

Question 1: Introduction to ADO.NET

Answer

Definition: ADO is a rich set of classes, interfaces, structures, and enumerated types that manage data access from various types of data stores.

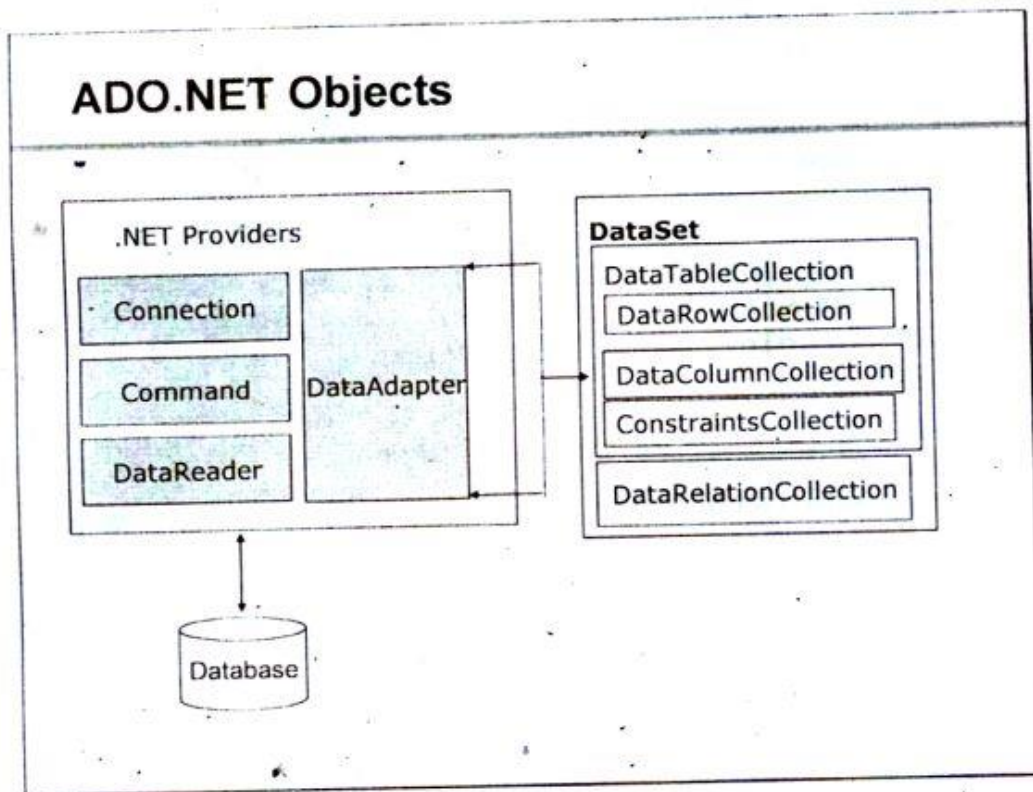
- Enterprise applications handle a large amount of data. This data is primarily stored in relational databases, like Oracle, SQL Server, Access, and so on. These databases use Structured Query Language (SQL) for retrieval of data.
- To access enterprise data from a .NET application, an interface was needed. This interface acts as a bridge between an RDBMS system and a .Net application. ADO.NET is such an interface that is created to connect .NET applications to RDBMS systems.
- In the .NET framework, Microsoft introduced a new version of Active X Data Objects (ADO) called ADO.NET. Any .NET application, either Windows-based or web-based, can interact with the database using a rich set of classes of the ADO.NET library. Data can be accessed from any database using connected or disconnected architecture.
- There were many data access technologies available prior to ADO.NET, primarily the following:
 - Open Database Connectivity (ODBC)
 - Data Access Objects (DAO)
 - Remote Data Objects (RDO)
 - Active X Data Objects (ADO)
- ADO is a simple component-based object-oriented interface to access data whether relational or non-relational databases. It is a successor of DAO and RDO.
- ADO reduces the number of objects. Their properties, methods, and events.
- ADO is built on COM; specifically Activex
- ADO supports universal data access using Object Linking and Embedding for DataBases (OLEDB). This means that there are no restrictions on the type of data that can be accessed.

ADO.NET provides mainly the following two types of architectures:

1. Connected Architecture
2. Disconnected Architecture

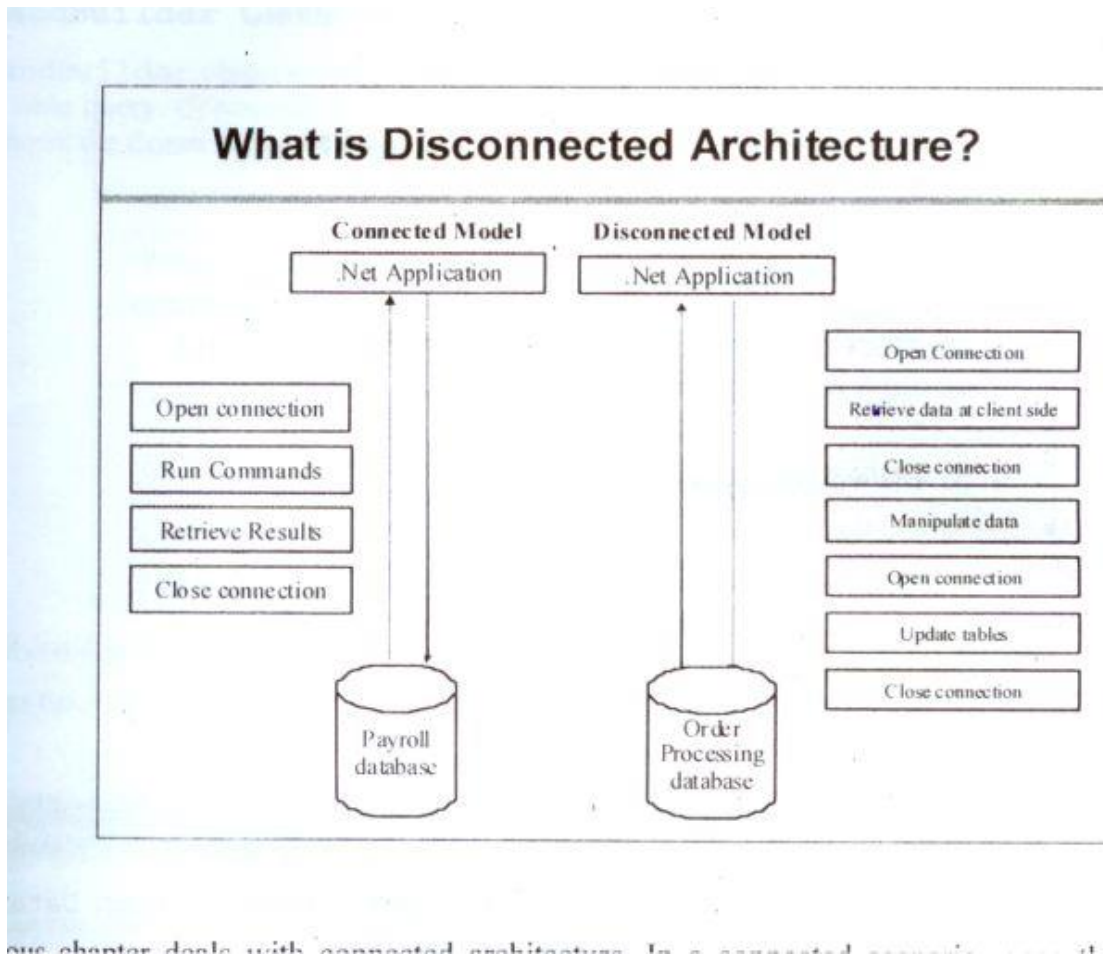
Question 2: ADO.NET Architecture

Answer



Connected and Disconnected Architectures

The following figure shows how to work with the connected and disconnected architectures.



Connected Architecture

1. In the connected architecture, connection with a data source is kept open constantly for data access as well as data manipulation operations.
2. The ADO.NET Connected architecture considers mainly three types of objects.
 - SqlConnection con;
 - SqlCommand cmd;
 - SqlDataReader dr;

Disconnected Architecture

1. Disconnected is the main feature of the .NET framework. ADO.NET contains various classes that support this architecture. The .NET application does not always stay connected with the database. The classes are designed in a way that they automatically open and close the connection. The data is stored client-side and is updated in the database whenever required.
2. The ADO.NET Disconnected architecture considers primarily the following types of objects:
 - DataSet ds;

- SqlDataAdapter da;
- SqlConnection con;
- SqlCommandBuilder bldr;

1. Connection Object and Connection string

Connection Object

1. One of the first ADO.NET objects is the connection object, that allows you to establish a connection to a data source.
2. The connection objects have the methods for opening and closing connections, for beginning a transaction of data.
3. The .Net Framework provides two types of connection classes: The SqlConnection object, that is designed specially to connect to Microsoft SQL Server and the OleDbConnection object, that is designed to provide connection to a wide range of databases, such as Microsoft Access and Oracle.
4. A connection is required to interact with the database. A Connection object helps to identify the database server name, user name and password to connect to the database. The Connection object is used by commands on the database.
5. A Connection object has the responsibility of establishing a connection with the data store.
6. How to use the SqlConnection object:
 - Instantiate the SqlConnection class.
 - Open connection.
 - Pass the connection to ADO.NET objects.
 - Perform the database operations with ADO.NET object.
 - Close the connection.

Connection String

No.	Connection String Parameter Name	Description
1	Data Source	Identify the server. Could be a local machine, machine domain name, or IP Address.
2	Initial Catalog	Data base name.
3	Integrated Security	Set to SSIP to make a connection with the user's window log in.
4	User ID	Name of user configured in SQL Server.
5	Password	Password matching SQL Server User ID

The connection string is different for each of the various data providers available in .NET 2.0. There are different connection strings for the various types of data sources. You can find a list of all the available providers for creating a connection in a table:

No	Provider	Description
----	----------	-------------

1	System.Data.SqlClient	Provides data for Microsoft SQL Server
2	System.Data.OleDb	Provides data access for data sources exposed using OLE DB
3	System.Data.Odbc	Provides data access for data source exposed using ODBC.
4	System.Data.OracleClient	Provides data access for Oracle.

