("provider=SQLOLEDB; ..." );
cmd.CommandText = "SELECT * FROM Person";
adapter.SelectCommand = cmd;
```

- Read data from data source and fill DataTable "Person"

```
adapter.Fill(ds, "Person");
adapter.Fill(ds, new DataTable("Person"));
```

Only works when DataTable „Person“ already exists and is compatible to database table!

DataAdapter: Writing Back Changes (1)

- Changes are written back with Update method
  - IMPORTANT
    Do not call AcceptChanges() before, this will be called automatically
- Update-, Insert- and DeleteCommand define how changes are written
- CommandBuilder can create Update-, Insert- und DeleteCommand from SelectCommand automatically (in simple cases)
- Conflict management for updates:
  - comparison of data in DataTable and data source
  - in case of conflict DBConcurrencyException is thrown

DataAdapter: Writing Back Changes (2)

- Create DataAdapter with SELECT expression

```
OleDbConnection con = new OleDbConnection ("provider=SQLOLEDB; ...");
adapter = new OleDbDataAdapter("SELECT * FROM Person", con);
```

- Create update commands using CommandBuilder

```
OleDbCommandBuilder cmdBuilder = new OleDbCommandBuilder(adapter);
```

- Call Update and handle conflicts

```
try {
    adapter.Update(ds, tableName);
} catch (DBConcurrencyException) {
    // Handle the error, e.g. by reloading the DataSet
}
adapter.Dispose();
```

Data Binding

- Types of data binding → Single-Value, Repeated-Value
- How data binding works
- Using Single Value Data Binding
- Using Repeated Value Data Binding
    - Data binding with simple List control
    - Multiple binding
    - Data binding with a Dictionary collection
    - Data binding with ADO.NET
- Working with Data Source Controls

Data Binding

- **Data binding**, in the context of .NET, is the method by which controls on a user interface (UI) of a client application are configured to fetch from, or update data into, a data source, such as a database or XML document.
- The basic principle of data binding is this: you tell a control where to find your data and how you want it displayed, and the control handles the rest of the details.
- Data binding in the case of Desktop applications involves creating data connection between data source and the control
  - If user makes changes to the control on screen data, the changes are immediately reflected in the linked database.
  - If you made changes to the database, those changes are reflected in the bounded UI control automatically.

Data binding with ASP.NET is more complicated (because of web connections)

- ASP.NET data binding works in one direction only. Information moves from a data object into a control. Then the data objects are thrown away, and the page is sent to the client.
- If the user modifies the data in a data-bound control, your program can update the corresponding record in the database, but nothing happens automatically.
- The data controls of ASP.NET allows much powerful data binding.

Types of Data binding

- Single Value or Simple Data binding.
- Repeated Value or List Binding.

Single Value or Simple Data binding:

- You can use *single-value data binding to add information anywhere on an ASP.NET page. You can even place* information into a control property or as plain text inside an HTML tag.
- Instead, single-value data binding allows you to take a variable, a property, or an expression and insert it dynamically into a page.

Repeated Value or List Binding:

- *Repeated-value data binding allows you to display an entire table (or just a single field from a table). Unlike* single-value data binding, this type of data binding requires a special control that supports it.
  - **Controls used can be ListBox, CheckBoxList, GridView etc…**

Note: *You'll know that a control supports repeated-value data binding if it provides a DataSource property.*

Types of Data binding

- As with single-value binding, repeated-value binding doesn’t necessarily need to use data from a database, and it doesn’t have to use the ADO.NET objects. For example, you can use repeatedvalue binding to bind data from a collection or an array.
- To use single-value binding, you must insert a data-binding expression into the markup in the .aspx file (not the code-behind file). To use repeated-value binding, you must set one or more properties of a data control.
- Once you specify data binding, you need to activate it. You accomplish this task by calling the DataBind() method. The DataBind() method performs repeated-value data binding.
- you can also bind the whole page at once by calling the DataBind() method of the current Page object. Once you call this method, all the data binding expressions in the page are evaluated and replaced with the specified value.

Using Single Value Data Binding

- Single-value data binding is really just a different approach to dynamic text. To use it, you add special data-binding expressions into your .aspx files,

  <%# expression_goes_here %>

Examples:

1. <%# Country %> → Country is a protected variable.
   - ✓ When you call the DataBind() method for the page, this text will be replaced with the value for Country.
2. <%# Request.Browser.Browser %> → gives browser name
3. <%# 1 + (2 * 20) %> → gives 41
4. <%# GetUserName(ID) %>

Problems with single value data binding

- 1. Violation of code separation from Presentation concept of ASP.NET

  i.e putting code into a page's user interface

  2. Fragmenting code: if you use data binding to fill a control and also modify that control directly in code, data binding will not work properly.

  - If the page code changes, or a variable or function is removed or renamed, the corresponding data binding expression could stop providing valid information without any explanation or even an obvious error, resulting in maintenace problems.

Using Repeated Value Binding

- Repeated-value data binding works with the ASP.NET list controls.
- To use repeated-value binding, you link one of these controls to a data source.
- When you call DataBind(), the control automatically creates a full list by using all the corresponding values. This saves you from writing code that loops through the array or data table and manually adds elements to a control.
- Repeated-value binding can also simplify your life by supporting advanced formatting and template options that automatically configure how the data should look when it's placed in the control.
- List controls used in Repeated value binding:
  - ListBox
  - HtmlSelect
  - GridView, DetailsView, FormView, ListView

Multiple Binding

- binding the same data list object to multiple controls is called multiple binding.

Ex: Populating multiple list controls with the same data.

