Unit 3

Benefits of ADO.NET

There are many benefits of ADO.NET.

- ADO.NET has one object-oriented class. All data providers use the same programming model to operate with diverse data sources.
- The managed classes used by ADO.NET are classes. They utilize all of the .NET CLR's benefits, including automated resource management and language independence.
- XML is used to transport ADO.NET data. This data may be shared by all components and apps and sent over HTTP.
- Visual Studio .NET provides ADO.NET components and data-bound controls for use in visual form.
- Performance and scalability are two important considerations when creating web-based apps and services.
- Data that can only be read and sent from a database is retrieved using the ADO.NET DataReader.

ADO.NET Compared to Classic ADO

Head to Head differences	ADO	ADO.NET
1 Data Representation	In ADO, data is represented by a RecordSet object, which can hold data from one data source at a time.	In ADO.NET, data is represented by a DataSet object, which can hold data from various data sources, integrate it, and after combining all the data it can write it back

		to one or several data sources.
2 Data Reads	In ADO, data is read in sequential manner.	In ADO.NET, data is read either in a sequential or non-sequential manner.
3 Disconnected Access	ADO first makes a call to an OLE DB provider. It supports the connected access, represented by the Connection with a database.	It uses standardized calls such as DataSetCommand object to communicate with a database and with the OLE DB provider.
4 Creating Cursors	It allows us to create client-side cursor only.	ADO.NET either created client-side or server-side cursors.
5 Locking	ADO has the feature of locking.&	This features is not used in ADO.NET.
6 XML integration	This feature is not used in ADO.	This feature is used in ADO.NET.
7 Relationship between Multiple Tables	It requires SQL Join and UNIONs to combine data from multiple tables in a single RecordSet.	It uses DataRelation objects for combining data from multiple tables without the use of SQL Join and UNIONs.
8 Conversion	ADO requires data to be converted to data types supported by receiving system.	ADO.NET does not require complex conversions that wasted processor time.
9 Data Storage	In ADO data is stored in a binary form.	In ADO.NET, data is stored in XML form. With the use of XML, it stores data more easily.
10 Firewalls	In ADO, firewalls are typically configured to prevent system-level	In ADO.NET firewalls are supported because ADO.NET DataSet objects use

	requests, such as COM marshaling.	XML for representing data.
11 Communication	In this, objects are communicate in a binary mode.	It uses XML for communication.
12 Programmability	It uses a Connection object to transfer commands to a data source with defined data constructs.	ADO.NET does not require data constructs because it uses strongly typed characteristics of XML.
13 Connection Model	ADO supports only one model of connection i.e. Connection Oriented Data Access Architecture Model.	ADO.NET supports two models of connection i.e. Connection-Oriented Data Access Architecture and Disconnected Oriented Data Access Architecture.
14. Environment	ADO was geared for a two-tiered architecture.	ADO.NET address a multi-tiered architecture.
15. Metadata	It derives information automatically at runtime based on metadata.	In ADO.NET derives information at design time on metadata in order to provide better runtime performance.
16. Multiple Transactions	In ADO multiple transactions cannot be send using a single connection.	In ADO.NET multiple transactions can be send using a single connection.
17. Data Types	It has a less number of data types.	It has a huge and rich collection of data types.
18. Serialization	ADO serialized data in a proprietary protocol.	ADO.NET serialized data using XML.
19. Security	ADO is less secured.	ADO.NET much secured as compared to ADO.

20 Based on	ADO is based on Component Object Modelling.	ADO.NET is a Common Language Runtime based library.
21 Scalability	ADO technology have less scalable.	ADO.NET have more scalability with the use of locking mechanism.
22. Resources	ADO technology use more resources as compared to ADO.NET.	ADO.NET conserves limited resources.
23. Performance	ADO have a poor performance.	In ADO.NET performance is much better as compared to ADO.

ADO.NET Architecture

ADO.NET supports both Connection-Oriented Architectures as well as Disconnection-Oriented Architecture. Depending upon the functionality or business requirement of an application, we can make it either Connection-Oriented or Disconnection-Oriented. Even, it is also possible to use both Architectures together in a single .NET application to communicate with different data sources.

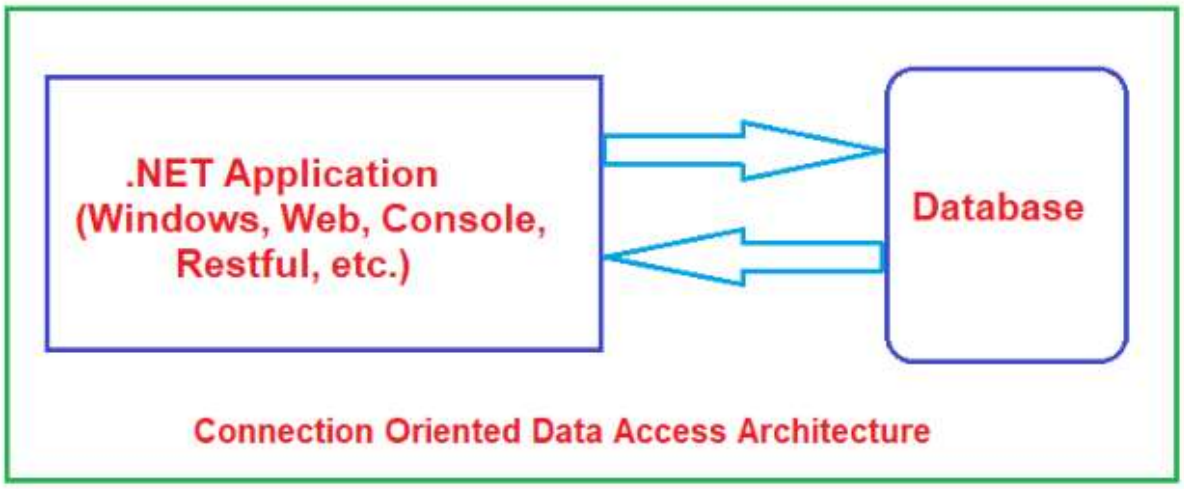
ADO.NET Connection-Oriented Data Access Architecture:

In the case of Connection Oriented Data Access Architecture, always an open and active connection is required in between the .NET Application and the database. An example is Data Reader and when we are accessing the data from the database, the Data Reader object requires an active and open connection to access the data, If the connection is closed then we cannot access the data from the database and in that case, we will get the runtime error.

The Connection Oriented Data Access Architecture is always forward only. That means using this architecture mode, we can only access the data in the forward direction. Once we read a row, then it will move to the next data row and there is no chance to move back to the previous row.

The Connection Oriented Data Access Architecture is read-only. This means using this architecture we can only read the data. We cannot modify the data i.e. we cannot update and delete the data row.

