Visual C# .Net using framework 4.5

Eng. Mahmoud Ouf

Lecture 08

Introduction to 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.

ADO.NET classes are found in System.Data.dll.

ADO.NET helps connect the UI, or presentation layer, of your application with the data source or database.

ADO.NET is a completely new data access technology, with a new design that was built entirely from scratch.

Dealing with Database

To Deal with the Database, there are two approaches:

Connected Model:

In which we maintain an open connection with the database while performing Database transactions

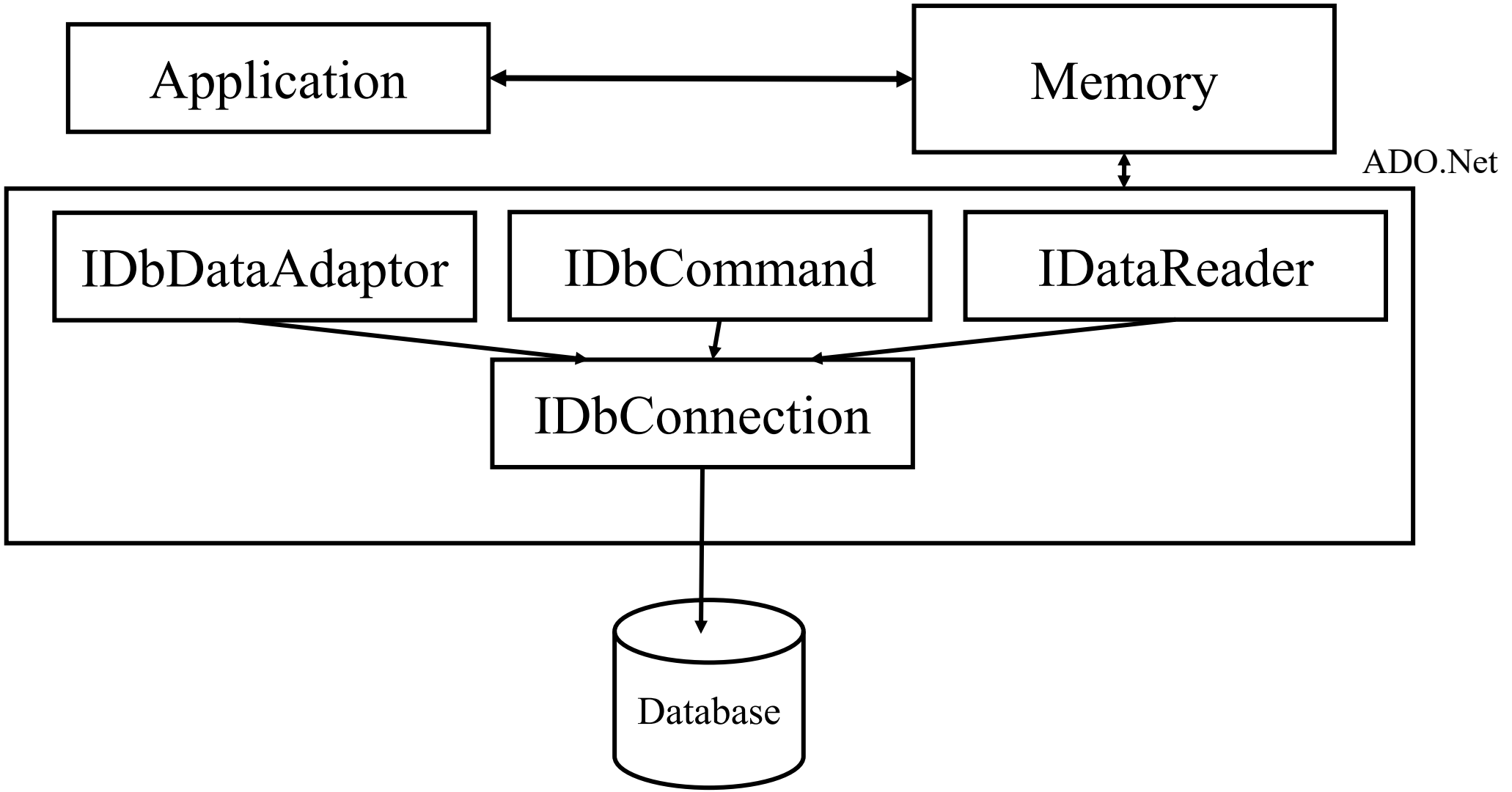
Disconnected Model:

In which we open connection with the database, retrieve the database (Tables, Relations, Columns, Rows, …) to the Memory then close the connection.