```
List<string> fruits = new List<string>();
    fruits.Add("Apple");
    fruits.Add("Grape");
    fruits.Add("Mango");
    fruits.Add("Orange");
    fruits.Add("PineApple");
    fruits.Add("Guava");
    //bind to ListBox
    lstItems.DataSource = fruits;
    //bind to dropdownlist
    dropdownLstItems.DataSource = fruits;
    //bind to checkboxlist
    chkBoxListItems.DataSource = fruits;
    //bind to html server select control
    Select1.DataSource = fruits;
    this.DataBind();
```

Data Binding with a Dictionary Collection

Ex:

```
// Use integers to index each item. Each item is a string.
Dictionary<int, string> fruit = new Dictionary<int, string>();
fruit.Add(1, "Kiwi");
fruit.Add(2, "Pear");
fruit.Add(3, "Mango");
fruit.Add(4, "Blueberry");
fruit.Add(5, "Apricot");
fruit.Add(6, "Banana");
fruit.Add(7, "Peach");
fruit.Add(8, "Plum");
// Define the binding for the list controls.
MyListBox.DataSource = fruit;
// Choose what you want to display in the list.
MyListBox.DataTextField = "Value";
// Activate the binding.
this.DataBind();
```

Using DataValueField Property

- Along with the DataTextField property, all list controls that support data binding also provide a DataValueField property, which adds the corresponding information to the value attribute in the control element.
- This allows you to store extra (undisplayed) information that you can access later.

Ex: MyListBox.DataTextField = "Value";
 MyListBox.DataValueField = "Key";

- <select name="MyListBox" id="MyListBox" >
- <option value="1">Kiwi</option>
- <option value="2">Pear</option>
- <option value="3">Mango</option>
- <option value="4">Blueberry</option>
- <option value="5">Apricot</option>
- <option value="6">Banana</option>
- <option value="7">Peach</option>
- <option value="8">Plum</option>
- </select>

Using DataValueField Property