For Connection Oriented Architecture, we generally use the object of the ADO.NET DataReader class. The DataReader object is used to retrieve the data from the database and it also ensures that an open and active connection should be there while accessing the data from the database. In Connection Oriented Architecture, the .NET Application is directly linked with the corresponding Database.



ADO.NET DataReader in Connected-Oriented Architecture

The ADO.NET DataReader object is used to read the data from the database using Connected Oriented Architecture. It works in forward only and only mode. It requires an active and open connection while reading the data from the database.

Example:

```
CREATE DATABASE EmployeeDB;

USE EmployeeDB;

CREATE TABLE Employee(

 Id INT IDENTITY(100, 1) PRIMARY KEY,

 Name VARCHAR(100),

 Email VARCHAR(50),

 Mobile VARCHAR(50),

)

INSERT INTO Employee VALUES ('Anurag','Anurag@dotnettutorial.net','1234567890')
```

INSERT INTO Employee VALUES ('Priyanka','Priyanka@dotnettutorial.net','2233445566')

INSERT INTO Employee VALUES ('Preety','Preety@dotnettutorial.net','6655443322')

INSERT INTO Employee VALUES ('Sambit','Sambit@dotnettutorial.net','9876543210')

Using ADO.NET Data Reader to Fetch the Data from the Database:

```
using System;
using System.Data.SqlClient;
namespace ConnectionOrientedArchitecture
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
string ConString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=EmployeeDB; integrated security=true";
            using (SqlConnection connection = new SqlConnection(ConString))
```

```
        {
            // Creating the command object
SqlCommand cmd = new SqlCommand("select Name, Email, Mobile from Employee", connection);
            // Opening Connection
            connection.Open();
            // Executing the SQL query
            SqlDataReader sdr = cmd.ExecuteReader();
            //Looping through each record
            //SqlDataReader works in Connection Oriented Architecture
            //So, it requires an active and open connection while reading the data
            //from the database
            while (sdr.Read())
            {
      Console.WriteLine(sdr["Name"] + ",  " + sdr["Email"] + ",  " + sdr["Mobile"]);
            }
        }//Here, the connection is going to be closed automatically
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception Occurred: {ex.Message}");
```

```
        }
        Console.ReadKey();
    }
  }
}
```

Now, let us do one thing. After reading the first row from the database, let us close the connection and see what happens. In the below example, we can see, within the while loop, after reading the first row, we are closing the database connection by calling the Close method.

```
using System;
using System.Data.SqlClient;
namespace ConnectionOrientedArchitecture
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
```

```
        string ConString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=EmployeeDB; integrated security=SSPI";
        using (SqlConnection connection = new SqlConnection(ConString))
        {
            // Creating the command object
            SqlCommand cmd = new SqlCommand("select Name, Email, Mobile from Employee", connection);
            // Opening Connection
            connection.Open();
            // Executing the SQL query
            SqlDataReader sdr = cmd.ExecuteReader();
            //Looping through each record
            //SqlDataReader works in Connection Oriented Architecture
            //So, it requires an active and open connection while reading the data
            //from the database
            while (sdr.Read())
            {
                //Read-only, we cannot modify the data
                //sdr["Name"] = "PKR";
    Console.WriteLine(sdr["Name"] + ",  " + sdr["Email"] + ",  " + sdr["Mobile"]);
                connection.Close();//Here, the connection is closed
            }
```

```
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
  }
}
```

As we can see in the above output, after reading the first row, the data reader throws an exception, and the reason the database connection is closed. So, this proves that connection-oriented architecture always requires an active and open connection to the database.

ADO.NET Disconnection-Oriented Data Access Architecture:

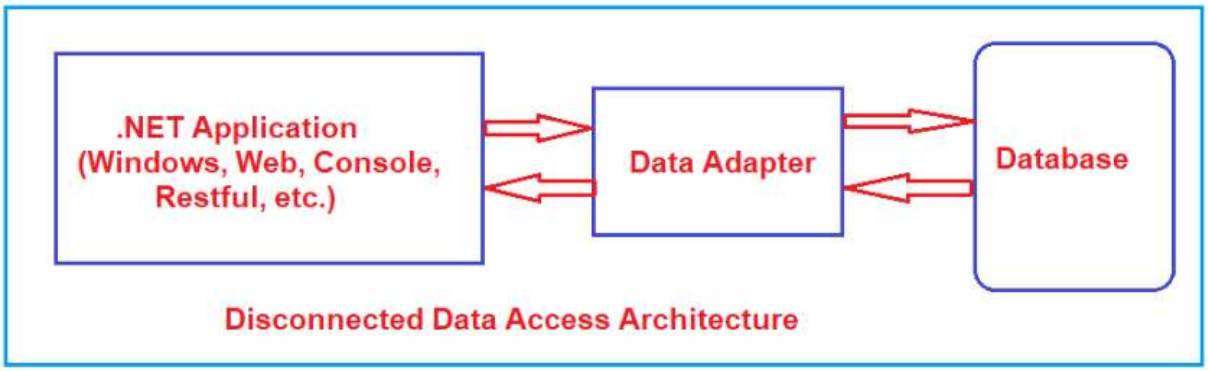
In the case of Disconnection Oriented Data Access Architecture, always an open and active connection is not required in between the .NET Application and the database. In this architecture, Connectivity is required only to read the data from the database and to update the data within the database.

An example is DataAdapter and DataSet or DataTable classes. Here, using the DataAdapter object and using an active and open connection, we can retrieve the data from the database and store the data in a DataSet or DataTable. The DataSets or DataTables are in-memory objects or we can say they store the data temporarily within .NET Application. Then whenever required in

our .NET Application, we can fetch the data from the dataset or data table and process the data. Here, we can modify the data, we can insert new data, can delete the data from within the dataset or data tables. So, while processing the data within the .NET Application using DataSet or Datatable, we do not require an active and open connection.

And finally, when we processed the data in our .NET Application, then if we want to update the modified data which is stored inside the dataset or Datatable into the database, then we need to establish the connection again and we need to update the data in the database.

The ADO.NET DataAdapter object acts as an interface between the .NET application and the database. The Data Adapter object fills the Dataset or DataTable which helps the user to perform the operations on the data. And once we modify the DataSet or DataTable, then we need to pass the modified DataSet or DataTable to the DataAdapter which will update the modified data into the database. The DataAdapter object will internally manage the connection i.e. when to establish the connection and when to terminate the connection.



The ADO.NET DataAdapter establishes a connection with the corresponding database and then retrieves the data from the database and fills the retrieved data into the Dataset or DataTable. And finally, when the task is completed i.e. the Data is processed by the application i.e. the data is modified by the application, and modified data is stored in the DataSet or DataTable. Then the

DataAdapter takes the modified data from the DataSet or DataTable and updates it into the database by again establishing the connection.