Perform the database transaction on the database copy in the memory

Then, open the database connection and return the updates to the real Database

ADO .Net Data Architecture



ADO .Net Namespaces

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.OleDb	The .NET Framework data provider for OLE DB
System.Data.SqlClient	The .NET Framework data provider for SQL Server

Working with Disconnected Model (DataSet)

In today's complex distributed application environments, it is not possible to rely on a dedicated database connection to retrieve and modify data.

To solve this issue, use ADO.NET's disconnected architecture; it offers flexible application design and helps organizations save database connections.

Hence, 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, then 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).

Working with Disconnected Model (DataSet)

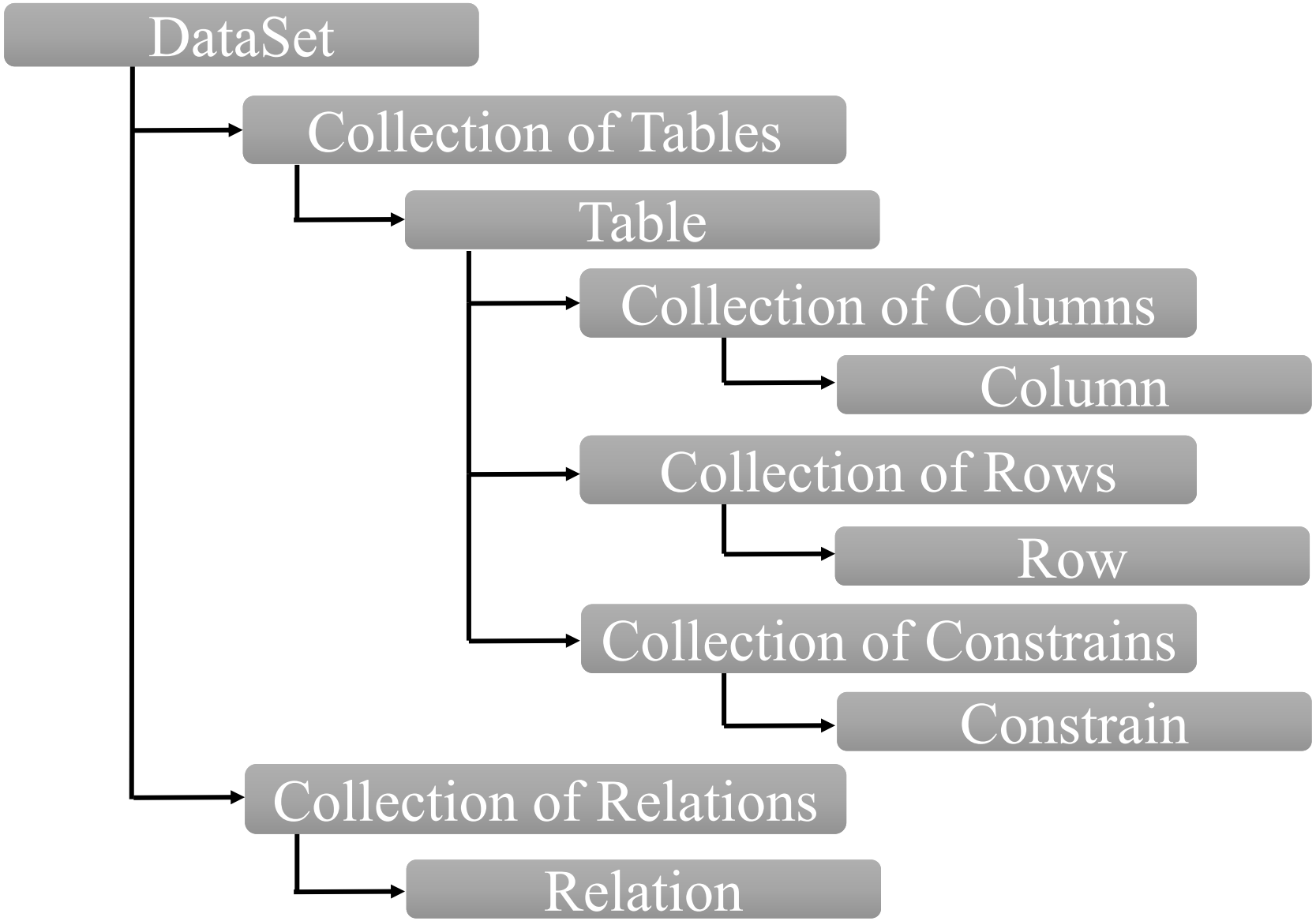
It is a disconnected, in-memory representation of data. When appropriate, The Data Set can act as a template for updating the central database.

