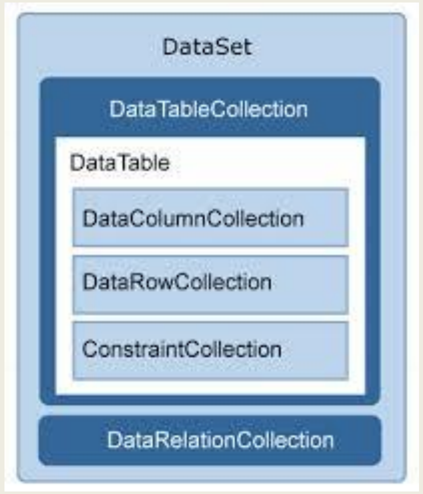
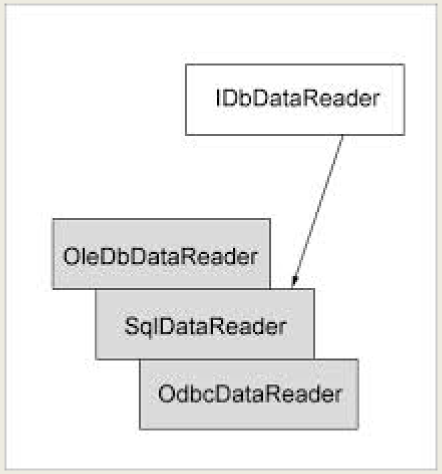
Accessing Data using ADO.NET

Connected Mode (DataReader), Disconnected mode (Dataset)





Abdallah Moujahid
PMP®, COBIT® V5, ITIL® V3, ISO27002
Abdallah.moujahid@uic.ac.ma

Introduction

- Before .NET, developers used data access technologies such as ODBC, OLE DB, and ActiveX Data Object(ADO).

- With the introduction of .NET, Microsoft created a new way to work with data, called ADO.NET.

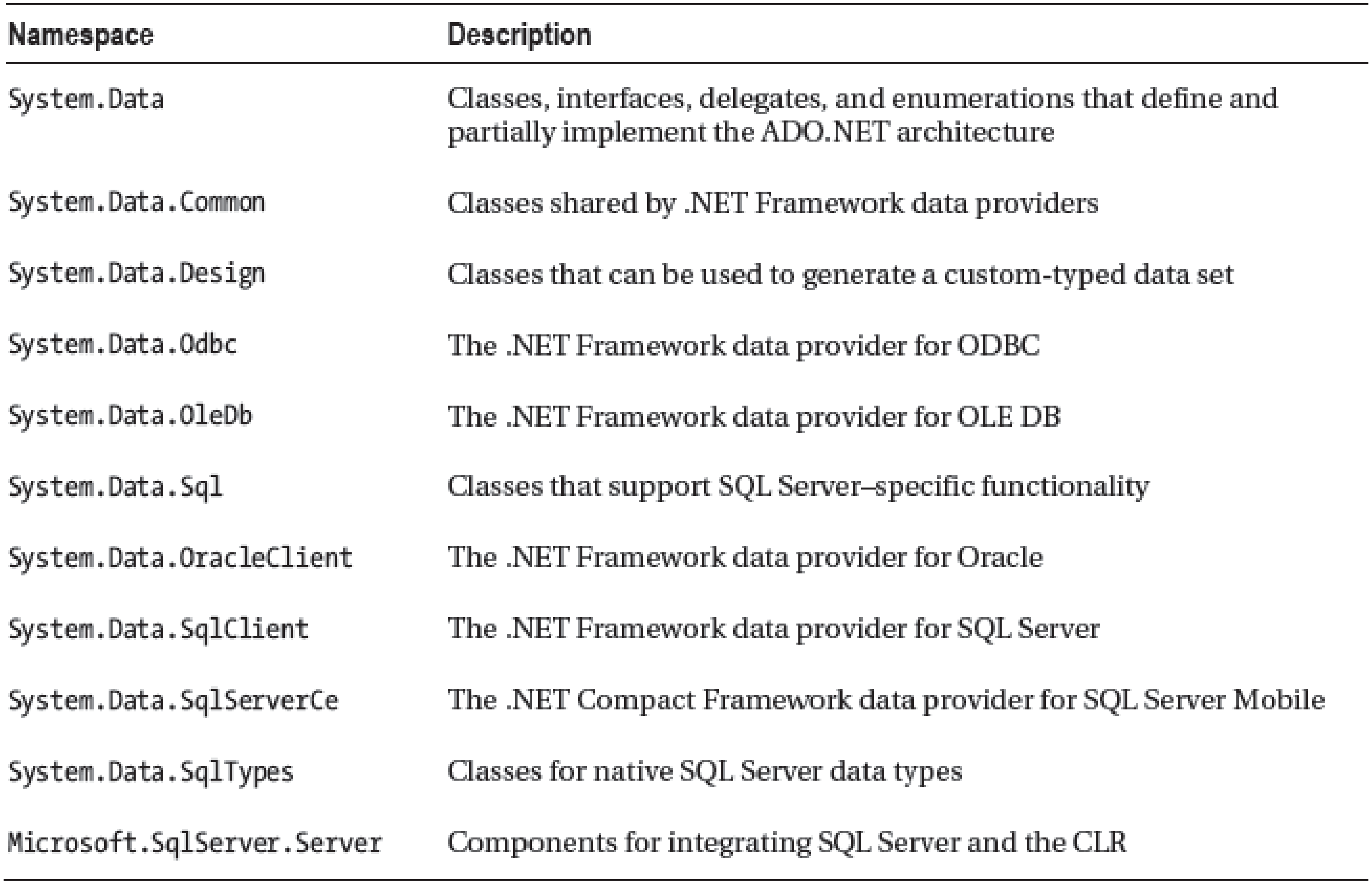
- ADO.NET is a set of classes exposing data access services to .NET programmers, providing a rich set of components for creating distributed, data-sharing applications.

- **ADO.NET is an integral part of the .NET Framework and provides access to relational, XML, and application data.**

Introductions (Cont.)

- ADO.NET classes can be found in **System.Data.dll**.
- This technology supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, and Internet browsers.
- **Hence, ADO.NET helps connect the UI, or presentation layer, of your application with the data source or database.**

ADO.NET and .NET Base Class Library



Namespace	Description
System.Data	Classes, interfaces, delegates, and enumerations that define and partially implement the ADO.NET architecture
System.Data.Common	Classes shared by .NET Framework data providers
System.Data.Design	Classes that can be used to generate a custom-typed data set
System.Data.Odbc	The .NET Framework data provider for ODBC
System.Data.OleDb	The .NET Framework data provider for OLE DB
System.Data.Sql	Classes that support SQL Server–specific functionality
System.Data.OracleClient	The .NET Framework data provider for Oracle
System.Data.SqlClient	The .NET Framework data provider for SQL Server
System.Data.SqlServerCe	The .NET Compact Framework data provider for SQL Server Mobile
System.Data.SqlTypes	Classes for native SQL Server data types
Microsoft.SqlServer.Server	Components for integrating SQL Server and the CLR