So, we can say that DataAdapter acts as a mediator between the Application and database which allows the interaction in disconnection-oriented architecture.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace BatchOperationUsingSqlDataAdapter
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            // Connection string.
            string connectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=EmployeeDB; integrated security=true";
            // Connect to the EmployeeDB database.
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
```

```
            // Create a SqlDataAdapter
  SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM EMPLOYEE", connection);
            //Fetch the Employee Data and Store it in the DataTable
            DataTable dataTable = new DataTable();
            //The Fill method will open the connection, fetch the data, fill the data in
            //the data table and close the connection automatically
            adapter.Fill(dataTable);
            // Set the UPDATE command and parameters.
            string UpdateQuery = "UPDATE Employee SET Name=@Name, Email=@Email, Mobile=@Mobile WHERE ID=@EmployeeID;";
            adapter.UpdateCommand = new SqlCommand(UpdateQuery, connection);
            adapter.UpdateCommand.Parameters.Add("@Name", SqlDbType.NVarChar, 50, "Name");
            adapter.UpdateCommand.Parameters.Add("@Email", SqlDbType.NVarChar, 50, "Email");
            adapter.UpdateCommand.Parameters.Add("@Mobile", SqlDbType.NVarChar, 50, "Mobile");
            adapter.UpdateCommand.Parameters.Add("@EmployeeID", SqlDbType.Int, 4, "ID");
            //Set UpdatedRowSource value as None
            //Any Returned parameters or rows are Ignored.
            adapter.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;
```

```
            //Change the Column Values of Few Rows
            DataRow Row1 = dataTable.Rows[0];
            Row1["Name"] = "Name Changed";
            DataRow Row2 = dataTable.Rows[1];
            Row2["Email"] = "Email Changed";
            DataRow Row3 = dataTable.Rows[2];
            Row2["Mobile"] = "Mobile Changed";
            // Execute the update.
            //The Update method will open the connection, execute the Update command by
takking
            //the data table data and then close the connection automatically
            adapter.Update(dataTable);
            Console.WriteLine($"Updated Data Saved into the DataBase");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception Occurred: {ex.Message}");
    }
    Console.ReadKey();
}
```

```
    }
}
```

ADO.NET Connected Oriented Architecture:

- It is connection-oriented data access architecture. It means always an active and open connection is required.
- Using the DataReader object we can implement the Connected Oriented Architecture.
- Connected Oriented Architecture gives a faster performance.
- Connected Oriented Architecture can hold the data of a single table only.
- We can access the data in a forward-only and read-only manner.
- Using Data Reader, we cannot persist the data in the database.

ADO.NET Disconnected Oriented Architecture:

- It is disconnection-oriented data access architecture. It means always an active and open connection is not required.
- Using DataAdapter and DataSet or Datatable we can implement Dis-Connected Oriented Architecture.
- Disconnected Oriented Architecture gives a lower performance.
- Disconnected Oriented Architecture can hold the data of multiple tables using dataset and single table data using Datatable.
- We can access the data in forward and backward directions and we can also modify the data.
- Using Data Adapter, we can persist the DataSet or DataTable data into the database.

Working with Dataset

The DataSet represents a subset of the database in memory. That means the ADO.NET DataSet is a collection of data tables that contains the relational data in memory in tabular format. The ADO.NET DataSet class is the core component for providing data access in a distributed and disconnected environment. The ADO.NET DataSet class belongs to the **System.Data** namespace.

Creating Customers Data Table:

We have already discussed how to create Data Table in our previous article. If we have not read that article then please our **ADO.NET DataTable** article before proceeding to this article. Please have a look at the below image. As we can see, here, we created one **DataTable** with the name **Customer**. Then we created three data columns (ID of type Int32, Name of type string, and Mobile of type string) and added these three columns to the Customer data table. Finally, we created two data rows and add these two data rows to the Customer data table.

```
// Craeting Customer table
DataTable Customer = new DataTable("Customer");

//Creating column and schema
DataColumn CustomerId = new DataColumn("ID", typeof(Int32));
Customer.Columns.Add(CustomerId);
DataColumn CustomerName = new DataColumn("Name", typeof(string));
Customer.Columns.Add(CustomerName);
DataColumn CustomerMobile = new DataColumn("Mobile", typeof(string));
Customer.Columns.Add(CustomerMobile);

//Adding Data Rows into Customer table
Customer.Rows.Add(101, "Anurag", "2233445566");
Customer.Rows.Add(202, "Manoj", "1234567890");
```

Creating Orders Data Table:

Please have a look at the following image. Here, we can see, we created the DataTable with the name Orders. Then we created three data columns (Id of type Int32, CustomerId of type Int32, and Amount of type Int32) and add these three data columns to the Orders data table. Finally, we created two data rows and add these two data rows to the Orders data table.

```
// Craeting Orders table
DataTable Orders = new DataTable("Orders");

//Creating column and schema
DataColumn OrderId = new DataColumn("ID", typeof(Int32));
Orders.Columns.Add(OrderId);
DataColumn CustId = new DataColumn("CustomerId", typeof(Int32));
Orders.Columns.Add(CustId);
DataColumn OrderAmount = new DataColumn("Amount", typeof(Int32));
Orders.Columns.Add(OrderAmount);

//Adding Data Rows into Orders table
Orders.Rows.Add(10001, 101, 20000);
Orders.Rows.Add(10002, 102, 30000);
```

Creating DataSet with DataTable:

As we already discussed the DataSet is a collection of DataTables. So, let's create a DataSet object and then add the two data tables (Customers and Orders) into the DataSet. Please have a look at the following image. Here, first, we created an instance of the DataSet and then add the two data tables using the Tables property of the DataSet object.

```
//Creating DataSet object
DataSet dataSet = new DataSet();

//Adding DataTables into DataSet
dataSet.Tables.Add(Customer);
dataSet.Tables.Add(Orders);
```

Fetch DataTable from DataSet:

Now, let us see how to fetch the data table from the dataset. We can fetch the data table from a dataset in two ways i.e. using the index position and using the table name (if provided).

Fetching DataTable from DataSet using index position:

As we first add the Customers table to DataSet, the Customer table Index position is 0. If we want to iterate through the Customer's table data, then we could use a for-each loop to iterate as shown in the below image.

```
//Fetching DataTable from dataset using the Index position
foreach (DataRow row in dataSet.Tables[0].Rows)
{
    Console.WriteLine(row["ID"] + ",  " + row["Name"] + ",  " + row["Mobile"]);
}
```

Fetching DataTable From DataSet using Name:

The second data table that we added to the dataset is Orders and it will be added at index position 1. Further, if we notice while creating the data table we have provided a name for the data table i.e. Orders. Now, if we want to fetch the data table from the dataset, then we can use the name instead of the index position. The following image shows how to fetch the data table using the name and looping through the data using a for each loop.