```
 protected void MyListBox_SelectedIndexChanged(Object sender,
EventArgs e)
{
lblMessage.Text = "You picked: " + MyListBox.SelectedItem.Text;
lblMessage.Text += " which has the key: " + MyListBox.SelectedItem.Value;
}
```

DataBinding with ADO.NET

Binding DataSet to a List:

To fill a DataSet by hand, you need to follow several steps:

1. Create the DataSet.
2. Create a new DataTable and add it to the DataSet.Tables collection.
3. Define the structure of the table by adding DataColumn objects (one for each field) to the DataTable.Columns collection.
4. Supply the data. You can get a new, blank row that has the same structure as your DataTable by calling the DataTable.NewRow() method. You must then set the data in all its fields, and add the DataRow to the DataTable.Rows collection.

Data Source controls

- Data source controls allow you to create data-bound pages without writing any data access code at all.
- These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.
- The data source controls include any control that implements the IDataSource interface.
- You can find the data source controls in the Data tab of ToolBox in visual studio.
- **They retrieve data from a data source and supply it to bound controls**.
- ***They can update the data source when edits take place in the rich data controls like GridView and DetailsView.***
- Using SqlDataSource control you can connect to any MS SQL, ORACLE, OleDB or ODBC data sources. MS SQL is the default.
- Another important fact to understand about the data source controls is that when you bind more than one control to the same data source, you cause the query to be executed multiple times.