ADO.NET Architecture

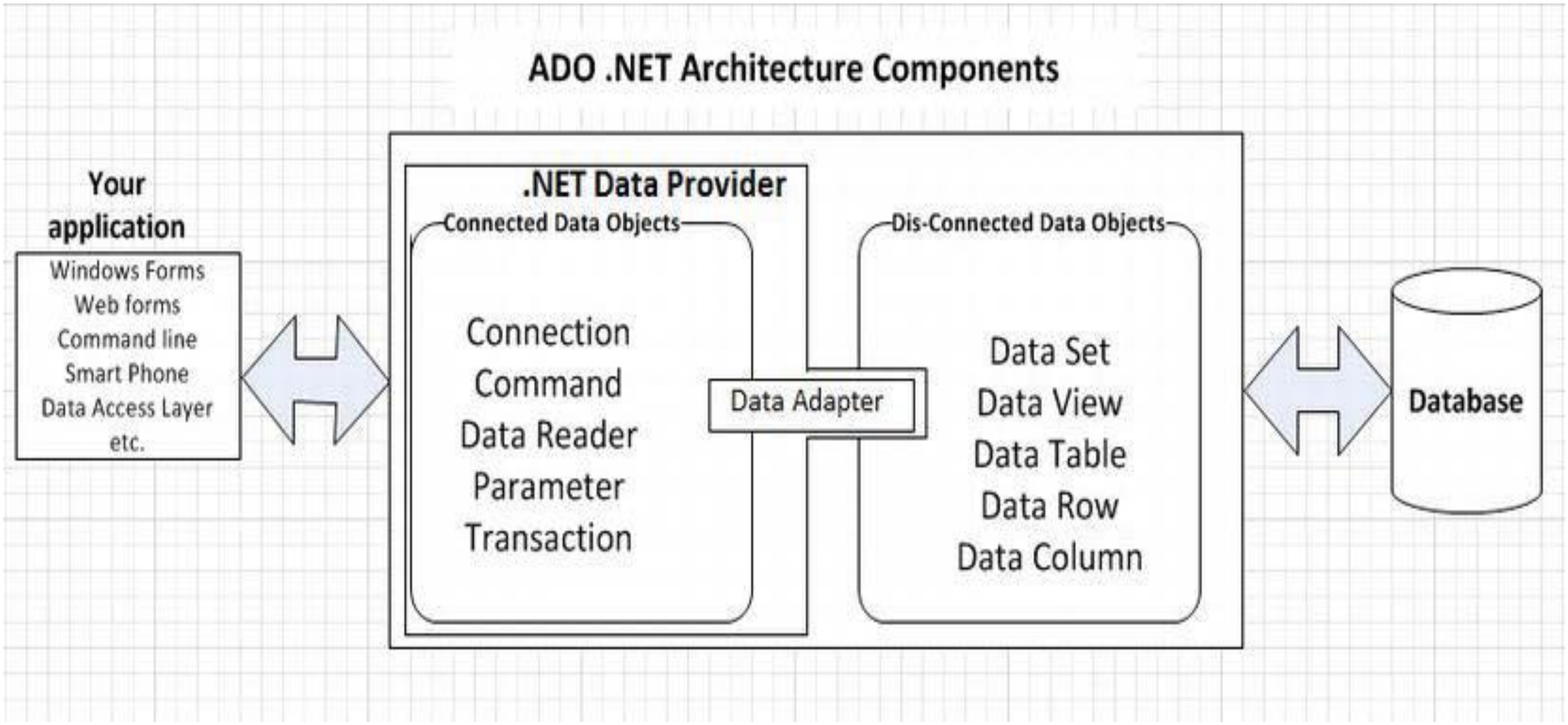
- ADO.NET offers two types of architectural components to build data-centric applications: **Connected** and **Disconnected**.

- Within Microsoft .NET Framework, ADO.NET is housed in the namespace **System.Data** (the assembly name is System.Data.dll)

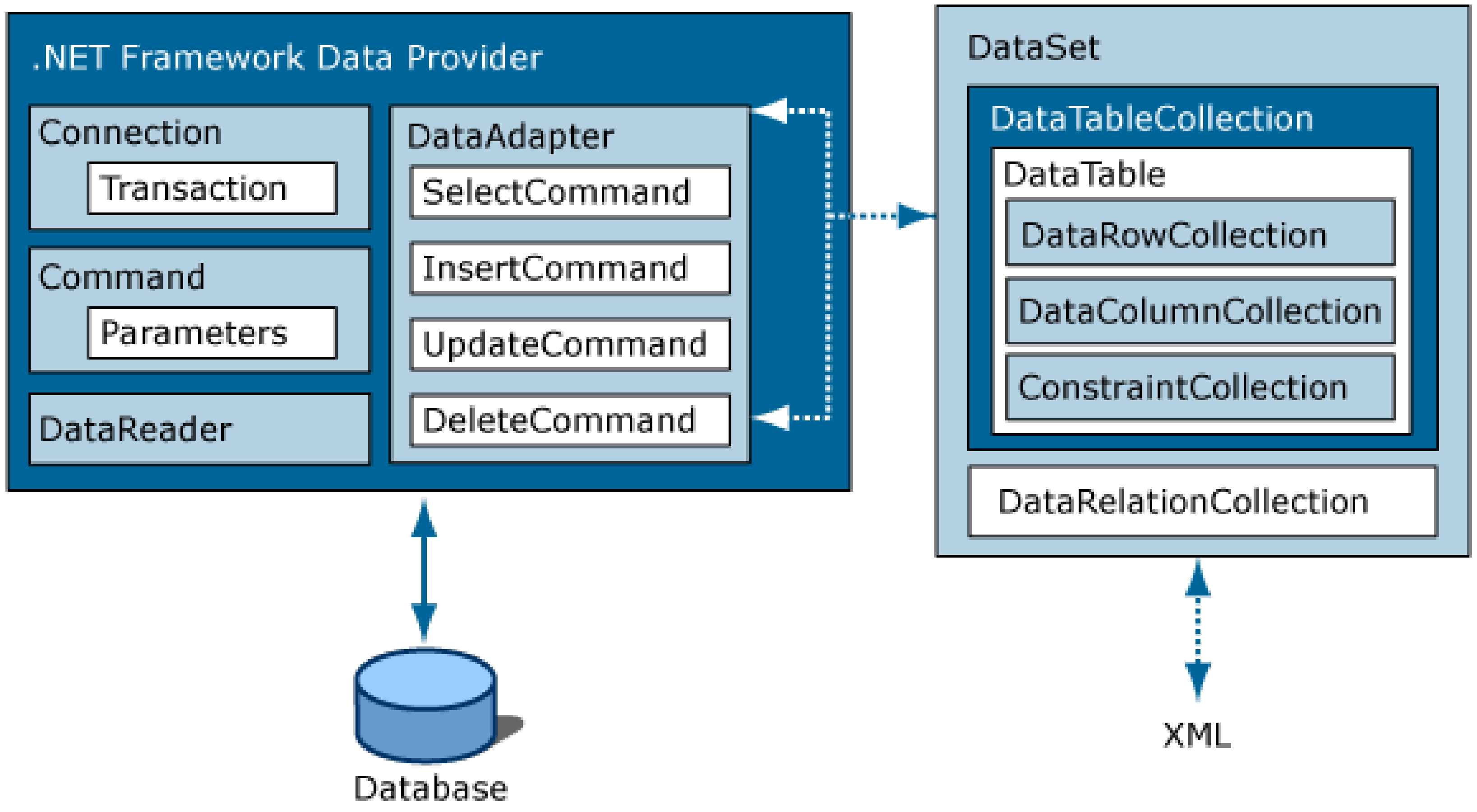
- All the classes and functions for connected and disconnected components live in the **same namespace**.

- Hence, it is important to add a reference of **System.Data** to your application irrespective of the connected or disconnected architecture style you have chosen or will choose later.

ADO.NET Architecture (Cont.)