Example

```
1. SqlConnection con;
2. con = new SqlConnection("Server=Krushna;Database=Anagha;U
   id=sa;Pwd=sa");
```

2. Command Object

- A Command object executes SQL statements on the database. These SQL statements can be SELECT, INSERT, UPDATE, or DELETE. It uses a connection object to perform these actions on the database.
- A Connection object specifies the type of interaction to perform with the database, like SELECT, INSERT, UPDATE, or DELETE.
- A Command object is used to perform various types of operations, like SELECT, INSERT, UPDATE, or DELETE on the database.

• SELECT

```
1. cmd =new SqlCommand("select * from Employee", con
);
```

• INSERT

```
1. cmd = new SqlCommand("INSERT INTO Employee(Emp_ID
,
2. Emp_Name)VALUES ('" + aa + "', '" + bb + "')", con
);
```

• UPDATE

```
1. SqlCommand cmd =new SqlCommand("UPDATE Employee S
ET
2. Emp_ID ='" + aa + "', Emp_Name ='" + bb + "'" WHER
E
3. Emp_ID = '" + aa + "'", con);
```

• DELETE

```
1. cmd =new SqlCommand("DELETE FROM Employee where
2. Emp_ID=''" + aa + "'", con);
```

- A Command object exposes several execute methods like:
 - **ExecuteScaler()**
Executes the query, and returns the first column of the first row in the result set returned by the query. Extra columns or rows are ignored.
 - **ExecuteReader()**
Display all columns and all rows in the client-side environment.

In other words, we can say that they display datatables client-side.

- **ExecuteNonQuery()**

Something is done by the database but nothing is returned by the database.

3. Data Reader Object

A DataReader object is used to obtain the results of a SELECT statement from a command object. For performance reasons, the data returned from a data reader is a forward-only stream of data. This means that the data can be accessed from the stream in a sequential manner. This is good for speed, but if data needs to be manipulated then a dataset is a better object to work with.

Example

```
1. dr = cmd.ExecuteReader();  
2. DataTable dt = new DataTable();  
3. dt.Load(dr);
```

- It is used in Connected architecture.
- Provide better performance.
- DataReader Object has Read-only access.
- DataReader Object Supports a single table based on a single SQL query of one database.
- While DataReader Object is Bind to a single control.
- DataReader Object has Faster access to data.
- DataReader Object Must be manually coded.
- we can't create a relation in the data reader.
- whereas Data reader doesn't support.
- The data reader communicates with the command object.
- DataReader can not modify data.

4. Data Adapter Object

- A Data Adapter represents a set of data commands and a database connection to fill the dataset and update a SQL Server database.
- A Data Adapter contains a set of data commands and a database connection to fill the dataset and update a SQL Server database. Data Adapters form the bridge between a data source and a dataset.
- Data Adapters are designed depending on the specific data source. The following table shows the Data Adapter classes with their data source.

Provider-Specific Data Adapter classes	Data Source
SqlDataAdapter	SQL Server
OleDbDataAdapter	OLE DB provider
OdbcDataAdapter	ODBC driver
OracleDataAdapter	Oracle