The DataSet object contains a collection of zero or more DataTable objects. Each is an in memory representation of a single table.

The DataTable is defined by DataColumns collection and the Constrains Collections. These two makes up the schema of the table. A DataTable contains a DataRows Collections.

The DataSet contains a DataRelations collections (allows the creation of association between one table and rows in another tables.

Working with Disconnected Model (DataSet)



Working with Disconnected Model (DataSet)

Create a DataSet

DataSet ds = new DataSet("MyDataSet");

Create 2 tables (Employee) and (Department)

DataTable Emp = new DataTable("Employee");

DataTable Dept = new DataTable (Department");

Create 5 column

DataColumn EmpId = new DataColumn("ID", Type.GetType("System.Int32"));

DataColumn EmpName = new DataColumn("Name", Type.GetType("System.string));

DataColumn EmpDeptId = new DataColumn("DeptID", Type.GetType("System.Int32"));

DataColumn DeptId = new DataColumn("ID", Type.GetType("System.Int32"));

DataColumn DeptName = new DataColumn("Name", Type.GetType("System.string));

Working with Disconnected Model (DataSet)

Add 3 to table “Employee” and 2 to table “Department”

```
Dept.Columns.Add(DeptId);
Dept.Columns.Add(DeptName);
Emp.Columns.AddRange(New DataColumns[]{EmpId,  EmpName,
EmpDeptId});
```

Add the 2 tables to the dataset

```
ds.Tables.Add(Emp);
ds.Tables.Add(Dept);
```

Create a Relation

```
DataRelation dr = new DataRelation(“EmpDept”, DeptId,  EmpDeptId);
```

Add the relation to the DataSet

```
ds.DataRelations.Add(dr);
```

Working with Disconnected Model (DataSet)

Create Rows

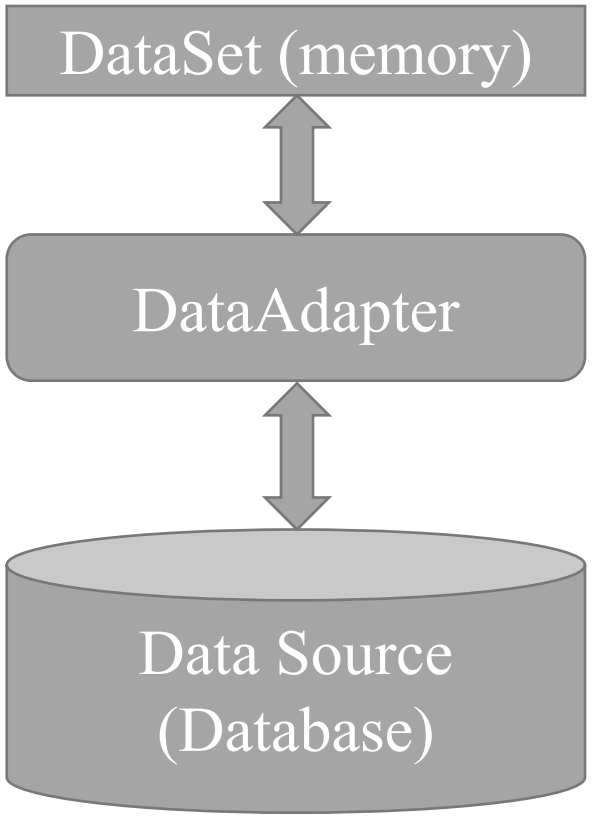
```
DataRow dRow = Emp.NewRow();
dRow[0] = 1;
dRow[“Name”] = “Aly”;
```

Add rows to the Table

```
Emp.Rows.Add(dRow);
```

Working with Disconnected Model (DataSet)

We can create a DataSet in the memory that represent an image of a real database.
This will help to have an offline image of database in the memory in a way that make it easy to deal with then update the real database.
This is done through the use of DataAdaptor



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.