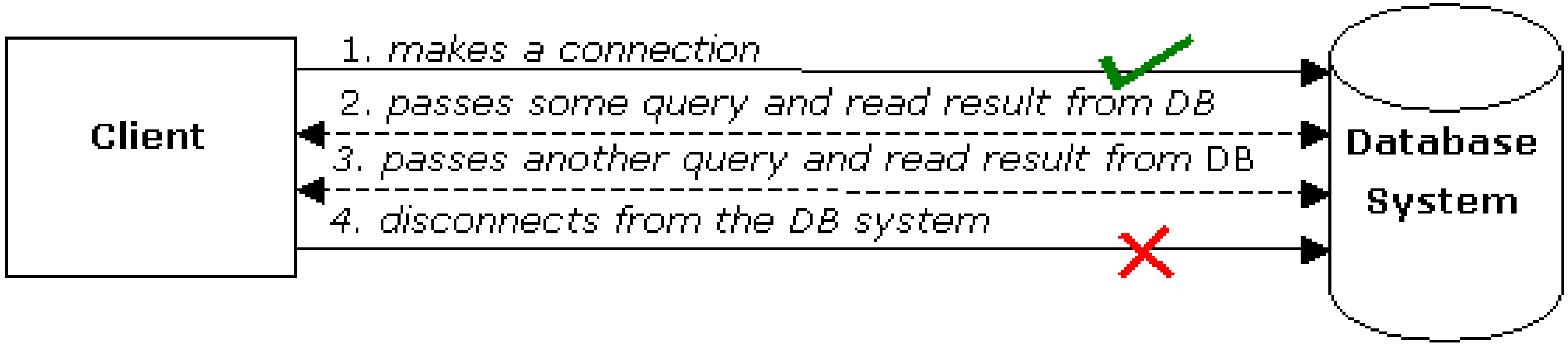
ADO.NET Architecture (Cont.)



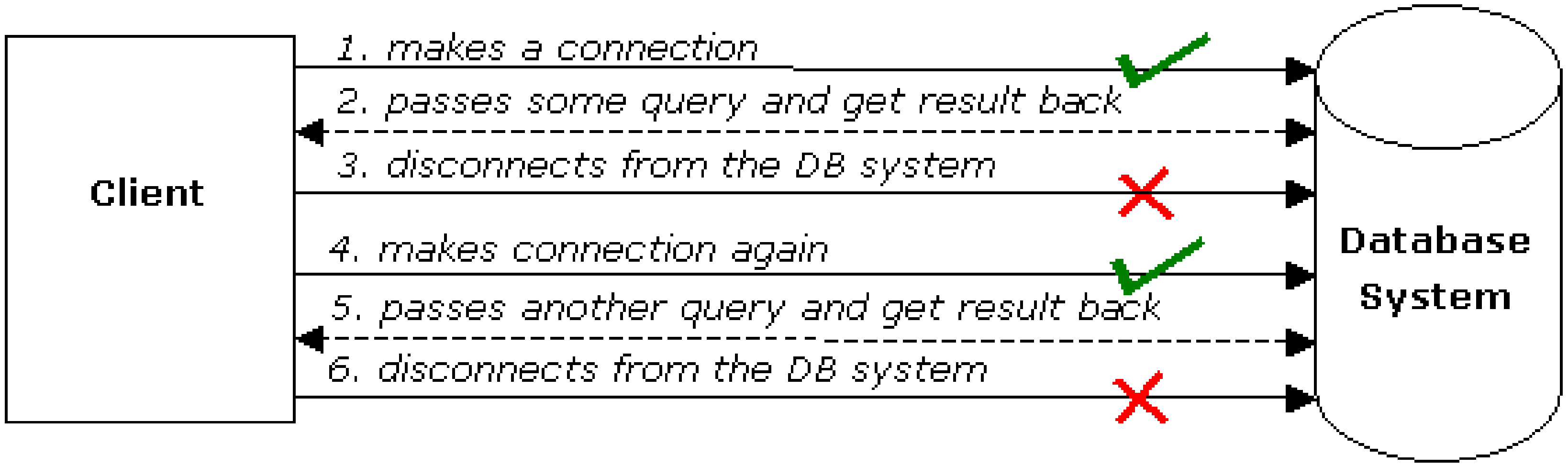
ADO.NET Core Objects

Object	Description
`Connection`	Establishes a connection to a specific data source. (Base class: DbConnection)
`Command`	Executes a command against a data source. Exposes **Parameters** and can execute within the scope of a **Transaction** from a **Connection**. (The base class: DbCommand)
`DataReader`	Reads a forward-only, read-only stream of data from a data source. (Base class: DbDataReader)
`DataAdapter`	Populates a **DataSet** and resolves updates with the data source. (Base class: DbDataAdapter)
`DataTable`	Has a collection of DataRows and DataColumns representing table data, used in disconnected model
`DataSet`	Represents a cache of data. Consists of a set of DataTables and relations among them

Connected Data Access Model


Client
1. makes a connection
2. passes some query and read result from DB
3. passes another query and read result from DB
4. disconnects from the DB system
Database System

Disconnected Data Access Model


Client
1. makes a connection
2. passes some query and get result back
3. disconnects from the DB system
4. makes connection again
5. passes another query and get result back
6. disconnects from the DB system
Database System

Pros and Cons

	Connected	Disconnected
Database Resources	-	+
Network Traffic	-	+
Memory Usage	+	-
Data Access	-	+

Connected Data Objects

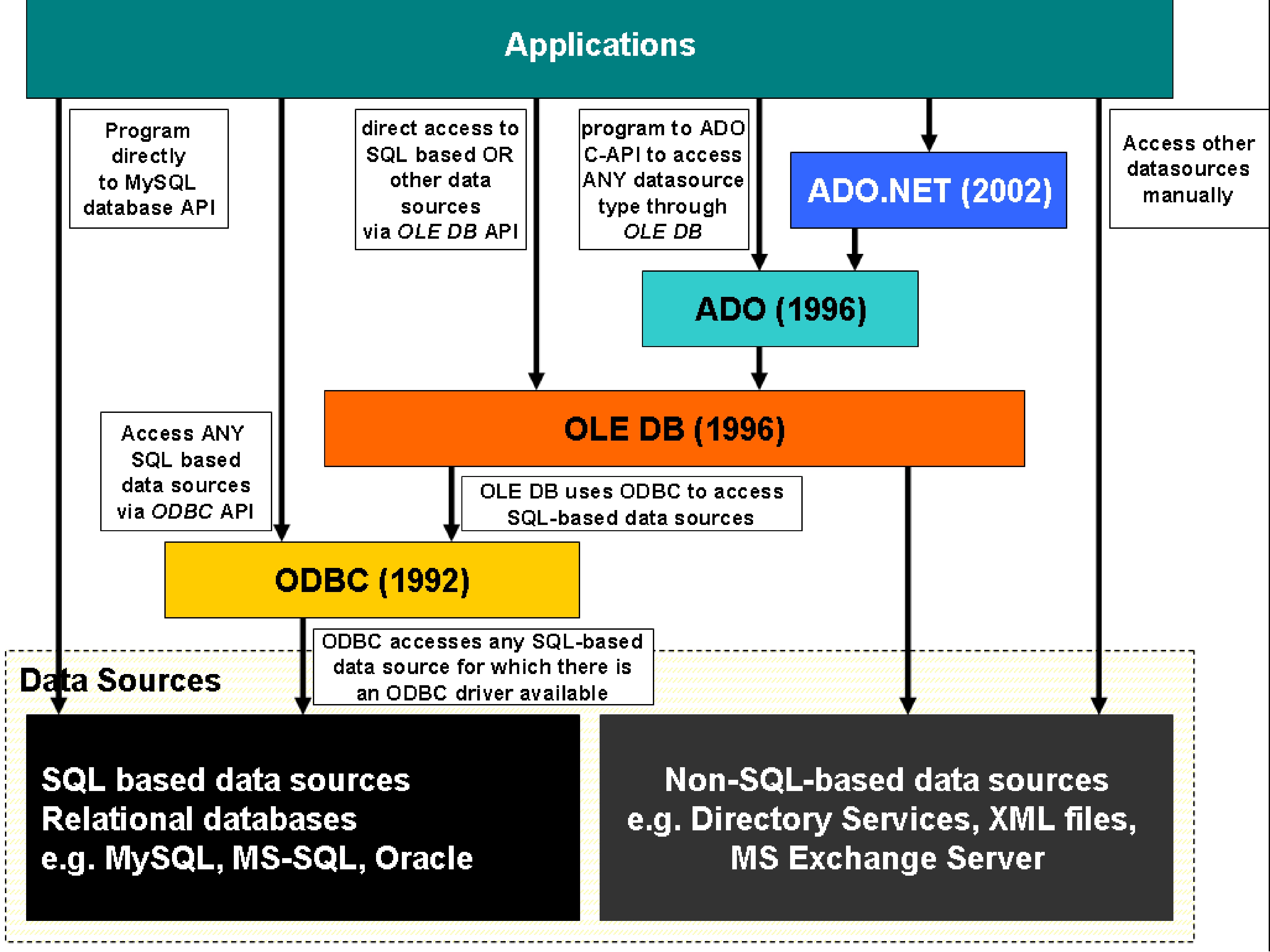
- ADO.NET's connected architecture relies on a consistent database connection to access data and perform any operations on the retrieved data.