```
//Fetching DataTable from the DataSet using the table name
foreach (DataRow row in dataSet.Tables["Orders"].Rows)
{
    Console.WriteLine(row["ID"] + ",  " + row["CustomerId"] + ",  " + row["Amount"]);
}
```

Example:

```
using System;

using System.Data;

namespace AdoNetConsoleApplication

{

   class Program

   {

      static void Main(string[] args)

      {

         try

         {

            // Creating Customer Data Table
```

```
DataTable Customer = new DataTable("Customer");


// Adding Data Columns to the Customer Data Table

DataColumn CustomerId = new DataColumn("ID", typeof(Int32));

Customer.Columns.Add(CustomerId);

DataColumn CustomerName = new DataColumn("Name", typeof(string));

Customer.Columns.Add(CustomerName);

DataColumn CustomerMobile = new DataColumn("Mobile", typeof(string));

Customer.Columns.Add(CustomerMobile);


//Adding Data Rows into Customer Data Table

Customer.Rows.Add(101, "Anurag", "2233445566");

Customer.Rows.Add(202, "Manoj", "1234567890");


// Creating Orders Data Table

DataTable Orders = new DataTable("Orders");


// Adding Data Columns to the Orders Data Table

DataColumn OrderId = new DataColumn("ID", typeof(System.Int32));
```

```
Orders.Columns.Add(OrderId);

DataColumn CustId = new DataColumn("CustomerId", typeof(Int32));

Orders.Columns.Add(CustId);

DataColumn OrderAmount = new DataColumn("Amount", typeof(int));

Orders.Columns.Add(OrderAmount);


//Adding Data Rows into Orders Data Table

Orders.Rows.Add(10001, 101, 20000);

Orders.Rows.Add(10002, 102, 30000);

//Creating DataSet Object

DataSet dataSet = new DataSet();


//Adding DataTables into DataSet

dataSet.Tables.Add(Customer);

dataSet.Tables.Add(Orders);


//Fetching Customer Data Table Data

Console.WriteLine("Customer Table Data: ");

//Fetching DataTable from Dataset using the Index position
```

```
        foreach (DataRow row in dataSet.Tables[0].Rows)
        {
            //Accessing the data using string column name
         Console.WriteLine(row["ID"] + ",  " + row["Name"] + ",  " + row["Mobile"]);
            //Accessing the data using integer index position
            //Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
        }

        Console.WriteLine();

        //Fetching Orders Data Table Data
        Console.WriteLine("Orders Table Data: ");
        //Fetching DataTable from the DataSet using the table name
        foreach (DataRow row in dataSet.Tables["Orders"].Rows)
        {
            //Accessing the data using string column name
Console.WriteLine(row["ID"] + ",  " + row["CustomerId"] + ",  " + row["Amount"]);

            //Accessing the data using integer index position
```

```
            //Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception Occurred: {ex.Message}");
    }
    Console.ReadKey();
  }
 }
}
```

Constructors of DataSet in C#:

7

The DataSet in C# provides the following four constructors.

```
public DataSet();
public DataSet(string dataSetName);
protected DataSet(SerializationInfo info, StreamingContext context);
protected DataSet(SerializationInfo info, StreamingContext context, bool ConstructSchema);
```

1. **DataSet():** It initializes a new instance of the System.Data.DataSet class..
2. **DataSet(string dataSetName):** It initializes a new instance of a System.Data.DataSet class with the given name. Here, the string parameter dataSetName specifies the name of the System.Data.DataSet.
3. **DataSet(SerializationInfo info, StreamingContext context):** It initializes a new instance of a System.Data.DataSet class that has the given serialization information and

context. Here, the parameter info is the data needed to serialize or deserialize an object. The context specifies the source and destination of a given serialized stream.

4. **DataSet(SerializationInfo info, StreamingContext context, bool ConstructSchema):** It initializes a new instance of the System.Data.DataSet class.

Properties of DataSet in C#:

The DataSet class provides the following properties.

1. **CaseSensitive**: It is used to get or set a value indicating whether string comparisons within System.Data.DataTable objects are case-sensitive. It returns true if string comparisons are case-sensitive; otherwise false. The default is false.
2. **DefaultViewManager**: It is used to get a custom view of the data contained in the System.Data.DataSet to allow filtering, searching, and navigating using a custom System.Data.DataViewManager.
3. **DataSetName**: It is used to get or set the name of the current System.Data.DataSet.
4. **EnforceConstraints**: It is used to get or set a value indicating whether constraint rules are followed when attempting any update operation.
5. **HasErrors**: It is used to get a value indicating whether there are errors in any of the System.Data.DataTable objects within this System.Data.DataSet.
6. **IsInitialized**: It is used to get a value that indicates whether the System.Data.DataSet is initialized. It returns true to indicate the component has completed initialization; otherwise false.
7. **Prefix**: It is used to get or set an XML prefix that aliases the namespace of the System.Data.DataSet.
8. **Locale**: It is used to get or set the locale information used to compare strings within the table.
9. **Namespace**: It is used to get or set the namespace of the System.Data.DataSet.
10. **Site**: It is used to get or set up a System.ComponentModel.ISite for the System.Data.DataSet.
11. **Relations**: It is used to get the collection of relations that link tables and allow navigation from parent tables to child tables.

12. **Tables**: It is used to get the collection of tables contained in the System.Data.DataSet.

Methods of ADO.NET DataSet Class:

Following are the methods provided by C# DataSet Class.

1. **BeginInit():** It Begins the initialization of a System.Data.DataSet that is used on a form or used by another component. The initialization occurs at run time.
2. **Clear():** It Clears the System.Data.DataSet of any data by removing all rows in all tables.
3. **Clone():** It Copies the structure of the System.Data.DataSet, including all System.Data.DataTable schemas, relations, and constraints. Do not copy any data.
4. **Copy():** It Copies both the structure and data for this System.Data.DataSet.
5. **CreateDataReader():** It Returns a System.Data.DataTableReader with one result set per System.Data.DataTable, in the same sequence as the tables, appears in the System.Data.DataSet.Tables collection.
6. **CreateDataReader(params DataTable[] dataTables):** It returns a System.Data.DataTableReader with one result set per System.Data.DataTable. Here, the parameter dataTables specifies an array of DataTables providing the order of the result sets to be returned in the System.Data.DataTableReader
7. **EndInit():** It Ends the initialization of a System.Data.DataSet that is used on a form or used by another component. The initialization occurs at run time.
8. **GetXml():** It Returns the XML representation of the data stored in the System.Data.DataSet.
9. **GetXmlSchema():** It Returns the XML Schema for the XML representation of the data stored in the System.Data.DataSet.

Which one to use DataReader or DataSet?

DataSet to use:

1. When we want to cache the data locally in our application so that we can manipulate the data.
2. When we want to work with disconnected architecture.

DataReader to use:

1. If we do not want to cache the data locally, then you need to use DataReader which will improve the performance of our application.
2. DataReader works on connected-oriented architecture i.e. it requires an open connection to the database.

ADO.NET DataSet using SQL Server Database

Let us understand how to use DataSet in C# to fetch and store the data from the SQL Server Database with an example. We are going to use the following Customers and Orders tables to understand the ADO.NET DataSet.

DataSet using SQL Server

Please use the below SQL Script to create a database and tables and populate the Customers and Orders tables with the required test data.