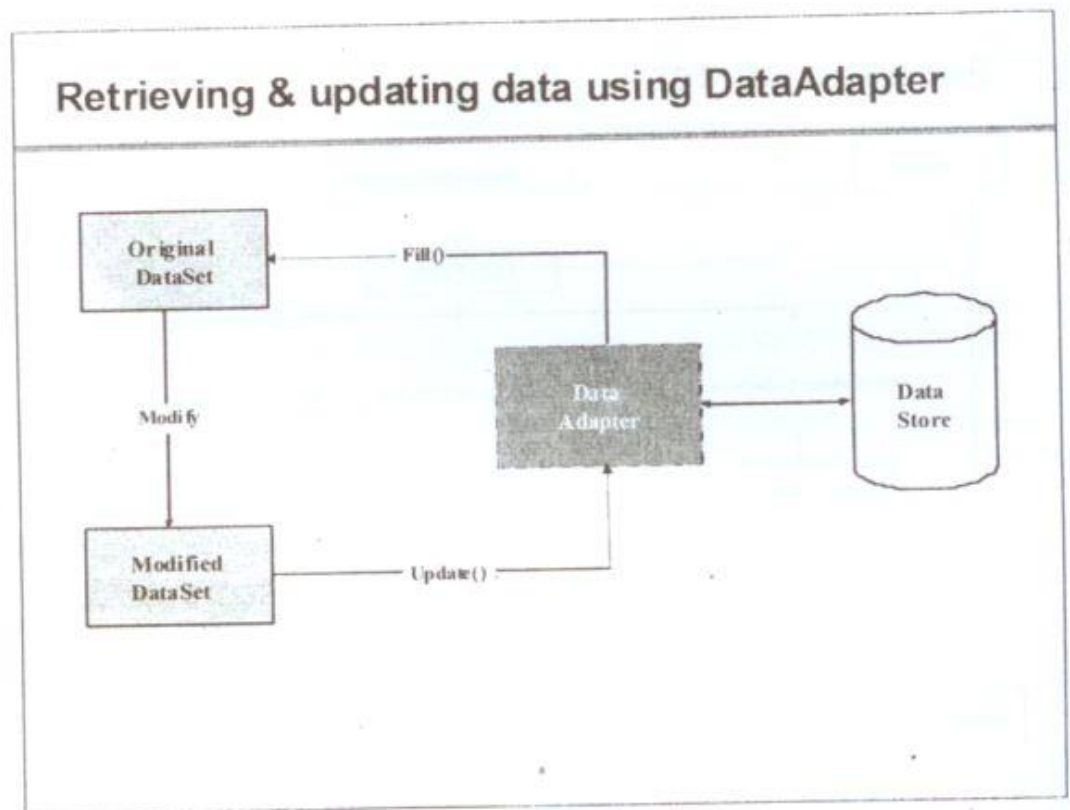
Provider-Specific Data Adapter classes

Data Source

- A Data Adapter object accesses data in a disconnected mode. Its object contains a reference to a connection object.
- It is designed in a way that implicitly opens and closes the connection whenever required.
- It maintains the data in a DataSet object. The user can read the data if required from the dataset and write back the changes in a single batch to the database. Additionally, the Data Adapter contains a command object reference for SELECT, INSERT, UPDATE, and DELETE operations on the data objects and a data source.
- A Data Adapter supports mainly the following two methods:
 - **Fill ()**
The Fill method populates a dataset or a data table object with data from the database. It retrieves rows from the data source using the SELECT statement specified by an associated select command property.
The Fill method leaves the connection in the same state as it encountered it before populating the data. If subsequent calls to the method for refreshing the data are required then the primary key information should be present.
 - **Update ()**
The Update method commits the changes back to the database. It also analyzes the RowState of each record in the DataSet and calls the appropriate INSERT, UPDATE, and DELETE statements.
A Data Adapter object is formed between a disconnected ADO.NET object and a data source.

Example

```
1. SqlDataAdapter da=new SqlDataAdapter("Select * fr  
   om Employee", con);  
2. da.Fill(ds,"Emp");  
3. bldr =new SqlCommandBuilder(da);  
4. dataGridView1.DataSource = ds.Tables["Emp"];
```