- ADO.NET offers the following objects to help you build your application with a connected architecture:
    - Connection
    - Command
    - DataReader

Connection

- Before you can do anything useful with a database, you need to establish a *session* with the database server.
- You do this with an object called a ***connection.*** Here below are data providers that provided by Microsoft:

Data Provider	Namespace	Connection Class
ODBC	System.Data.Odbc	OdbcConnection
OLE DB	System.Data.OleDb	OleDbConnection
Oracle	System.Data.OracleClient	OracleConnection
SQL Server	System.Data.SqlClient	SqlConnection
SQL Server CE	System.Data.SqlServerCe	SqlCeConnection


Applications
Program directly to MySQL database API
direct access to SQL based OR other data sources via *OLE DB* API
program to ADO C-API to access ANY datasource type through *OLE DB*
ADO.NET (2002)
Access other datasources manually
ADO (1996)
OLE DB (1996)
Access ANY SQL based data sources via *ODBC* API
OLE DB uses ODBC to access SQL-based data sources
ODBC (1992)
ODBC accesses any SQL-based data source for which there is an ODBC driver available
Data Sources
SQL based data sources
Relational databases
e.g. MySQL, MS-SQL, Oracle
Non-SQL-based data sources
e.g. Directory Services, XML files, MS Exchange Server

Connection (Cont.)

```
String connString = string.Empty;
// Window Authentication
connString = “server = sqlexpress; integrated security = true”;
// SQL Authentication
connString = “server = sqlexpress; user id = sa; password = 1234567”;
// Code
SqlConnection conn = new SqlConnection(connString);
Conn.Open();
Conn.Close();
```

Command

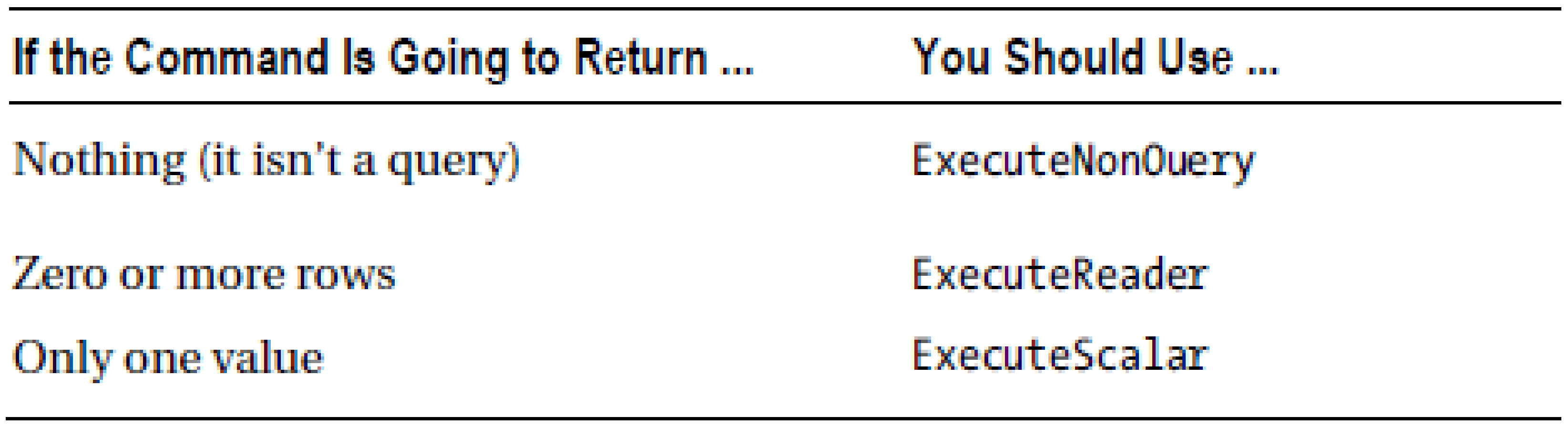
- Once you’ve established a connection to the database, you want to start interacting with it and getting it doing something useful for you.

- You may need to retrieve, add, update, or delete some data, or perhaps modify the database in some other way, usually by running a query.

- **Whatever the task, it will inevitably involve a *command.***

Command (Cont.)

- Commands aren’t much use unless you can execute them.
- Commands have several different methods for executing SQL. The differences between these methods depend on the results you expect from the SQL.
- Queries return rows of data *(result sets*), but the INSERT, UPDATE, and DELETE statements don’t.



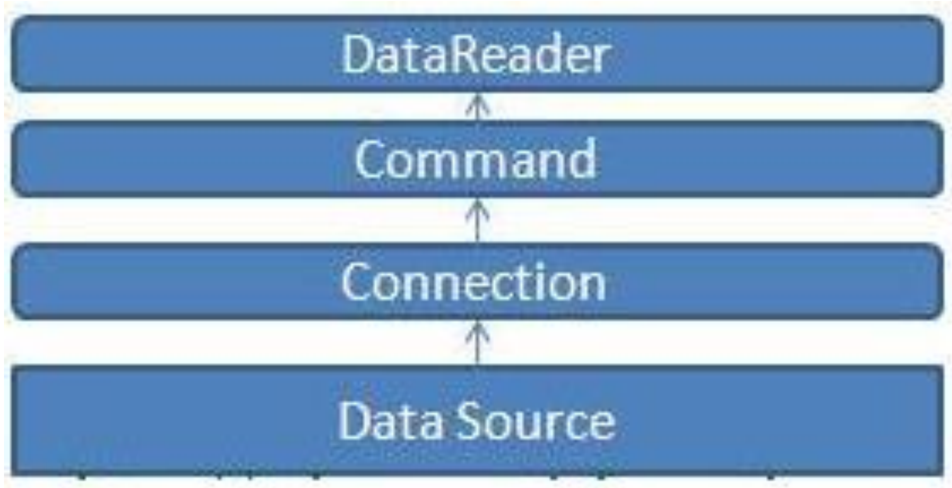
If the Command Is Going to Return ...	You Should Use ...
Nothing (it isn't a query)	`ExecuteNonQuery`
Zero or more rows	`ExecuteReader`
Only one value	`ExecuteScalar`

DataReader

- A data reader is a fast, unbuffered, forward-only, read-only *connected* stream that retrieves data on a per-row basis.