```
CREATE DATABASE ShoppingCartDB;
USE ShoppingCartDB;
CREATE TABLE Customers(
 ID INT PRIMARY KEY,
 Name VARCHAR(100),
 Mobile VARCHAR(50)
)
INSERT INTO Customers VALUES (101, 'Anurag', '1234567890')
INSERT INTO Customers VALUES (102, 'Priyanka', '2233445566')
INSERT INTO Customers VALUES (103, 'Preety', '6655443322')

CREATE TABLE Orders(
 ID INT PRIMARY KEY,
```

```
 CustomerId INT,
 Amount INT
)
INSERT INTO Orders VALUES (10011, 103, 20000)
INSERT INTO Orders VALUES (10012, 101, 30000)
INSERT INTO Orders VALUES (10013, 102, 25000)
```

Example to Understand ADO.NET DataSet using SQL Server Database in C#:

Our business requirement is to fetch all the data from the Customers table and then need to display it on the console. The following example exactly does the same using DataSet. In the below example, we created an instance of the DataSet and then fill the dataset using the Fill method of the data adapter object. The following example is self-explained, so please go through the comment lines.

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

namespace AdoNetConsoleApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=ShoppingCartDB; integrated security=SSPI";
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
```

```
            //Create the SqlDataAdapter instance by specifying the command text and connection object
            SqlDataAdapter dataAdapter = new SqlDataAdapter("select * from customers", connection);

            //Creating DataSet Object
            DataSet dataSet = new DataSet();

            //Filling the DataSet using the Fill Method of SqlDataAdapter object
            //Here, we have not specified the data table name and the data table will be created at index position 0
            dataAdapter.Fill(dataSet);

            //Iterating through the DataSet
    //First fetch the Datatable from the dataset and then fetch the rows using the Rows property of Datatable
            foreach (DataRow row in dataSet.Tables[0].Rows)
            {
                //Accessing the Data using the string column name as key
              Console.WriteLine(row["Id"] + ",  " + row["Name"] + ",  " + row["Mobile"]);
            }
          }
        }
        catch (Exception ex)
        {
           Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
      }
   }
}
```

By default, the dataset assigns a name to the table as Table, Table1, and Table2. So, the above example can be rewritten as shown below and it should give the same output as the previous example. As we can see, here, we are fetching the table using the name (Table).

DataSet with Multiple Database Tables using SQL Server in C#:

It is also possible that your SQL Query may return multiple tables. Let us understand this with an example. Now our business requirement is to fetch the Customers as well as Orders table data which needs to display on the Console. Here, you can access the first table from the dataset using an integral index 0 or string Table name. On the other hand, you can access the second table using the integral index 1 or the string name Table1.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=ShoppingCartDB; integrated security=SSPI";
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
```

```
//We have written two Select Statements to return data from customers and orders table
SqlDataAdapter dataAdapter = new SqlDataAdapter("select * from customers; select * from orders", connection);
DataSet dataSet = new DataSet();
//Data Table 1 will be customers data which is at Index Position 0
//Data Table 2 will be orders data which is at Index Position 1
dataAdapter.Fill(dataSet);
// Fetching First Table Data i.e. Customers Data
Console.WriteLine("Table 1 Data");
//Accessing the Data Table from the DataSet using Integer Index Position
foreach (DataRow row in dataSet.Tables[0].Rows)
{
   //Accessing using string column name as keys
   Console.WriteLine(row["Id"] + ",  " + row["Name"] + ",  " + row["Mobile"]);
   //Accessing using integer index position as keys
   //Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
}
Console.WriteLine();
// Fetching Second Table Data i.e. Orders Data
Console.WriteLine("Table 2 Data");
//Accessing the Data Table from the DataSet using Integer Index Position
foreach (DataRow row in dataSet.Tables[1].Rows)
{
   //Accessing using string column name as keys
   //Console.WriteLine(row["Id"] + ",  " + row["CustomerId"] + ",  " + row["Amount"]);
   //Accessing using integer index position as keys
   Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
}
```

```
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
  }
}
```

Accessing Data table from DataSet Using Default Table Name in C#:

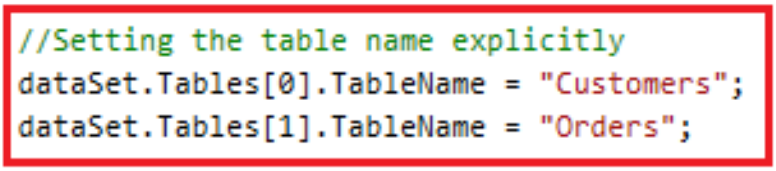
```
using System;
using System.Data;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=ShoppingCartDB; integrated security=SSPI";
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
          //We have written two Select Statements to return data from customers and orders table
```

```
            SqlDataAdapter dataAdapter = new SqlDataAdapter("select * from customers;
select * from orders", connection);
            DataSet dataSet = new DataSet();
            //Data Table 1 will be customers data which is at Index Position 0
            //Data Table 2 will be orders data which is at Index Position 1
            dataAdapter.Fill(dataSet);
            // Fetching First Table Data i.e. Customers Data
            Console.WriteLine("Table 1 Data");
            //Accessing the Data Table from the DataSet using Default Table name
            //By Default, first table name is Table
            foreach (DataRow row in dataSet.Tables["Table"].Rows)
            {
               //Accessing using string column name as keys
               Console.WriteLine(row["Id"] + ",  " + row["Name"] + ",  " +
row["Mobile"]);
               //Accessing using integer index position as keys
               //Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
            }
            Console.WriteLine();
            // Fetching Second Table Data i.e. Orders Data
            Console.WriteLine("Table 2 Data");
            //Accessing the Data Table from the DataSet using Default Table name
            //By Default, second table name is Table1
            foreach (DataRow row in dataSet.Tables["Table1"].Rows)
            {
               //Accessing using string column name as keys
               //Console.WriteLine(row["Id"] + ",  " + row["CustomerId"] + ",  " +
row["Amount"]);
               //Accessing using integer index position as keys
               Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
            }
```