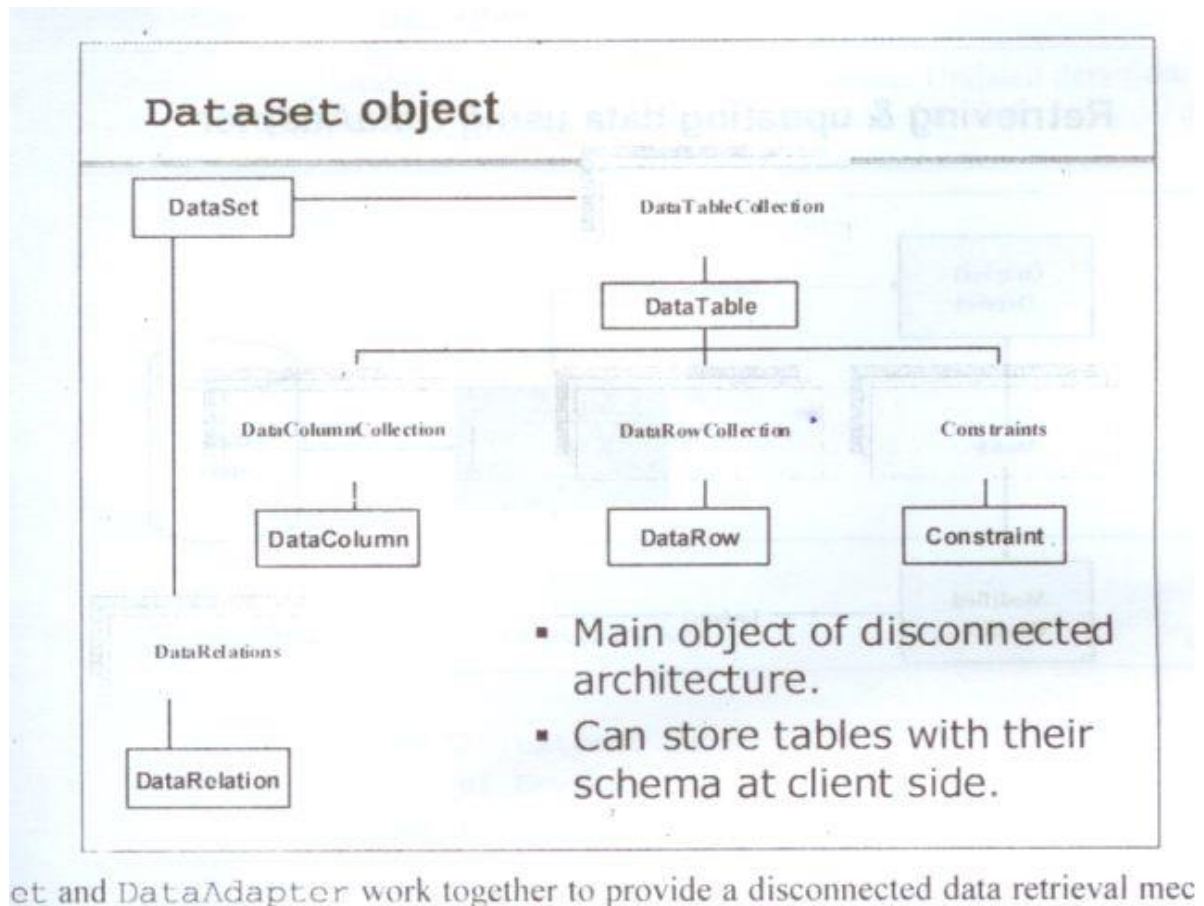


5. DataSet Object

- In the disconnected scenario, the data retrieved from the database is stored in a local buffer called DataSet. It is explicitly designed to access data from any data source. This class is defined in the System.Data namespace.
- A Data Set object is an in-memory representation of the data. It is specially designed to manage data in memory and to support disconnected operations on data.
- A Data Set is a collection of DataTable and DataRelations. Each DataTable is a collection of DataColumn, DataRow, and Constraints.
- A DataTable, DataColumn, and DataRow could be created as follows.

Example

```
1. DataTable dt = new DataTable();  
2. DataColumn col = new DataColumn();  
3. Dt.columns.Add(col2);  
4. DataRow row = dt.newRow();
```

- It is used in a disconnected architecture.
- Provides lower performance.
- A DataSet object has read/write access.
- A DataSet object supports multiple tables from various databases.
- A DataSet object is bound to multiple controls.
- A DataSet object has slower access to data.
- A DataSet object is supported by Visual Studio tools.
- We can create relations in a dataset.
- A Dataset supports integration with XML.
- A DataSet communicates with the Data Adapter only.
- A DataSet can modify data.

6. Command Builder Object

- Automatically generates insert, update, delete queries using the SelectCommand property of a DataAdapter.
- A Command Builder Object is used to build commands for data modification from objects based on a single table query. CommandBuilders are designed depending on the specific data source. The following table shows the CommandBuilder classes with their data source.

Provider-Specific Data Adapter classes	Data Source
SqlDataAdapter	SQL Server
OleDbDataAdapter	OLE DB provider

OdbcDataAdapter	ODBC driver
OracleDataAdapter	Oracle

Example

```

1. da = new SqlDataAdapter("Select * from Employee", con);
2. ds = new DataSet();
3. da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
4. da.Fill(ds, "Emp");
5. bldr = new SqlCommandBuilder(da);
6. dataGridView1.DataSource = ds.Tables["Emp"];

```

Question 3: Differences Between DataReader and DataSet

Answer

No	Data Reader	DataSet
1	Used in a connected architecture	used in a disconnected architecture
2	Provides better performance	Provides lower performance
3	DataReader object has read-only access	A DataSet object has read/write access
4	DataReader object supports a single table based on a single SQL query of one database	A DataSet object supports multiple tables from various databases
5	A DataReader object is bound to a single control	A DataSet object is bound to multiple controls
6	A DataReader object has faster access to data	A DataSet object has slower access to data
7	A DataReader object must be manually coded	A DataSet object is supported by Visual Studio tools
8	We can't create a relation in a data reader	We can create relations in a dataset
9	Whereas a DataReader doesn't support data reader communicates with the command object.	A Dataset supports integration with XML Dataset communicates with the Data Adapter only
10	DataReader cannot modify data	A DataSet can modify data