- It reads one row at a time as it loops through a result set.

- You can’t directly instantiate a data reader; instead, you create one with the **ExecuteReader** method of a command.



Steps of Data Acces : Connected Environment

- ▶Create connection
- ▶Create command (select-insert-update-delete)
- ▶Open connection
- ▶If SELECT -> use a **`DataReader`** to fetch data
- ▶If UPDATE,DELETE, INSERT -> use command object’s methods
- ▶Close connection

DataReader (Cont.)

```
// Connection string
String connString = @"server=.\sql2012;database=AdventureWorks;
                              Integrated Security=SSPI";
// Query
string sql = @"select Name from Production. Product";
// Create connection
SqlConnection conn = new SqlConnection(connString);
// Open connection
conn.Open();
```

DataReader (Cont.)

```
SqlCommand cmd = new SqlCommand(sql, conn); // Create command
SqlDataReader rdr = cmd.ExecuteReader(); // Create data reader


// Loop through result set
while (rdr.Read())
{
    // Add to listbox - one row at a time
    Console.WriteLine(rdr[0]);
}
// Close data reader
rdr.Close();
conn.Close();
```

Connected – Update, Delete, Insert

- Command class core methods:

  - **ExecuteNonQuery** : Executes a SQL statement against a connection object

  - **ExecuteReader**: Executes the CommandText against the Connection and returns a DbDataReader

  - **ExecuteScalar**: Executes the query and returns the first column of the first row in the result set returned by the query

Connected – Update, Delete, Insert

```
string connString =
       Properties.Settings.Default.connStr;
SqlConnection conn = new
        SqlConnection(connString);
SqlCommand cmd = new SqlCommand("delete from
    Customers" + "where custID=12344", conn);
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();
```

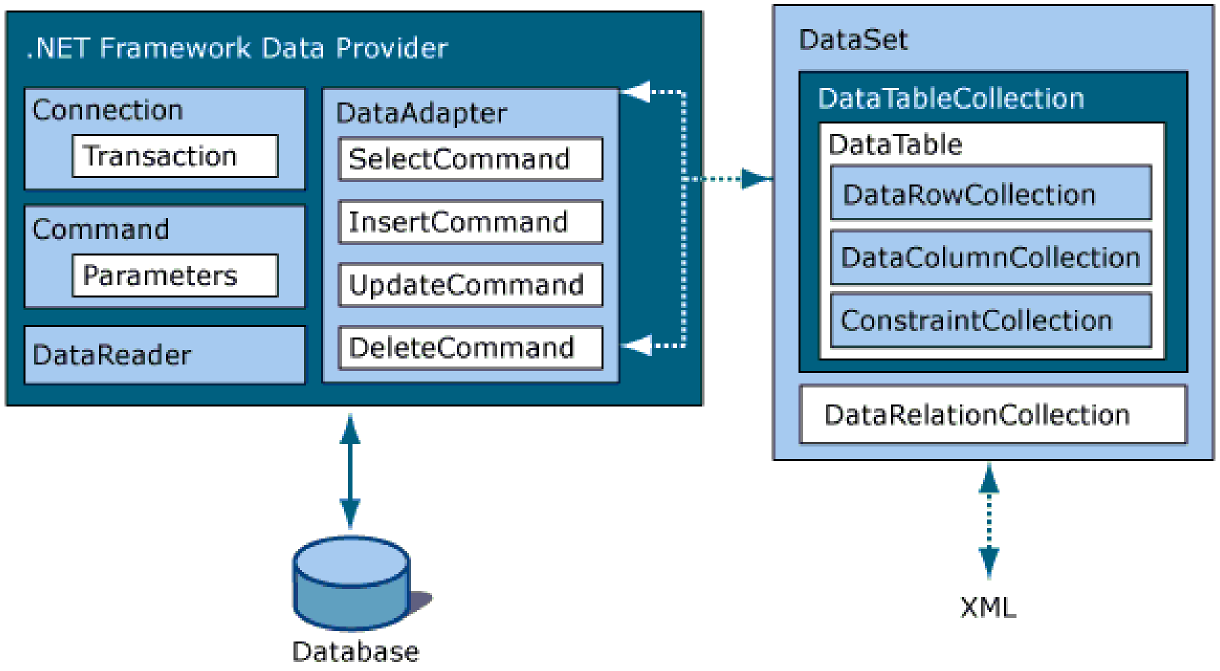
Can be an update or insert command

Disconnected Data Objects

- ADO.NET’s disconnected architecture offers flexible application design and helps organizations save database connections.
- Data can be retrieved and then stored locally on the device in the form of a DataSet object.
- The retrieved DataSet can be modified by users on their local devices such as laptops, handhelds, tablets, and so on, and once that’s done, they can sync the changes into the central data source.
- Disconnected architecture utilizes expansive resources like Connection in a very optimum way (that is, to open late and close early).

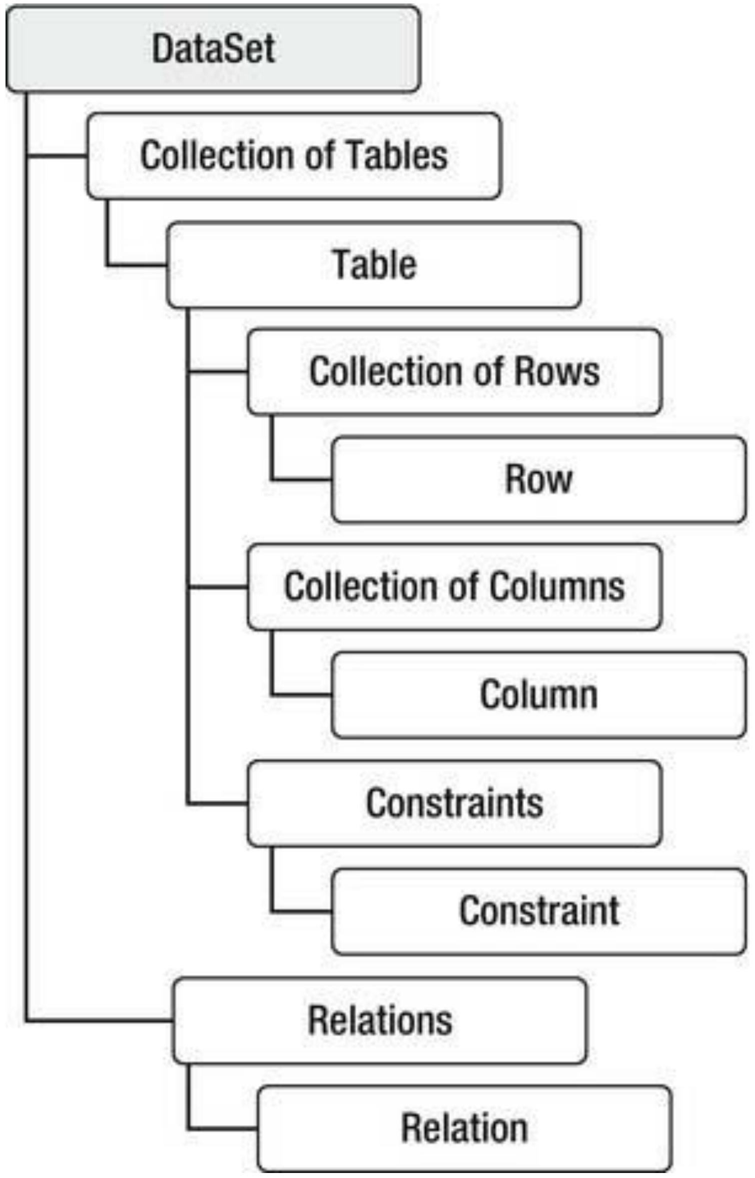
Disconnected Data Objects (Cont.)