```
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
  }
}
```

Set the Data Table name Explicitly in ADO.NET DataSet

If our dataset going to contain multiple tables of data, then it is very difficult for us to identify using the integral index position or using the default table names. In such a scenario, it is always recommended to provide an explicit name for the data table. Let us understand this with an example. Now, we need to set the first table as Customers and the second table as Orders and then we need to use these custom table names to fetch the actual table data. We can set the table name using the TableName property as shown in the below image.



The following is the complete example that uses the tableName property of the dataset object to set and get the table name. The following example code is self-explained, so please go through the comment lines.

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

```
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=ShoppingCartDB; integrated security=SSPI";
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
               //We have written two Select Statements to return data from customers and orders table
               SqlDataAdapter dataAdapter = new SqlDataAdapter("select * from customers; select * from orders", connection);
               DataSet dataSet = new DataSet();
               //Data Table 1 will be customers data which is at Index Position 0
               //Data Table 2 will be orders data which is at Index Position 1
               dataAdapter.Fill(dataSet);
               dataSet.Tables[0].TableName = "Customers";
               dataSet.Tables[1].TableName = "Orders";
               // Fetching First Table Data i.e. Customers Data
               Console.WriteLine("Table 1 Data");
               //Accessing the Data Table from the DataSet using the Custom Table name
               foreach (DataRow row in dataSet.Tables["Customers"].Rows)
               {
                  //Accessing using string column name as keys
                  Console.WriteLine(row["Id"] + ",  " + row["Name"] + ",  " + row["Mobile"]);
```

```
                //Accessing using integer index position as keys
                //Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
            }
            Console.WriteLine();
            // Fetching Second Table Data i.e. Orders Data
            Console.WriteLine("Table 2 Data");
            //Accessing the Data Table from the DataSet using the Custom Table name
            foreach (DataRow row in dataSet.Tables["Orders"].Rows)
            {
                //Accessing using string column name as keys
                //Console.WriteLine(row["Id"] + ",   " + row["CustomerId"] + ",   " + row["Amount"]);
                //Accessing using integer index position as keys
                Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception Occurred: {ex.Message}");
    }
    Console.ReadKey();
  }
}
```

Note: Once we set the custom name for a data table, then we cannot access that table using the default name. we will not get any compilation error, but at runtime, we will get the runtime error.

Important Methods of DataSet in C#:

Let us understand a few important methods of DataSet with examples. We are going to discuss the following three important methods of the DataSet object:

Copy(): Copies both the structure and data of the DataSet. That means it returns a new DataSet with the same structure (table schemas, relations, and constraints) and data as the original DataSet.

Clone(): Copies the structure of the DataSet, including all schemas, relations, and constraints. But does not copy any data. That means it returns a new DataSet with the same schema as the current DataSet but without the data.

Clear(): Clears the DataSet of any data by removing all rows in all tables.

Example to understand Copy, Clone, and Clear Methods of DataSet Object in C#:

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=StudentDB; integrated security=SSPI";
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
          SqlDataAdapter da = new SqlDataAdapter("select * from student", connection);
```

```
            DataSet originalDataSet = new DataSet();
            da.Fill(originalDataSet);
            Console.WriteLine("Original Data Set:");
            foreach (DataRow row in originalDataSet.Tables[0].Rows)
            {
               Console.WriteLine(row["Name"] + ",   " + row["Email"] + ",   " +
row["Mobile"]);
            }
            Console.WriteLine();
            Console.WriteLine("Copy Data Set:");
            //Copies both the structure and data for this System.Data.DataSet.
            DataSet copyDataSet = originalDataSet.Copy();
            if (copyDataSet.Tables != null)
            {
               foreach (DataRow row in copyDataSet.Tables[0].Rows)
               {
                  Console.WriteLine(row["Name"] + ",   " + row["Email"] + ",   " +
row["Mobile"]);
               }
            }
            Console.WriteLine();
            Console.WriteLine("Clone Data Set");
            // Copies the structure of the DataSet, including all DataTable
            // schemas, relations, and constraints. Does not copy any data.
            DataSet cloneDataSet = originalDataSet.Clone();
            if (cloneDataSet.Tables[0].Rows.Count > 0)
            {
               foreach (DataRow row in cloneDataSet.Tables[0].Rows)
               {
                  Console.WriteLine(row["Name"] + ",   " + row["Email"] + ",   " +
row["Mobile"]);
```

```
            }
        }
        else
        {
            Console.WriteLine("Clone Data Set is Empty");
            Console.WriteLine("Adding Data to Clone Data Set Table");
            cloneDataSet.Tables[0].Rows.Add(101, "Test1", "Test1@dotnettutorial.net",
"1234567890");
            cloneDataSet.Tables[0].Rows.Add(101, "Test2", "Test1@dotnettutorial.net",
"1234567890");
            foreach (DataRow row in cloneDataSet.Tables[0].Rows)
            {
                Console.WriteLine(row["Name"] + ",   " + row["Email"] + ",   " +
row["Mobile"]);
            }
        }
        Console.WriteLine();
        //Clears the DataSet of any data by removing all rows in all tables.
        copyDataSet.Clear();
        if(copyDataSet.Tables[0].Rows.Count > 0)
        {
            foreach (DataRow row in copyDataSet.Tables[0].Rows)
            {
                Console.WriteLine(row["Name"] + ",   " + row["Email"] + ",   " +
row["Mobile"]);
            }
        }
        else
        {
            Console.WriteLine("After Clear No Data is Their...");
        }
```

```
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
  }
}
```

Remove a DataTable from ADO.NET DataSet in C#

Now, suppose we have a DataSet object which contains multiple data tables. If we initialize the data set object with null then all the data tables are going to be dropped. That means we cannot access the data tables anymore. But what is our requirement, instead of deleting all the data tables, we want to delete a specific data table and we still want to access other data tables. For this, we need to use the following Remove method.

Remove(DataTable table): This method removes the specified DataTable object from the collection. Here, the parameter table specifies the DataTable to remove.

Note: We need to call the above Remove method on the Tables collection property of the DataSet object. Further to avoid the Runtime exception, first you need to check whether the DataSet contains the data table which you are trying to remove as well as you also need to check if the DataTable can be removed from DataSet.