Question 4: DataView Object

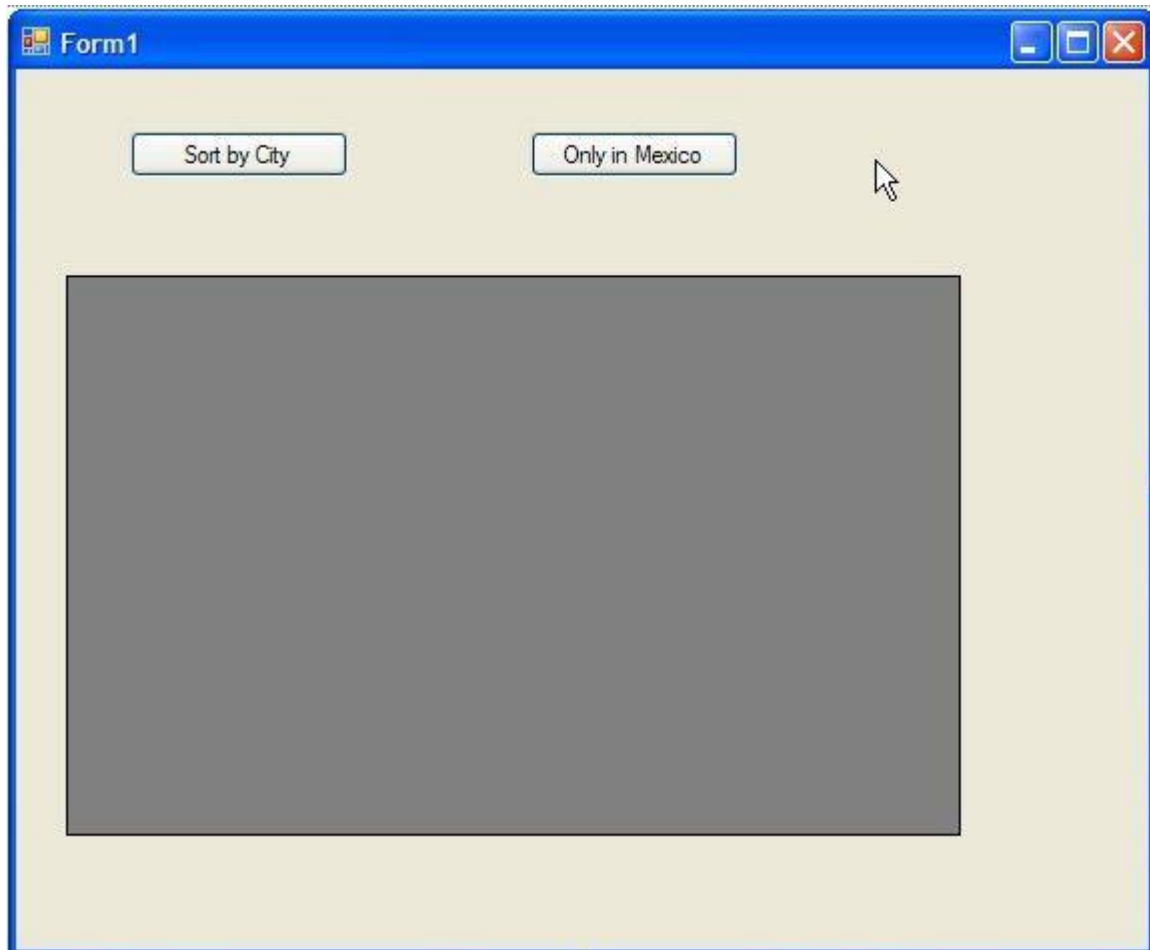
Answer

- A DataView is the same as a read-only mini-dataset.
- You typically load only a subset into a DataView.
- A DataView provides a dynamic view of data. It provides a datarow using the DataView.

Example

Add two buttons and a DataGridView control. Change the text of the first button to sort by city and that of button2 to only select records in Mexico and add the code in form.cs.

Form Design



Coding Part

```
1. SqlConnection con;
2. SqlCommand cmd;
3. SqlDataReader dr;
4. public DataTable GetTable()
5. {
6.     con = new SqlConnection("Data Source=.\sqlexpress;Initial Catalog=information;Integrated Security=True;Pooling=False");
7.     con.Open();
8.     cmd = new SqlCommand("select * from Customers", con);

9.     dr = cmd.ExecuteReader();
10.     DataTable dt = new DataTable();
11.     dt.Load(dr);
12.     dataGridView1.DataSource = dt;
```

```
13.     con.Close();
14.     return dt;
15. }
```

Form1_Load

```
1. dataGridView1.DataSource = GetTable().DefaultView;
```

SortByCity_Click

```
1. DataView dv = new DataView(GetTable());
2. dv.Sort = "City ASC";
3. dataGridView1.DataSource = dv;
```

OnlyInMexico_Click

```
1. DataView dv = new DataView(GetTable());
2. dv.RowFilter = "Country = 'Mexico'";
3. dataGridView1.DataSource = dv;
```

- At the click of the sort by city button, the data already in the DataGridView control is sorted by city.
- On clicking the second button, only the records in Mexico are displayed in the DataGridView control. The output after clicking the only in Mexico button is as the Mexico button.

Working With System.Data.SqlClient and System.Data.OleDb

Program: Design a simple Winform for accepting the details of Employee. Using the connected architecture of ADO.NET, perform the following operations:

- Insert record.
- Search record.
- Update record.
- Delete record.

1. Form Design

Emp_ID	Emp_Name	Salary
11	eweew	2323
33	ewew	232
44	wenwer	43433
55	kkkkkkk	434343
*		

2. Coding Part

Step 1: add namespace using System.Data.SqlClient;for SQL dataBase.

Step 2: Create a connection object.

```
1. SqlConnection con;
2. SqlCommand cmd;
3. SqlDataReader dr;
```

Step3: Form1_Load

```
1. con =new SqlConnection("Data Source=.\sqlexpress
;Initial Catalog=information;Integratedecurity=Tr
ue;Pooling=False");
2. private void Display()
3. {
4. con.Open();
5. cmd =new SqlCommand("select * from Employee", con
);
6. dr = cmd.ExecuteReader();
7. DataTable dt = new DataTable();
8. dt.Load(dr);
9. dataGridView1.DataSource = dt;
10. con.Close();
11. }
```

Insertbtn_Click

```
1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. string bb = textBox2.Text;
4. int cc = Convert.ToInt32(textBox3.Text);
```

```

5. cmd =new SqlCommand("INSERT INTO Employee(Emp_ID,
    Emp_Name,Salary)VALUES ('" + aa + "','" + bb + "'
    ,'" + cc + "')", con);
6. cmd.ExecuteNonQuery();
7. MessageBox.Show("one record inserted:");
8. con.Close();
9. Display();

```

Deletebtn_Click

```

1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. cmd =new SqlCommand("DELETE FROM Employee where E
    mp_ID='" + aa + "'", con);
4. cmd.ExecuteNonQuery();
5. MessageBox.Show("one record Delete:");
6. con.Close();
7. Display();