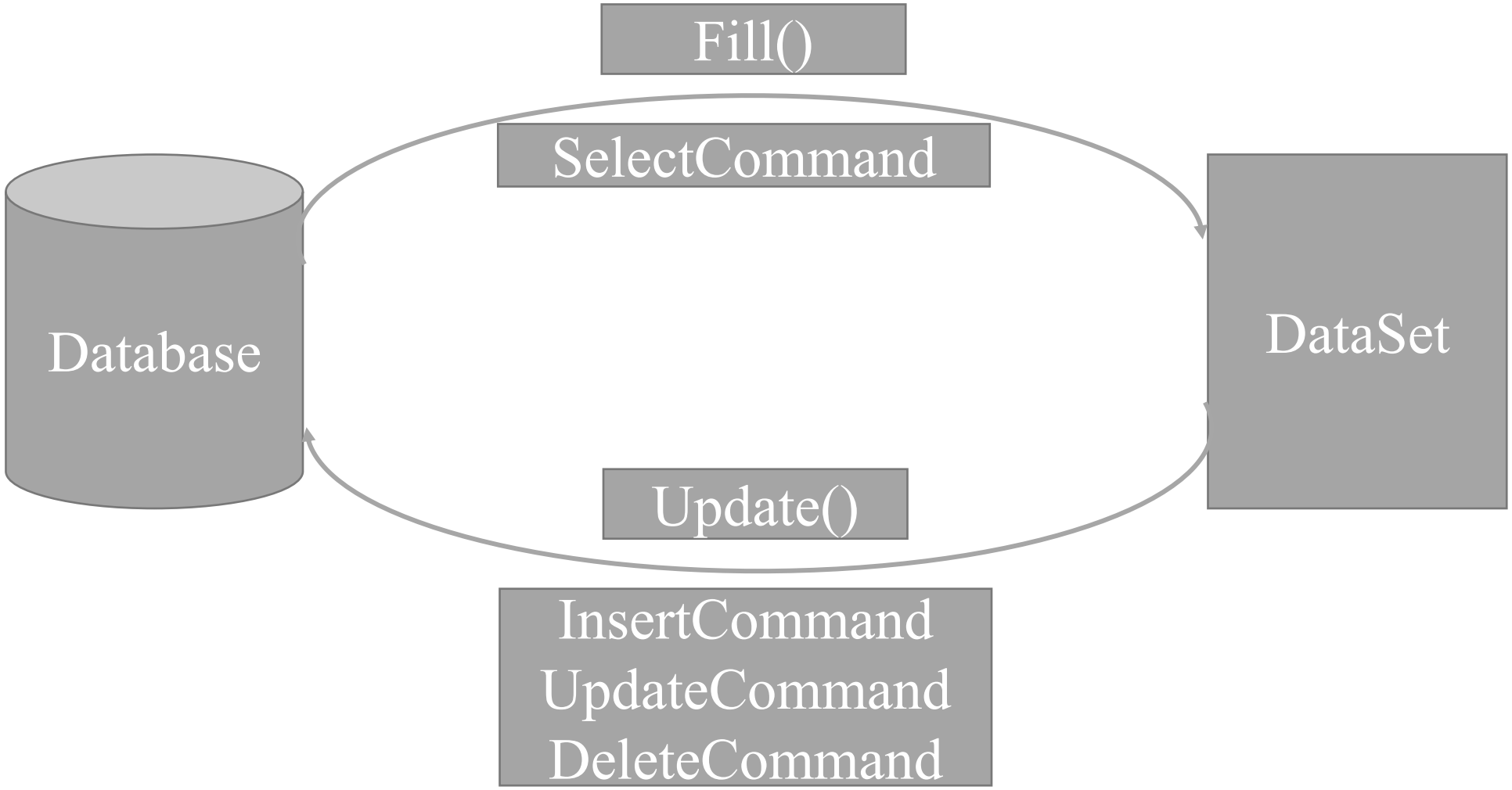
The data set needs a data adapter to populate it with data and to support access to the data source.

DataAdapter

DataAdapter has 4 important properties represent the database command:
SelectCommand
InsertCommand
UpdateCommand
Delete Command

Also, it has 2 important methods
Fill(): which is used to transfer the data from database to dataset
Update(): which is used to transfer the data from dataset to database

DataAdapter



Working with DataAdapter

1. Create Object From DataAdapter (dAdapt)
2. Create Object From DataSet (ds)
3. Open the connection with the database
4. Call the Fill() method (dAdapt.Fill(ds)), this will use the SelectCommad
5. Close the connection with the database
6. Process the dataset
7. Open the connection with the database
8. Call the Update() method (dAdapt.Update()), this will use the InsertCommad, UpdateCommand, DeleteCommand
9. Close the connection with the database

Working with Connection

1. Before dealing with database, you have to establish a session with the database server
2. This is done by making an object from connection, which is an instance of a class that implements the System.Data.IDbConnection interface for a specific data provider.
3. The name of the Connection class differ from data provider to another.
4. We have 2 data providers:
   Sql server (using System.Data.SqlClient)
   OleDb (using System.Data.OleDb)

The Connection class for SQL Server (SqlConnection)

The Connection class for MS-Access (OleDbConnection)