Example to Remove a DataTable from a Dataset in C#:

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; database=ShoppingCartDB; integrated security=SSPI";
            using (SqlConnection connection = new SqlConnection(ConnectionString))
            {
               //We have written two Select Statements to return data from customers and orders table
               SqlDataAdapter dataAdapter = new SqlDataAdapter("select * from customers; select * from orders", connection);
               DataSet dataSet = new DataSet();
               //Data Table 1 will be customers data which is at Index Position 0
               //Data Table 2 will be orders data which is at Index Position 1
               dataAdapter.Fill(dataSet);
               dataSet.Tables[0].TableName = "Customers";
               dataSet.Tables[1].TableName = "Orders";
               // Fetching First Table Data i.e. Customers Data
               Console.WriteLine("Customers Data");
               //Accessing the Data Table from the DataSet using the Custom Table name
               foreach (DataRow row in dataSet.Tables["Customers"].Rows)
```

```
            {
               //Accessing using string column name as keys
               Console.WriteLine(row["Id"] + ",  " + row["Name"] + ",  " +
row["Mobile"]);
            }
            Console.WriteLine();
            // Fetching Second Table Data i.e. Orders Data
            Console.WriteLine("Orders Data");
            //Accessing the Data Table from the DataSet using the Custom Table name
            foreach (DataRow row in dataSet.Tables["Orders"].Rows)
            {
               //Accessing using integer index position as keys
               Console.WriteLine(row[0] + ",  " + row[1] + ",  " + row[2]);
            }
            Console.WriteLine();
            //Now, we want to delete the Orders data table from the DataSet

if(dataSet.Tables.Contains("Orders")&&dataSet.Tables.CanRemove(dataSet.Tables["Ord
ers"]))    {
               Console.WriteLine("Deleting Orders Data Table..");
               dataSet.Tables.Remove(dataSet.Tables["Orders"]);
               //dataSet.Tables.Remove(dataSet.Tables[1]);
            }
            //Now check whether the DataTable exists or not
            if (dataSet.Tables.Contains("Orders"))
            {
               Console.WriteLine("Orders Data Table Exits");
            }
            else
            {
               Console.WriteLine("Orders Data Table Not Exits Anymore");
```

```
            }
          }
        }
        catch (Exception ex)
        {
          Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
      }
    }
}
```

Manage Provider

This enforces a common interface for accessing data. It simplified data access architecture that often results in improved performance without the loss of functional capabilities.

The .NET Framework by default carries two Managed Providers, SQL Managed Provider and OleDB Managed Provider. The SQL Managed Provider to deal specifically with the Microsoft SQL Server and OleDB Managed Provider to deal with data sources having OleDb connectors. The SQL Server managed provider comes with SqlConnection , SqlCommand , and SqlDataReader. The OLE DB managed provider comes with a similar set of objects, whose names start with OleDb instead of Sql. In both cases, the programming model is essentially the same. The providers abstract the functionality and handle all the heavy lifting under the hood. Both of the providers live in the System.Data namespace.

SQL Server .NET Data Provider	OLE DB .NET Data Provider
SqlCommand	OleDbCommand
SqlConnection	OleDbConnection
SqlDataAdapter	OleDbDataAdapter
SqlDataReader	OleDbDataReader
SqlParameter	OleDbParameter

Typed DataSet

We can specify the Data type when we create a DataColumn for a DataTable. This is to enforce the runtime Type-safety for the column so that only data of specified data type can be stored in the column. In the same way, in most cases, we prefer to make a DataSet itself as Type-safe so as to protect it from runtime mismatch. Hence Typed DataSets generate classes that expose each object in the DataSet in a Type-safe manner. These classes inherit directly from the DataSet class.

1. Using DataSet

```
1. //Create DataAdapter
2. SqlDataAdapter daEmp = new SqlDataAdapter("SELECT empno,empname,empaddress
3.
   FROM EMPLOYEE",conn);
4. //Create a DataSet Object
5. DataSet dsEmp = new DataSet();
6.
7. //Fill the DataSet
8. daEmp.Fill(dsEmp,"EMPLOYEE");
9.
10. //Let us print first row and first column of the table
11. Console.Write(dsEmp.Tables["EMPLOYEE"].Rows[0][0].ToString());
12.
13. //Assign a value to the first column
14. dsEmp.Tables["EMPLOYEE"].Rows[0][0] = "12345";//This will generate runtime error as
    empno column is integer
```

If we observe the above code we will get a runtime error when this code gets executed as the value assigned to the column (empno) does not take string value. Also, any misspelling of the

column will generate a runtime error. And also we need to go through the hierarchy to get the final value.

2. Using Typed DataSet

```
1. //Create DataAdapter
2. SqlDataAdapter   daEmp = new SqlDataAdapter("SELECT empno,empname,empaddress F
   ROM EMPLOYEE",conn);
3. //Create a DataSet Object
4. EmployeeDS dsEmp = new EmployeeDS ();
5. //Fill the DataSet
6. daEmp.Fill(dsEmp,"EMPLOYEE");
7. //Let us print first row and first column of the table
8. Console.Write(dsEmp.EMPLOYEE[0].empno.ToString());
9. //Assign a value to the first column
10. dsEmp.EMPLOYEE[0].empno = "12345";//This will generate compile time error.
```

If we see the above code, a typed dataset is very much similar to a normal dataset. But the only difference is that the schema is already present for the same. Hence any mismatch in the column will generate compile-time errors rather than runtime error as in the case of a normal dataset. Also accessing the column value is much easier than the normal dataset as the column definition will be available in the schema.

Transaction

A Transaction is a set of operations (multiple DML Operations) that ensures either all database operations succeeded or all of them failed to ensure data consistency. This means the job is never half done, either all of it is done or nothing is done.

ADO.NET Transactions Supports:

The ADO.NET supports both Single Database Transactions as well as Distributed Transactions (i.e. Multiple Database Transactions). The single database transaction is implemented using the .NET Managed Providers for Transaction and Connection classes which basically belong to System.Data namespace. On the other hand, the Distributed Transactions are implemented using the TransactionScope object which belongs to System.Transactions namespace

In two ways, we can implement transactions in C# using ADO.NET. They are as follows:

Single Database Transaction using BeginTransaction which belongs to System.Data namespace

Distributed Transactions using TransactionScope which belongs to System.Transactions namespace

Single Database Transaction in C# using BeginTransaction

Single Database Transaction using BeginTransaction

Create a Database and Database Table with Sample Data

Please execute the below SQL Statements in the SQL Server database to create the BankDB and Accounts table as well as populate the Accounts table with the required data.