  <asp:SqlDataSource ProviderName="System.Data.SqlClient" ... />

To refer connection string in .aspx file:

<%$ ConnectionStrings:[NameOfConnectionString] %>

Ex:

<asp:SqlDataSource ConnectionString="<%$ ConnectionStrings:Northwind %>" ... />

Data Source controls

- ASP.NET includes **data source controls** that allow you to work with different types of **data sources** such as a database, an XML file, or a middle-tier business object. **Data source controls** connect to and retrieve **data** from a **data source** and make it available for other **controls** to bind to, without requiring code.

Data Source controls

- You can use each SqlDataSource control you create to retrieve a single query. Optionally, you can also add corresponding commands for deleting, inserting, and updating rows.
- SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand properties of SqlDataSource is used to specify the SQL statements.
- The SqlDataSource supports automatic caching if you set EnableCaching to true.
- It’s also important to remember that data binding is performed at the end of your web page processing, just before the page is rendered. This means the Page.Load event will fire, followed by any control events, followed by the Page.PreRender event. Only then will the data binding take place.

SqlDataSource that defines a Select command for retrieving product information from the Products table:

```
<asp:SqlDataSource ID="sourceProducts" runat="server"
ConnectionString="<%$ ConnectionStrings:Northwind %>"
SelectCommand="SELECT ProductName, ProductID FROM Products"
/>
```

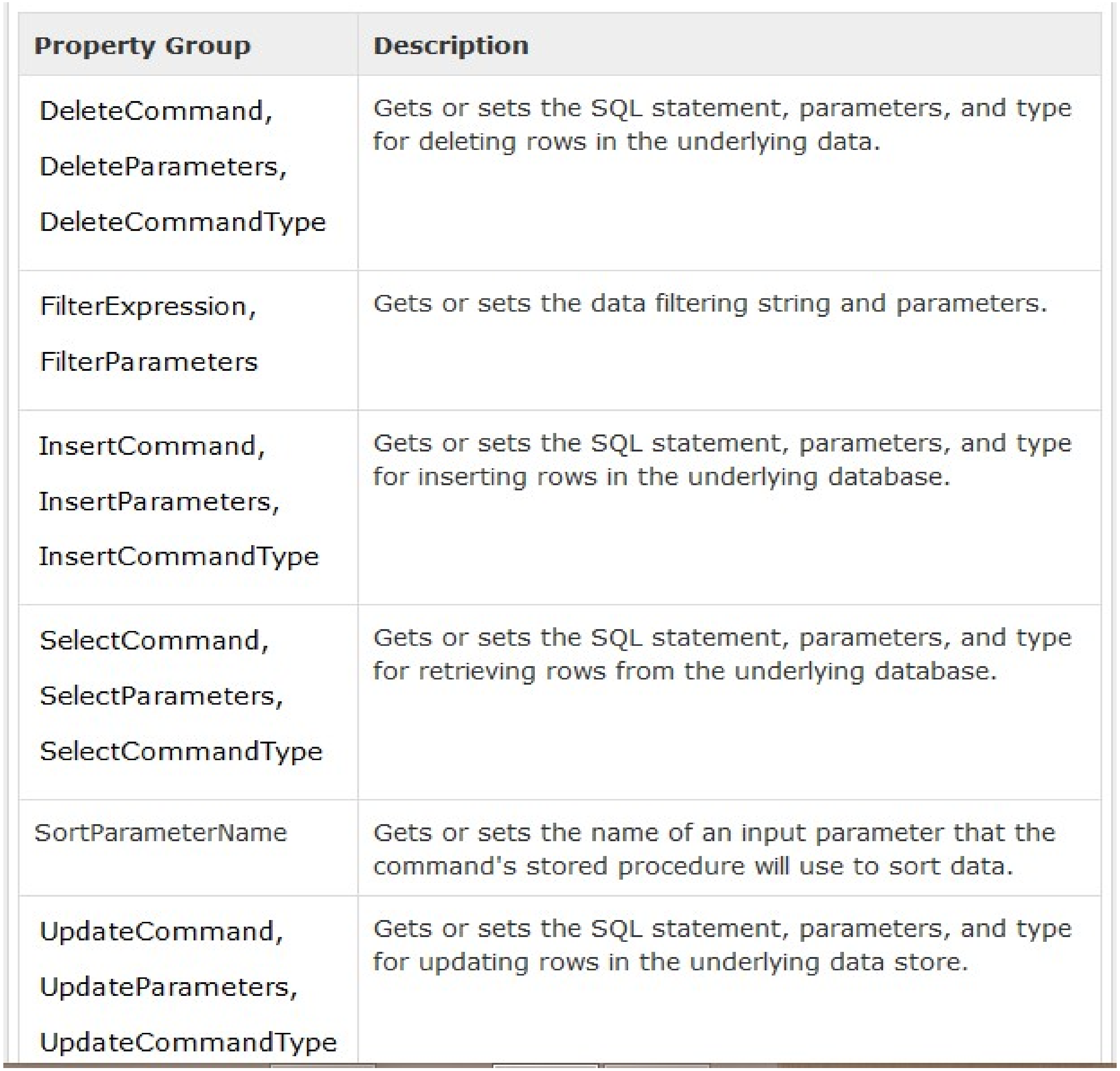
A DropDownList control that lists all the products:

```
<asp:DropDownList ID="lstProduct" runat="server"
    AutoPostBack="True“ DataSourceID="sourceProducts"
    DataTextField="ProductName“ DataValueField="ProductID" />
```

Data Source controls

- .NET framework includes the following data sources:

Data Source control	Description
SqlDataSource	This data source allows you to connect to any data source that has an ADO.NET data provider. This includes SQL Server, Oracle, and OLE DB or ODBC data sources.
AccessDataSource	This data source allows you to read and write the data in an Access database file (.mdb).
ObjectDataSource	This data source allows you to connect to a custom data access class.
XmlDataSource	This data source allows you to connect to an XML file.
SiteMapDataSource	This data source allows you to connect to a .sitemap file that describes the navigational structure of your website
EntityDataSource	This data source allows you to query a database by using the LINQ to Entities feature.
LinqDataSource	This data source allows you to query a database by using the LINQ to SQL feature, which is a similar (but somewhat less powerful) predecessor to LINQ to Entities.



Property Group	Description
DeleteCommand, DeleteParameters, DeleteCommandType	Gets or sets the SQL statement, parameters, and type for deleting rows in the underlying data.
FilterExpression, FilterParameters	Gets or sets the data filtering string and parameters.
InsertCommand, InsertParameters, InsertCommandType	Gets or sets the SQL statement, parameters, and type for inserting rows in the underlying database.
SelectCommand, SelectParameters, SelectCommandType	Gets or sets the SQL statement, parameters, and type for retrieving rows from the underlying database.
SortParameterName	Gets or sets the name of an input parameter that the command's stored procedure will use to sort data.
UpdateCommand, UpdateParameters, UpdateCommandType	Gets or sets the SQL statement, parameters, and type for updating rows in the underlying data store.

SqlDataSource class

Properties	Description
CacheDuration	Gets or sets the length of time, in seconds, that the data source control caches data that is retrieved by the Select method.
ConnectionString	Gets or sets the ADO.NET provider–specific connection string that the SqlDataSource control uses to connect to an underlying database.
DataSourceMode	Gets or sets the data retrieval mode that the SqlDataSource control uses to fetch data.
DeleteCommand	Gets or sets the SQL string that the SqlDataSource control uses to delete data from the underlying database.
DeleteCommandType	Gets or sets a value indicating whether the text in the DeleteCommand property is an SQL statement or the name of a stored procedure.
DeleteParameters	Gets the parameters collection that contains the parameters that are used by the DeleteCommand property from the SqlDataSourceView object that is associated with the SqlDataSource control.

SqlDataSource class

Properties	Description
EnableCaching	Gets or sets a value indicating whether the SqlDataSource control has data caching enabled.
EnableTheming	Gets a value indicating whether this control supports themes.
EnableViewState	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
FilterExpression	Gets or sets a filtering expression that is applied when the Select method is called.
InsertCommand	Gets or sets the SQL string that the SqlDataSource control uses to insert data into the underlying database.
InsertCommandType	Gets or sets a value indicating whether the text in the InsertCommand property is an SQL statement or the name of a stored procedure.

SqlDataSource class

Properties	Description
InsertParameters	Gets the parameters collection that contains the parameters that are used by the InsertCommand property from the SqlDataSourceView object that is associated with the SqlDataSource control.
IsViewStateEnabled	Gets a value indicating whether view state is enabled for this control.
Parent	Gets a reference to the server control's parent control in the page control hierarchy.
SelectCommand	Gets or sets the SQL string that the SqlDataSource control uses to retrieve data from the underlying database.
SelectCommandType	Gets or sets a value indicating whether the text in the SelectCommand property is an SQL query or the name of a stored procedure.
SelectParameters	Gets the parameters collection that contains the parameters that are used by the SelectCommand property from the SqlDataSourceView object that is associated with the SqlDataSource control.

SqlDataSource class

Properties	Description
UpdateCommand	Gets or sets the SQL string that the SqlDataSource control uses to update data in the underlying database.
UpdateCommandType	Gets or sets a value indicating whether the text in the UpdateCommand property is an SQL statement or the name of a stored procedure.
UpdateParameters	Gets the parameters collection that contains the parameters that are used by the UpdateCommand property from the SqlDataSourceView control that is associated with the SqlDataSource control.

SqlDataSource class

Methods	Description
DataBind()	Binds a data source to the invoked server control and all its child controls.
Delete()	Performs a delete operation using the DeleteCommand SQL string and any parameters that are in the DeleteParameters collection.
Focus()	Sets input focus to the control.
Insert()	Performs an insert operation using the InsertCommand SQL string and any parameters that are in the InsertParameters collection.
Update()	Performs an update operation using the UpdateCommand SQL string and any parameters that are in the UpdateParameters collection.

ObjectDataSource

specify SelectMethod="GetStudents" TypeName="Student“

Where GetStudents( ) is the method of Student class.

Data controls

Rich data controls are:

- **GridView**
- **DetailsView**
- **FormView**
- **ListView**

GridView: *The GridView is an all-purpose grid control for showing large tables of* information. The GridView is the heavyweight of ASP.NET data controls.

DetailsView: *The DetailsView is ideal for showing a single record at a time, in a table that* has one row per field. The DetailsView also supports editing.

FormView: *Like the DetailsView, the FormView shows a single record at a time and* supports editing. The difference is that the FormView is based on templates, which allow you to combine fields in a flexible layout that doesn’t need to be table based.

ListView: *The ListView plays the same role as the GridView—it allows you to show* multiple records. The difference is that the ListView is based on templates.

Data controls

GridView Control:

- ✓ **GridView** → Shows the tabular data in a Grid.
- ✓ The GridView control is used to display the values of a data source in a table. Each column represents a field where each row represents a record.
- ✓ The GridView control provides many built-in capabilities that allow the user to sort, update, delete, select and page through items in the control.
- ✓ The GridView control can be bound to a data source control.

GridView features:

- ➢ Improved data source binding capabilities
- ➢ Tabular rendering – displays data as a table
- ➢ Built-in sorting capability
- ➢ Built-in select, edit and delete capabilities
- ➢ Built-in paging capability
- ➢ Built-in row selection capability
- ➢ Multiple key fields
- ➢ Programmatic access to the GridView object model to dynamically set properties, handle events and so on
- ➢ Richer design-time capabilities
- ➢ Control over Alternate item, Header, Footer, Colors, font, borders, and so on.
- ➢ Slow performance as compared to Repeater and DataList control .

Populating columns of GridView

1. By setting AutoGenerateColumns property to true.

Disadv: it is not possible to explicity say, which properties should be displayed as columns, what the HeaderText or width of each column should be.

Ex:

```
<asp:GridView ID="gvUsers" runat="server"
        AutoGenerateColumns="true"></asp:GridView>
```

2. By using BoundField:

This allows you to create the columns allows to explicitly define, which columns should be displayed, how they look and in which order they are displayed.

✓ In order to specify the columns we need to set the **AutoGeneratedColumns** property to false.

Ex:

```
<asp:GridView ID="gvUsers" runat="server" AutoGenerateColumns="false">
  <Columns>
    <asp:BoundField HeaderText="ID" DataField="IDUser" ItemStyle-Width="50"/>
    <asp:BoundField HeaderText="Name" DataField="Name" ItemStyle-Width="200"/>
    <asp:BoundField HeaderText="Username" DataField="UserName" ItemStyle-Width="200"/>
  </Columns>
</asp:GridView>
```

Column types

Column type	Description
BoundField	This column displays text from a field in the data source.
ButtonField	This column displays a button in this grid column.
CheckBoxField	This column displays a check box in this grid column. It’s used automatically for True / false fields.
CommandField	This column provides selection or editing buttons.
HyperLinkField	This column displays its contents (a field from the data source or static text) as a hyperlink.
ImageField	This column displays image data from a binary field.
TemplateField	This column allows you to specify multiple fields, custom controls, and arbitrary HTML using a custom template.

Populating columns of GridView

Ex:

```
<asp:BoundField DataField = "ProductID" HeaderText = "ID" />
```

Configuring columns of GridView using BoundField properties

BoundField properties:

Properties	Description
DataField	Identifies the field (by name) that you want to display in this column
DataFormatString	Formats the field. This is useful for getting the right representation of numbers and dates.
FooterText, HeaderText, and HeaderImageUrl	Sets the text in the header and footer region of the grid if this grid has a header (GridView.ShowHeader is true) and footer (GridView.ShowFooter is true).
ReadOnly	If true, it prevents the value for this column from being changed in edit mode. No edit control will be provided**. Primary key fields are often read-only.**
InsertVisible	If true, it prevents the value for this column from being set in insert mode

Configuring columns of GridView using BoundField properties

BoundField properties:

Properties	Description
Visible	If false, the column won’t be visible in the page.
SortExpression	Sorts your results based on one or more columns.
HtmlEncode	If true (the default), all text will be HTML encoded to prevent special characters from mangling the page.
NullDisplayText	Displays the text that will be shown for a null value.
ConvertEmptyStringToNull	If true, converts all empty strings to null values.
ControlStyle, HeaderStyle, FooterStyle, and ItemStyle	Configures the appearance for just this column, overriding the styles for the row.

Formatting GridView

- Each BoundField column provides a DataFormatString property you can use to configure the appearance of numbers and dates using a *format string.*
- Format strings generally consist of a placeholder and a format indicator, which are wrapped inside curly brackets.
- Ex: {0:C} → Currency format

Ex:

```
<asp:BoundField DataField = "UnitPrice" HeaderText = "Price"
DataFormatString = "{0:C}" />
```

Time & Date Format Strings:

Type	Format String	Syntax	Example
Short Date	{0:d}	M/d/yyyy	10/30/2012
Long Date	{0:D}	dddd, MMMM dd, yyyy	Monday, January 30, 2012
Long Date and Short Time	{0:f }	dddd, MMMM dd, yyyy HH:mm aa	Monday, January 30, 2012 10:00 AM
Long Date and Long Time	{0:F}	dddd, MMMM dd, yyyy HH:mm:ss aa	Monday, January 30, 2012 10:00:23 AM

GridView styles

Style	Description
HeaderStyle	Configures the appearance of the header row that contains column titles, if you’ve chosen to show it (if ShowHeader is true).
RowStyle	Configures the appearance of every data row.
AlternatingRowStyle	If set, applies additional formatting to every other row
SelectedRowStyle	Configures the appearance of the row that’s currently selected.
EditRowStyle	Configures the appearance of the row that’s in edit mode. This formatting acts in addition to the RowStyle formatting.
EmptyDataRowStyle	Configures the style that’s used for the single empty row in the special case where the bound data object contains no rows.
FooterStyle	Configures the appearance of the footer row at the bottom of the GridView, if you’ve chosen to show it
PagerStyle	Configures the appearance of the row with the page links, if you’ve enabled paging (set AllowPaging to true).

Styles

EX:

```
<RowStyle BackColor = "#E7E7FF" ForeColor = "#4A3C8C" />
<HeaderStyle BackColor = "#4A3C8C" Font-Bold = "True" ForeColor = "#F7F7F7" />
```

EX:

```
<asp:BoundField DataField = "ProductName" HeaderText = "Product Name">
    <ItemStyle BackColor = "#E7E7FF" ForeColor = "#4A3C8C" />
    <HeaderStyle BackColor = "#4A3C8C" Font-Bold = "True" ForeColor = "#F7F7F7" />
</asp:BoundField>
```

Using a Data Field as a Select Button

- You don’t need to create a new column to support row selection. Instead, you can turn an existing column into a link.

- To use this technique use a add a ButtonField column. Then, set the DataTextField to the name of the field you want to use.

EX: <asp:ButtonField ButtonType = "Button" DataTextField = "ProductID" />

Sorting & Paging the GridView

- To enable sorting, you must set the GridView.AllowSorting property to true. Next, you need to define a SortExpression for each column that can be sorted.
- To use automatic paging, you need to set AllowPaging to true (which shows the page controls), and you need to set PageSize to determine how many rows are allowed on each page

Paging with the GridView

Ex:

```
<asp:GridView ID = "GridView1" runat = "server" DataSourceID = "sourceProducts"
PageSize = "10" AllowPaging = "True" . . .>
. . .
</asp:GridView>
```

- Set **GridView.EnablePersistedSelection** property to true to avoid the same row position from being selected as you move from one page to another.

GridView templates

- The TemplateField allows you to define a completely customized *template for a column. Inside the template* you can add control tags, arbitrary HTML elements, and data binding expressions.
- you want to create a column that combines the in-stock, on-order, and reorder level information for a product using ItemTemplate as shown below:

```
<asp:TemplateField HeaderText = "Status">
    <ItemTemplate>
        <b > In Stock:</b>
        <%# Eval("UnitsInStock") % > <br />
        <b > On Order:</b>
        <%# Eval("UnitsOnOrder") % > <br />
        <b > Reorder:</b>
        <%# Eval("ReorderLevel") %>
    </ItemTemplate>
</asp:TemplateField>
```

Paging member of the GridView

Property	Description
AllowPaging	Enables or disables the paging of the bound records. It is false by default
PageSize	Gets or sets the number of items to display on a single page of the grid. The default value is 10.
PageIndex	Gets or sets the zero-based index of the currently displayed page, if paging is enabled.
PagerSettings	Provides a PagerSettings object that wraps a variety of formatting options for the pager controls. These options determine where the paging controls are shown and what text or images they contain.
PagerStyle	Provides a style object you can use to configure fonts, colors, and text alignment for the paging controls.
PageIndexChanging and PageIndexChanged events	Occur when one of the page selection elements is clicked, just before the PageIndex is changed (PageIndexChanging) and just after (PageIndexChanged).

TemplateField templates

Mode	Description
HeaderTemplate	Determines the appearance and content of the header cell.
FooterTemplate	Determines the appearance and content of the footer cell (if you set ShowFooter to true).
ItemTemplate	Determines the appearance and content of each data cell.
AlternatingItemTemplate	Determines the appearance and content of even-numbered rows.
EditItemTemplate	Determines the appearance and controls used in edit mode.
InsertItemTemplate	Determines the appearance and controls used in edit mode. The GridView doesn't support this template, but the DetailsView and FormView controls do.

DetailsView control

- ✓ The DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control.
- ✓ Unlike the GridView control, the DetailsView control displays one row from a data source at a time by rendering an HTML table.
- ✓ The DetailsView supports both declarative and programmatic data binding.
- ✓ The DetailsView control is often used in master-detail scenarios where the selected record in a master control determines the record to display in the DetailsView control. It shows the details for the row in a separate space.
- ✓ We can provide styles or CSS for customizing the appearance of the DetailsView.
- ✓ By default displays information in two columns.

drawback: In pager navigation, whole set of records are retrieved from database even though one record data is displayed in the control.

Features of DetailsView control:

- ➢ Tabular rendering
- ➢ Supports column layout, by default two columns at a time
- ➢ Optional support for paging and navigation.
- ➢ Built-in support for data grouping
- ➢ Built-in support for edit, insert and delete capabilities

DetailsView control

- ✓ Set AutoGenerateRows to false to stop automatic generation of rows. Then you can display only the columns you want in the DetailsView.
- ✓ You can display, page, edit, insert, and delete database records with the DetailsView.
- ✓ If you need more control over the appearance of the DetailsView, including the particular order in which columns are displayed, then you can use fields with the DetailsView control.
- **BoundField**—Enables you to display the value of a data item as text.
- **CheckBoxField**—Enables you to display the value of a data item as a check box.
- **CommandField**—Enables you to display links for editing, deleting, and selecting rows.
- **ButtonField**—Enables you to display the value of a data item as a button (image button, link button, or push button).
- **HyperLinkField**—Enables you to display the value of a data item as a link.
- **ImageField**—Enables you to display the value of a data item as an image.
- **TemplateField**—Enables you to customize the appearance of a data item.
- ✓

DetailsView control