- ADO.NET offers the following objects to help you build your application with a disconnected architecture:
  - DataSet
  - DataAdapter
  - DataTable
  - DataColumn
  - DataRow



DataSet

- The fundamental of DataSet purpose is to provide a relational view of data stored in an in-memory cache.



DataSet vs DataReader

- If we simply want to read and display data, then we need to use only a data reader, particularly if we're working with large quantities of data.

- In situations where we need to loop through thousands or millions of rows, we want a fast sequential reader (reading rows from the result set one at a time), and the data reader does this job in an efficient way.

DataSet vs DataReader (Cont.)

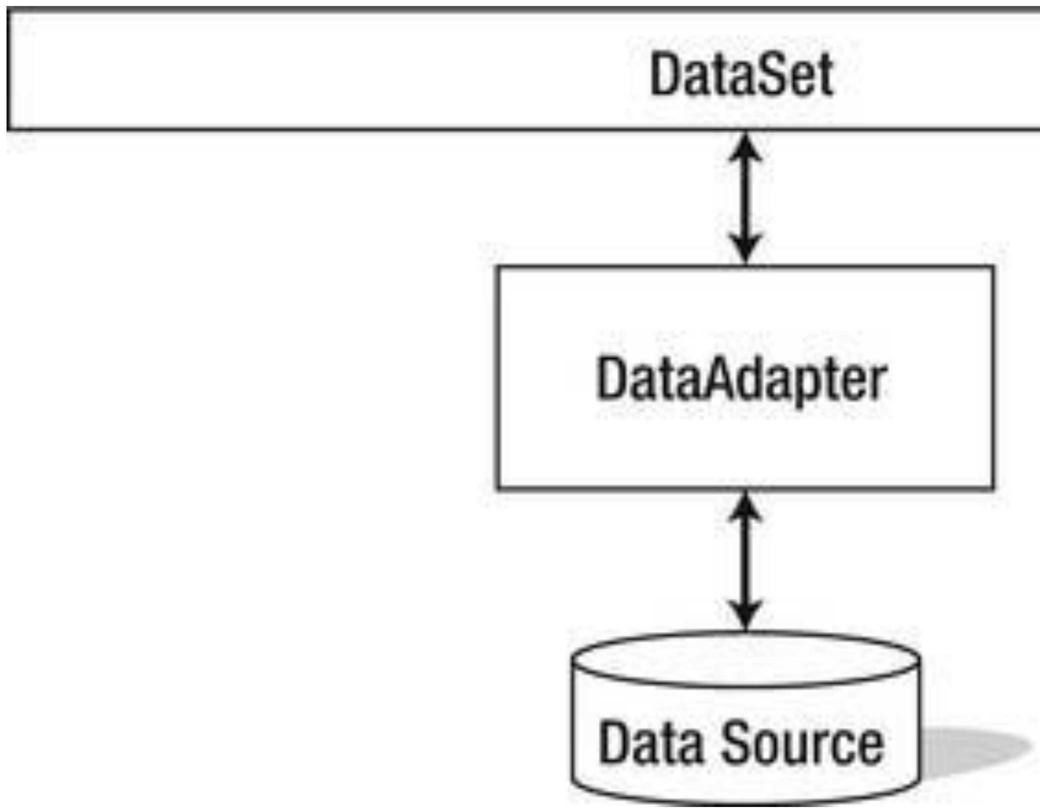
- If we need to manipulate the data in any way and then update the database, we need to use a data set.

- We need to think about whether we really need a data set; otherwise, we’ll just be wasting resources. Unless the need is to update the data source or use other data set features such as reading and writing to XML files, exporting database schemas, and creating XML views of a database, we should use a data reader.

DataAdapter

- When you first instantiate a data set, it contains no data. You obtain a populated data set by passing it to a **data adapter**, which takes care of connection details and is a component of a data provider.
- A data set isn't part of a data provider. It's like a bucket, ready to be filled with water, but it needs an external pipe to let the water in.
- In other words, the data set needs a data adapter to populate it with data and to support access to the data source. Each data provider has its own data adapter in the same way that it has its own connection, command, and data reader.

DataAdapter (Cont.)



DataAdapter (Cont.)

The data adapter constructor is overloaded. You can use any of the following to get a new data adapter. We're using the SQL Server data provider, but the constructors for the other data providers are analogous.

```
SqlDataAdapter da = new SqlDataAdapter();
SqlDataAdapter da = new SqlDataAdapter(cmd);
SqlDataAdapter da = new SqlDataAdapter(sql, conn);
SqlDataAdapter da = new SqlDataAdapter(sql,  connString);
```

DataAdapter (Cont.)

So, you can create a data adapter in four ways:

- You can use its parameterless constructor (assigning SQL and the connection later).
- You can pass its constructor a command (here, cmd is a SqlCommand object).
- You can pass a SQL string and a connection.
- You can pass a SQL string and a connection string.

DataSet with DataAdapter

```
// Connection string
string connString = @"server=sqlexpress;database=
                        AdventureWorks; Integrated Security=true";
// Query
string sql = @"select Name,ProductNumberfromProduction.Product
                where SafetyStockLevel > 600";
// Create connection
SqlConnection conn = new SqlConnection(connString);
// Open connection
conn.Open();
```

DataSet with DataAdapter (Cont.)

```
// Create Data Adapter
SqlDataAdapter da = new SqlDataAdapter(sql, conn);
// Create Dataset
DataSet ds = new DataSet();
// Fill Dataset
da.Fill(ds, "Production.Product");
// Display data
gvProduct.DataSource = ds.Tables["Production.Product"];
//Connection close
conn.Close();
```

DataTable

- A data table is an instance of the class System.Data.DataTable.

- It’s conceptually analogous to a relational table. You can access these nested collections via the Rows and Columns properties of the data table.

- A data table can represent a stand-alone independent table, either inside a data set, or as an object created by another method.

DataTable (Cont.)

```
DataTable table = new DataTable("ParentTable");
DataColumn column;
DataRow row;

column = new DataColumn();
column.DataType = System.Type.GetType("System.Int32");
column.ColumnName = "id";
column.ReadOnly = true;
column.Unique = true;
table.Columns.Add(column);
```

DataTable (Cont.)

```
column = new DataColumn();
column.DataType = System.Type.GetType("System.String");
column.ColumnName = "ParentItem";
column.AutoIncrement = false;
column.ReadOnly = false;
column.Unique = false;
 table.Columns.Add(column);

DataColumn[] PrimaryKeyColumns = new DataColumn[1];
PrimaryKeyColumns[0] = table.Columns["id"];
table.PrimaryKey = PrimaryKeyColumns;
```

DataTable (Cont.)

```
dataSet = new DataSet();
dataSet.Tables.Add(table);


for (int i = 0; i<= 2; i++)
{
   row = table.NewRow();
   row["id"] = i;
   row["ParentItem"] = "ParentItem " + i;
   table.Rows.Add(row);
}
```

DataColumn

- A data column represents the schema of a column within a data table and can then be used to set or get column properties.

- For example, you could use it to set the default value of a column by assigning a value to the DefaultValue property of the data column.