```

Updatebtn_Click

```

1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. string bb = textBox2.Text;
4. int cc = Convert.ToInt32(textBox3.Text);
5. string abc = "UPDATE Employee SET Emp_ID='" + aa
    + "', Emp_Name='" + bb + "',Salary='" + cc + "'
    ' WHERE Emp_ID = '" + aa + "'";
6. SqlCommand cmd =new SqlCommand(abc, con);
7. cmd.ExecuteNonQuery();
8. MessageBox.Show("one record updated:");
9. con.Close();
10. Display();

```

Displaybtn_Click

```

1. Display();

```

Searchbtn_Click

```

1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. string abc = "SELECT Emp_ID,Emp_Name,Salary FROM
    Employee where Emp_ID='" + aa + "'";
4. cmd =new SqlCommand(abc, con);
5. MessageBox.Show("one record search:");
6. dr = cmd.ExecuteReader();
7. DataTable dt =new DataTable();
8. dt.Load(dr);
9. dataGridView1.DataSource = dt;
10. con.Close();

```

Totalrecordbtn_Click

```

1. con.Open();

```

```

2. cmd = new SqlCommand("select Count(*) from Employ
   ee", con);
3. int a = (int)cmd.ExecuteScalar();
4. label14.Text = "Total Record:--
   > " + a.ToString();
5. con.Close();

```

Exit_Click

```
1. Application.Exit();
```

Program: Design a Simple Winform for accepting the details of an Employee. Using the connected architecture of ADO.NET, perform the following operations:

- Insert record.
- Search record.
- Update record.
- Delete record.

Form Design

The screenshot shows a Windows Form titled "Form1" with a light beige background. On the left, there are three text boxes for "Emp_ID", "Emp_Name", and "Salary". Below these are six buttons arranged in two columns: "Insert", "Delete", "Update", "Display", "SearchRcd", and "TotalRcd". On the right, there is a table with three columns: "Emp_ID", "Emp_Name", and "Salary". The table contains four rows of data: (11, eweew, 2323), (33, ewew, 232), (44, wenwer, 43433), and (55, kkkkkkk, 434343). Below the table is a row with an asterisk (*) in the first column. At the bottom right, there is an "Exit" button. A label "label4" is positioned below the table.

Emp_ID	Emp_Name	Salary
11	eweew	2323
33	ewew	232
44	wenwer	43433
55	kkkkkkk	434343
*		

Coding Part

Step 1: add namespace using System.Data.OleDb; for access Database.

Step 2: Create connection object.

```

1. OleDbConnection con;
2. OleDbCommand cmd;
3. OleDbDataReader dr;

```

Step 3: Form1_Load

```

1. con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Documents and Settings\\Admin\\Desktop\\Ado Connected Demo\\db1.mdb");
2. private void display()
3. con.Open();
4. cmd = new OleDbCommand("select * from Employee",con);
5. dr = cmd.ExecuteReader();
6. DataTable dt = new DataTable();
7. dt.Load(dr);
8. dataGridView1.DataSource = dt;
9. con.Close();

```

Display_Click

```

1. display();

```

Insert_Click

```

1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. string bb = textBox2.Text;
4. int cc = Convert.ToInt32(textBox3.Text);
5. cmd = new OleDbCommand("INSERT INTO Employee(Emp_ID, Emp_Name,Salary) VALUES ('" + aa + "', '" + bb + "', '" + cc + "')", con);
6. cmd.ExecuteNonQuery();
7. MessageBox.Show("one record inserted:");
8. con.Close();
9. display();

```

Delete_Click

```

1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. cmd = new OleDbCommand("DELETE FROM Employee where Emp_ID=" + aa + "", con);
4. cmd.ExecuteNonQuery();
5. MessageBox.Show("one record Delete:");
6. con.Close();
7. display();

```

update_Click

```

1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. string bb = textBox2.Text;
4. int cc = Convert.ToInt32(textBox3.Text);
5. string abc = "UPDATE Employee SET Emp_ID = '" + aa + "', Emp_Name = '" + bb + "', Salary = '" + cc + "' WHERE Emp_ID = '" + aa + "'";
6. OleDbCommand cmd = new OleDbCommand(abc, con);
7. cmd.ExecuteNonQuery();
8. MessageBox.Show("one record updated:");
9. con.Close();

```



```
10. display();
```

Find_Click

```
1. con.Open();
2. int aa = Convert.ToInt32(textBox1.Text);
3. string abc = "SELECT Emp_ID,Emp_Name,Salary FROM Employee
   where Emp_ID='" + aa + "'";
4. cmd = new OleDbCommand(abc, con);
5. MessageBox.Show("one record search:");
6. dr = cmd.ExecuteReader();
7. DataTable dt = new DataTable();
8. dt.Load(dr);
9. dataGridView1.DataSource = dt;
10. con.Close();
```