```
CREATE DATABASE BankDB;

USE BankDB;

CREATE TABLE Accounts

(

    AccountNumber VARCHAR(60) PRIMARY KEY,

    CustomerName VARCHAR(60),

    Balance int
```

);

INSERT INTO Accounts VALUES('Account1', 'James', 1000);

INSERT INTO Accounts VALUES('Account2', 'Smith', 1000);

How to Implement a Single Database Transaction in C# using ADO.NET?

In order to understand how to implement transactions in C# using ADO.NET, please have a look at the below image.

Step 1: First we need to create and open the connection object. And the following two statements exactly do the same thing.

SqlConnection connection = new SqlConnection(ConnectionString)

connection.Open();

Step 2: Then we need to create the SqlTransaction object and to do so, we need to call the BeginTransaction method on the connection object. The following piece of code does the same.

SqlTransaction transaction = connection.BeginTransaction();

Step 3: Then we need to create the command object and while creating the command object, we need to text (in this case of the UPDATE statement) that we want to execute in the database, the connection object where we want to execute the command, and the transaction object which will execute the command as part of the transaction. And then we need to call the ExecuteNonQuery method to execute the DML Statement. The following code exactly does the same thing.

// Associate the first update command with the transaction

SqlCommand cmd = new SqlCommand("UPDATE Accounts SET Balance = Balance – 500 WHERE AccountNumber = 'Account1'", connection, transaction);

cmd.ExecuteNonQuery();

// Associate the second update command with the transaction

cmd = new SqlCommand("UPDATE Accounts SET Balance = Balance + 500 WHERE AccountNumber = 'Account2'", connection, transaction);

cmd.ExecuteNonQuery();

Step 4: If everything goes well then commit the transaction i.e. if both the UPDATE statements are executed successfully, then commit the transaction. To do so call the Commit method on the transaction object as follows.

transaction.Commit();

Step 4: If anything goes wrong then roll back the transaction. To do so call the Rollback method on the transaction object as follows.

transaction.Rollback();

Example to Understand ADO.NET Transactions using C#:

In the below example, we are executing two UPDATE statements by implementing ADO.NET Transactions. The following example code is self-explained, so please go through the comment lines. If both the UPDATE statements are executed successfully, it will commit the transaction and changes are going to be reflected in the database, and if anything goes wrong, then it will

Rollback the transaction and the changes will not reflect in the database, and in this way it will maintain data consistency.

```
using System;
using System.Data.SqlClient;
namespace ADOTransactionsDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Before Transaction");
                GetAccountsData();
                //Doing the Transaction
                MoneyTransfer();
                //Verifying the Data After Transaction
                Console.WriteLine("After Transaction");

                GetAccountsData();
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
    private static void MoneyTransfer()
    {
        //Store the connection string in a variable
        string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; initial catalog=BankDB; integrated security=True";

        //Creating the connection object
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            //Open the connection
            //The connection needs to be open before we begin a transaction
            connection.Open();
            // Create the transaction object by calling the BeginTransaction method on connection object
            SqlTransaction transaction = connection.BeginTransaction();
```

```
try
{
   // Associate the first update command with the transaction
   SqlCommand cmd = new SqlCommand("UPDATE Accounts SET Balance = Balance - 500 WHERE AccountNumber = 'Account1'", connection, transaction);
   //Execute the First Update Command
   cmd.ExecuteNonQuery();
   // Associate the second update command with the transaction
   cmd = new SqlCommand("UPDATE Accounts SET Balance = Balance + 500 WHERE AccountNumber = 'Account2'", connection, transaction);
   //Execute the Second Update Command
   cmd.ExecuteNonQuery();
   // If everythinhg goes well then commit the transaction
   transaction.Commit();
   Console.WriteLine("Transaction Committed");
}
catch(Exception EX)
{
   // If anything goes wrong, then Rollback the transaction
   transaction.Rollback();
   Console.WriteLine("Transaction Rollback");
}
```

```
        }
    }
    private static void GetAccountsData()
    {
        //Store the connection string in a variable
        string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; initial catalog=BankDB; integrated security=True";
        //Create the connection object
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand("Select * from Accounts", connection);
            SqlDataReader sdr = cmd.ExecuteReader();
            while (sdr.Read())
            {
                Console.WriteLine(sdr["AccountNumber"] + ",  " + sdr["CustomerName"] + ",  " + sdr["Balance"]);
            }
        }
    }
  }
}
```

Output: As we can see in the below output the data is in a consistent state i.e. updated in both the Account Number.

Verifying Data Consistency:

Let us modify the example code as follows. In the following code, we deliberately introduce a change that would crash the application at runtime after executing the first update statement. Here, in the second update statement rename the table name as MyAccounts which does not exist in the database. In this case, the first UPDATE statement is executed, then it will try to execute the second UPDATE statement which will throw an exception and in that case, that runtime exception is going to handle by the Catch block and inside the catch block, we are calling the Rollback method which will rollback everything which is executed as part of the transaction.

```
using System;
using System.Data.SqlClient;
namespace ADOTransactionsDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            Console.WriteLine("Before Transaction");
```

```
            GetAccountsData();
            //Doing the Transaction
            MoneyTransfer();
            //Verifying the Data After Transaction
            Console.WriteLine("After Transaction");

            GetAccountsData();
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Exception Occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
    private static void MoneyTransfer()
    {
        //Store the connection string in a variable
        string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; initial catalog=BankDB; integrated security=True";

        //Creating the connection object
```

```
using (SqlConnection connection = new SqlConnection(ConnectionString))
{
    //Open the connection
    //The connection needs to be open before we begin a transaction
    connection.Open();
    // Create the transaction object by calling the BeginTransaction method on connection object
    SqlTransaction transaction = connection.BeginTransaction();
    try
    {
        // Associate the first update command with the transaction
        SqlCommand cmd = new SqlCommand("UPDATE Accounts SET Balance = Balance - 500 WHERE AccountNumber = 'Account1'", connection, transaction);
        //Execute the First Update Command
        cmd.ExecuteNonQuery();
        // Associate the second update command with the transaction
        //MyAccounts table does not exists, so it will throw an exception
        cmd = new SqlCommand("UPDATE MyAccounts SET Balance = Balance + 500 WHERE AccountNumber = 'Account2'", connection, transaction);
        //Execute the Second Update Command
        cmd.ExecuteNonQuery();
        // If everythinhg goes well then commit the transaction
```

```
                transaction.Commit();
                Console.WriteLine("Transaction Committed");
            }
            catch(Exception ex)
            {
                // If anything goes wrong, then Rollback the transaction
                transaction.Rollback();
                Console.WriteLine("Transaction Rollback");
            }
        }
    }
    private static void GetAccountsData()
    {
        //Store the connection string in a variable
        string ConnectionString = @"data source=LAPTOP-ICA2LCQL\SQLEXPRESS; initial catalog=BankDB; integrated security=True";
        //Create the connection object
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand("Select * from Accounts", connection);
```

```
            SqlDataReader sdr = cmd.ExecuteReader();
            while (sdr.Read())
            {
                Console.WriteLine(sdr["AccountNumber"] + ",  " + sdr["CustomerName"] + ",  " +
sdr["Balance"]);
            }
        }
    }
}
```