- You obtain the collection of data columns using the data table's Columns property, whose indexer accepts either a column name or a zero-based index, for example (where dt is a data table):

```
DataColumn col = dt.Columns["ContactName"];
DataColumn col = dt.Columns[2];
```

DataRow

- A data row represents the data in a row.
- You can programmatically add, update, or delete rows in a data table.
- To access rows in a data table, you use its Rows property, whose indexer accepts a zero-based index, for example (where dt is a data table):

```
DataRow row = dt.Rows[2];
```

Understanding .NET Data Providers

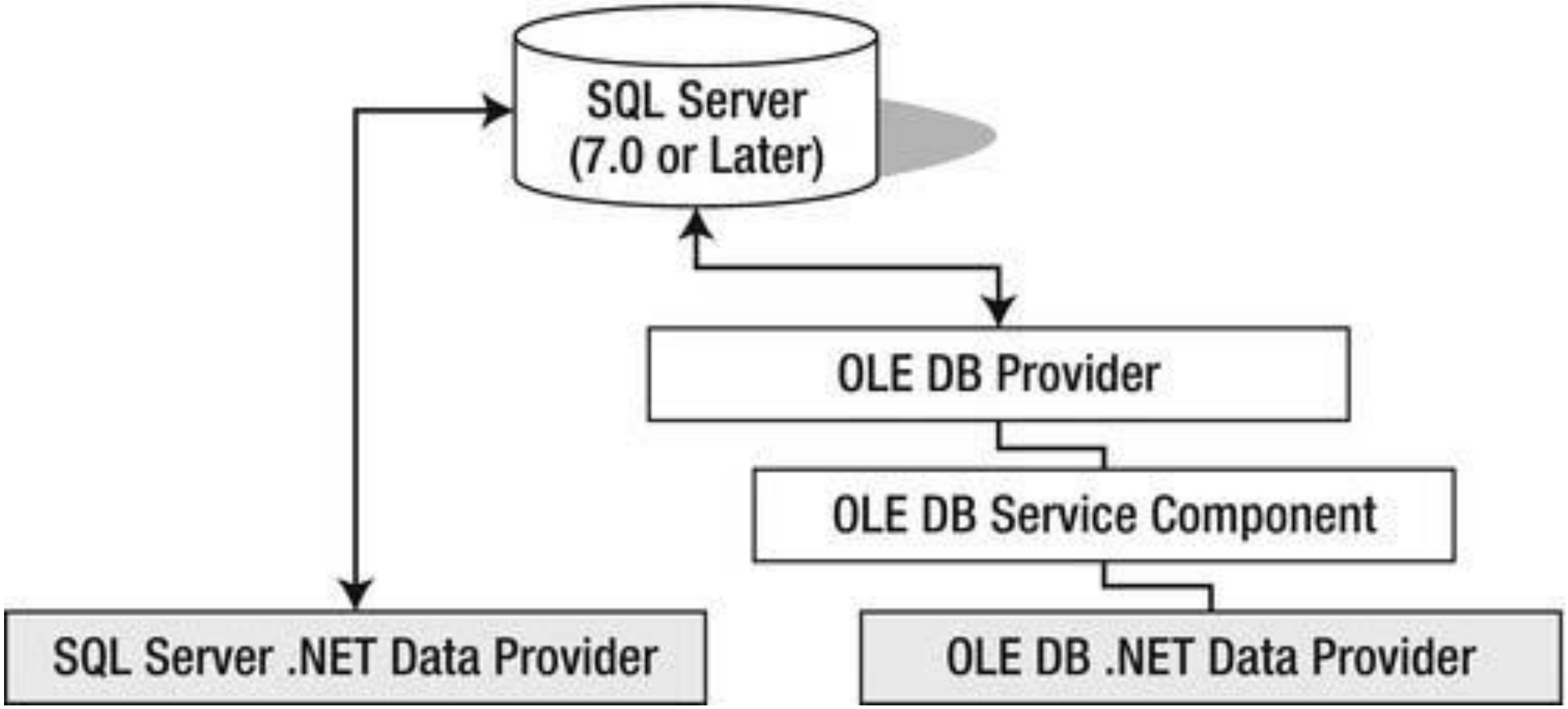
- **ADO.NET** consists of various data providers that allow an easy and predefined object model to communicate with various industry databases such as SQL Server, Oracle, Microsoft Access, and many others.

- For example, if you use SQL Server, you should use the SQL Server data provider (**System.Data.SqlClient**) because it’s the most efficient way to access **SQL Server**.

- The **OLE DB** data provider supports access to **older versions of SQL Server** as well as to other databases, such as **Access**, **DB2**, **MySQL**, and **Oracle**.

Understanding .NET Data Providers

- **Native** data providers (such as **System.Data.OracleClient**) are preferable for performance, since the OLE DB data provider works through two other layers, the OLE DB service component and the OLE DB provider, before reaching the data source.



Understanding .NET Data Providers

- Each .NET data provider is designed to do the following two things very well:
  - Provide access to data with an active connection to the data source
  - Provide data transmission to and from disconnected data sets and data tables
- Database connections are established by using the data provider's Connection class (for example, **System.Data.SqlClient.SqlConnection**).
- Other components such as data readers, commands, and data adapters support retrieving data, executing SQL statements, and reading or writing to data sets or data tables, respectively.

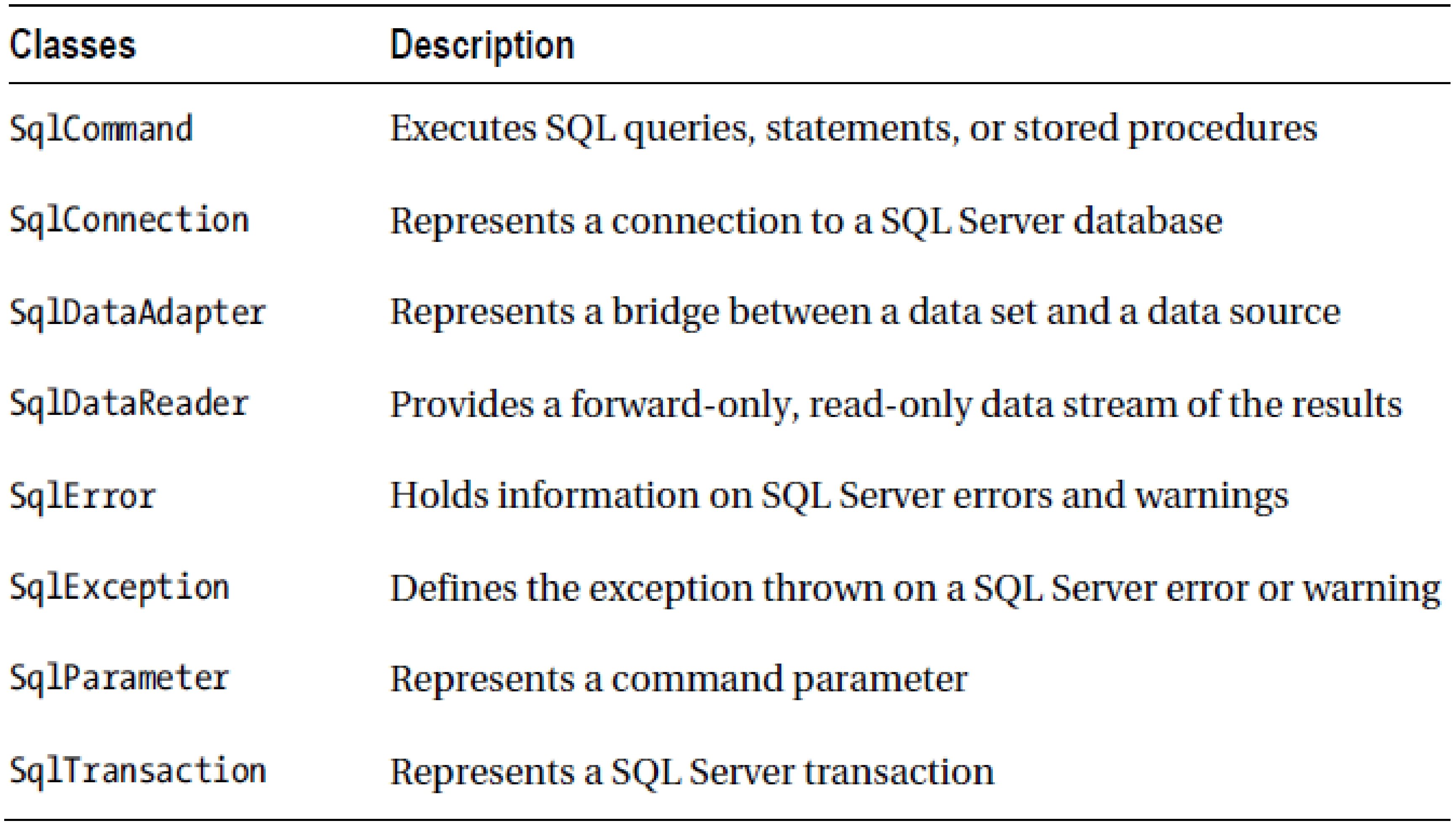
Understanding the SQL Server Data Provider

- The .NET data provider for SQL Server is in the **System.Data.SqlClient** namespace.