```
<asp:DetailsView ID = "DetailsView1" runat = "server" AutoGenerateRows = "False"
DataSourceID = "sourceProducts">
        <Fields>
        <asp:BoundField DataField = "ProductID" HeaderText = "ProductID"
        ReadOnly = "True" />
        <asp:BoundField DataField = "ProductName" HeaderText = "ProductName" />
        <asp:BoundField DataField = "SupplierID" HeaderText = "SupplierID" />
        <asp:BoundField DataField = "CategoryID" HeaderText = "CategoryID" />
        <asp:BoundField DataField = "QuantityPerUnit" HeaderText = "QuantityPerUnit" />
        <asp:BoundField DataField = "UnitPrice" HeaderText = "UnitPrice" />
        <asp:BoundField DataField = "UnitsInStock" HeaderText = "UnitsInStock" />
        <asp:BoundField DataField = "UnitsOnOrder" HeaderText = "UnitsOnOrder" />
        <asp:BoundField DataField = "ReorderLevel" HeaderText = "ReorderLevel" />
        <asp:CheckBoxField DataField = "Discontinued" HeaderText = "Discontinued" />
        </Fields>
. . .
</asp:DetailsView>
```

DetailsView properties

- ✓ AutoGenerateDeleteButton, AutoGenerateEditButton, AutoGenerateInsertButton propertie are used for enabling delete/edit/insert for detailsview.
- ✓ AutoGenerateEditButton → Gets or sets a value indicating whether the built-in controls to edit the current record are displayed in a DetailsView control.
- ✓ AutoGenerateInsertButton → Gets or sets a value indicating whether the built-in controls to insert a new record are displayed in a DetailsView control.
- ✓ AutoGenerateRows → Gets or sets a value indicating whether row fields for each field in the data source are automatically generated and displayed in a DetailsView control.
- ✓ BackImageUrl → Gets or sets the URL to an image to display in the background of a DetailsView control.
- ✓ DataMember → Gets or sets the name of the list of data that the data-bound control binds to, in cases where the data source contains more than one distinct list of data items.
- ✓ DataSourceID → Gets or sets the ID of the control from which the data-bound control retrieves its list of data items.
- ✓ **Hyperlinks are displayed at the bottom of the control.**

DetailsView control styles

- ✓ **AlternatingRowStyle** → allows you to set the appearance of the alternating data rows in a DetailsView control.
- ✓ **CommandRowStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of a command row in a DetailsView control.
- ✓ **EditRowStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the data rows when a DetailsView control is in edit mode.
- ✓ **EmptyDataRowStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the empty data row displayed when the data source bound to a DetailsView control does not contain any records.
- ✓ **FieldHeaderStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the header column in a DetailsView control.
- ✓ **FooterStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the footer row in a DetailsView control.
- ✓ **HeaderStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the header row in a DetailsView control.
- ✓ **InsertRowStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the data rows in a DetailsView control when the DetailsView control is in insert mode.
- ✓ **PagerStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the pager row in a DetailsView control.
- ✓ **RowStyle** → Gets a reference to the TableItemStyle object that allows you to set the appearance of the data rows in a DetailsView control.

FormView

- ✓ **Requires Templates**
- ✓ **Displays columns without a table**
- ✓ **Templates supported by FormView control:**
  - ItemTemplate
  - EditItemTemplate
  - InsertItemTemplate
  - FooterTemplate
  - HeaderTemplate
  - EmptyDataTemplate
  - PagerTemplate

FormView

You can use FormView to display multiple item values in a single value:

Ex:

```
<asp:FormView ID = "FormView1" runat = "server" DataSourceID =
   "sourceProducts">
<ItemTemplate>
<b > In Stock:</b>
<%# Eval("UnitsInStock") %>
<br />
<b > On Order:</b>
<%# Eval("UnitsOnOrder") %>
<br />
<b > Reorder:</b>
<%# Eval("ReorderLevel") %>
<br />
</ItemTemplate>
</asp:FormView>
```