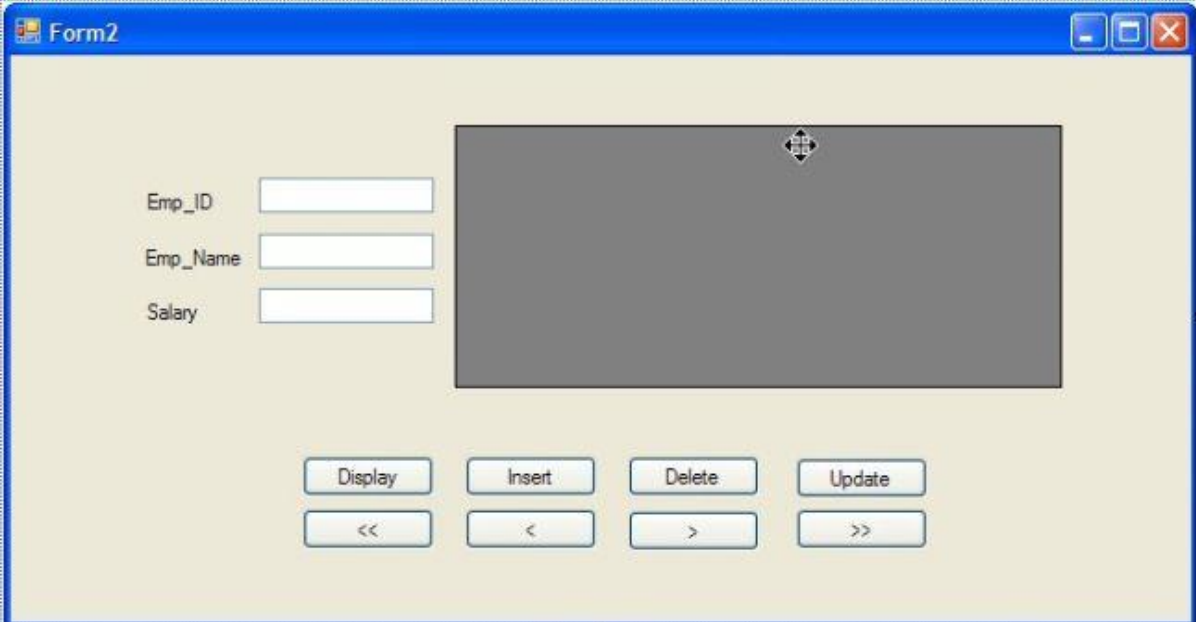
Exit_Click

```
Application.Exit();
```

Program: Design a simple Winform for accepting the details of an Employee. Using the disconnected architecture of ADO.NET, perform the following operations.

- Insert record.
- Display record.
- Update record.
- Delete record

Form Design



Coding Part

Step 1: add namespace using System.Data.SqlClient;for SQL dataBase.

Step 2: Create connection object.

```
1. DataSet ds;  
2. SqlDataAdapter da;  
3. SqlConnection con;  
4. SqlCommandBuilder bldr;
```

Step 3: Form1_Load

```
1. con = new SqlConnection("Data Source=.\sqlexpress;Initial  
    Catalog=information;IntegratedSecurity=True;Pooling=False  
    ");  
2. private void display()  
3. da = new SqlDataAdapter("Select * from Employee", con);  
4. ds = new DataSet();  
5. da.MissingSchemaAction = MissingSchemaAction.AddWithKey;  
6. da.Fill(ds, "Emp");  
7. bldr = new SqlCommandBuilder(da);  
8. dataGridView1.DataSource = ds.Tables["Emp"];
```

Display_Click

```
1. display();
```

Insert_Click

```
1. DataRow drnew = ds.Tables["Emp"].NewRow();  
2. drnew[0] = textBox1.Text;  
3. drnew[1] = textBox2.Text;  
4. drnew[2] = textBox3.Text;  
5. ds.Tables["Emp"].Rows.Add(drnew);  
6. da.Update(ds, "Emp");  
7. MessageBox.Show("Record added");  
8. dataGridView1.DataSource = ds.Tables["Emp"];
```

Delete_Click

```
1. DataRow row = ds.Tables["Emp"].Rows.Find(Convert.ToInt32(t  
    extBox1.Text));  
2. row.Delete();  
3. da.Update(ds, "Emp");  
4. MessageBox.Show("Record Deleted");  
5. dataGridView1.DataSource = ds.Tables["Emp"];
```

Update_Click

```
1. DataRow dr = ds.Tables[0].Rows.Find(textBox1.Text);  
2. dr["Emp_Name"] = textBox2.Text;  
3. dr["Salary"] = textBox3.Text;  
4. da.Update(ds, "Emp");  
5. MessageBox.Show("updated..");  
6. dataGridView1.DataSource = ds.Tables[0];
```

FirstRcd_Click

```
1. this.BindingContext[ds.Tables[0]].Position = 0;
```

LastRcd_Click

```
1. this.BindingContext[ds.Tables[0]].Position = ds.Tables[0].  
   Rows.Count - 1;
```

NextRcd_Click

```
1. if (this.BindingContext[ds.Tables[0]].Position > 0)  
2. {  
3.     this.BindingContext[ds.Tables[0]].Position -= 1;  
4. }
```

PreviousRcd_Click

```
1. if (this.BindingContext[ds.Tables[0]].Position < ds.Tables  
   [0].Rows.Count - 1)  
2. {  
3.     this.BindingContext[ds.Tables[0]].Position += 1;  
4. }
```