- Although you can use **System.Data.OleDb** to connect with SQL Server, Microsoft has specifically designed the **System.Data.SqlClient** namespace to be used with **SQL Server**, and it works in a more efficient and optimized way than System.Data.OleDb.

- The reason for this efficiency and optimized approach is that this data provider communicates directly with the server using its native network protocol instead of through multiple layers.

Commonly Used SqlClient Classes



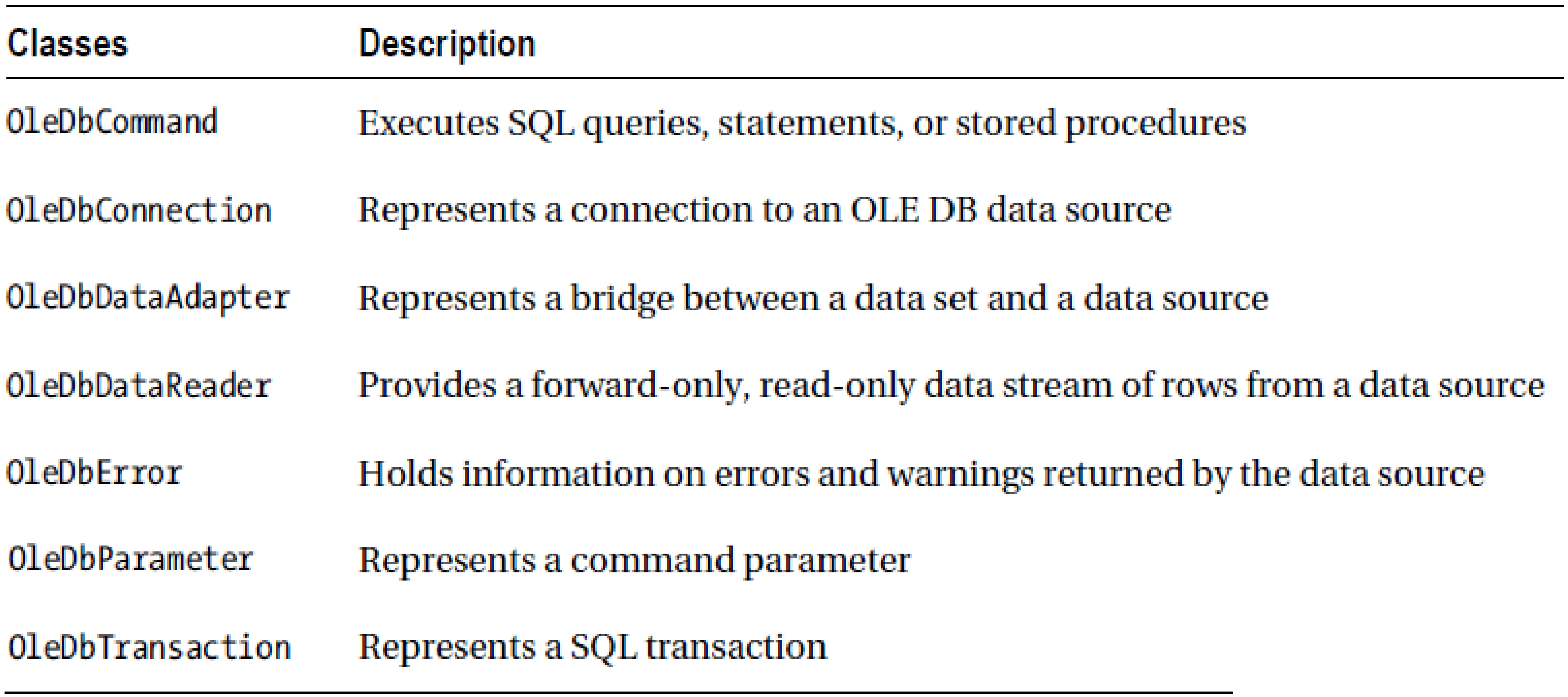
Classes	Description
SqlCommand	Executes SQL queries, statements, or stored procedures
SqlConnection	Represents a connection to a SQL Server database
SqlDataAdapter	Represents a bridge between a data set and a data source
SqlDataReader	Provides a forward-only, read-only data stream of the results
SqlError	Holds information on SQL Server errors and warnings
SqlException	Defines the exception thrown on a SQL Server error or warning
SqlParameter	Represents a command parameter
SqlTransaction	Represents a SQL Server transaction

Understanding the OLE DB Data Provider

- Outside .NET, **OLE DB** is still Microsoft’s high-performance data access technology.

- You can use this data provider to access data stored in any format, so even in ADO.NET it plays an important role in accessing data sources that don’t have their own ADO.NET data providers.

Commonly Used OleDb Classes



Classes	Description
`OleDbCommand`	Executes SQL queries, statements, or stored procedures
`OleDbConnection`	Represents a connection to an OLE DB data source
`OleDbDataAdapter`	Represents a bridge between a data set and a data source
`OleDbDataReader`	Provides a forward-only, read-only data stream of rows from a data source
`OleDbError`	Holds information on errors and warnings returned by the data source
`OleDbParameter`	Represents a command parameter
`OleDbTransaction`	Represents a SQL transaction

Commonly Used Odbc Classes

Classes	Description
OdbcCommand	Executes SQL queries, statements, or stored procedures
OdbcConnection	Represents a connection to an ODBC data source
OdbcDataAdapter	Represents a bridge between a data set and a data source
OdbcDataReader	Provides a forward-only, read-only data stream of rows from a data source
OdbcError	Holds information on errors and warnings returned by the data source
OdbcParameter	Represents a command parameter
OdbcTransaction	Represents a SQL transaction

Understanding the ODBC Data Provider

- **ODBC** was Microsoft's original general-purpose data access technology.
- The ODBC architecture is essentially a three-tier process.
- An application uses ODBC functions to submit database requests.
- ODBC converts the function calls to the protocol (call-level interface) of a driver specific to a given data source.

Understanding the ODBC Data Provider

- The driver communicates with the data source, passing any results or errors back up to ODBC.

- Obviously, this is less efficient than a database-specific data provider’s direct communication with a database, so for performance it’s preferable to avoid the ODBC data provider, since it merely offers a simpler interface to ODBC but still involves all the ODBC overhead.

Summary

- There is two main types of data access that provided from ADO.NET: Connected Data Objects and Disconnected Data Objects.
- Both types have their own advantages to fulfill the full-functionality to access data.
- Both types their own main components:
  - Connected Data Objects : Connection, Command, and DataReader.
  - Disconnected Data Objects : DataSet, DataAdapter, DataTable, DataColumn and DataRow.