**Overview of SQL Server Data Provider**

The **SQL Server Data Provider** in ADO.NET is part of the System.Data.SqlClient namespace and includes classes to connect to SQL Server databases, execute commands, and retrieve results. The key classes include:

1. **SqlConnection**– Manages the database connection.
2. **SqlCommand**– Executes SQL commands .
3. **SqlDataReader**– Reads data in a forward-only manner.
4. **SqlDataAdapter**– Manages disconnected data access.

Dataset and dataTable

**SqlConnection**

**Step 1: Import the Namespace**

At the top of your file, include the **System.Data.SqlClient** namespace:

using System.Data.SqlClient;

**Step 2: Define the Connection String**

The connection string contains the details required to connect to the SQL Server, such as server name, database name, user ID, and password.

string connectionString = "Server=YourServerName;Database=YourDatabaseName;User Id=YourUsername;Password=YourPassword;";

**Step 3: Create and Open the Connection**

Use a using statement to ensure that the connection is closed automatically once the block is exited.

using (SqlConnection connection = new SqlConnection(connectionString))

{

try

{

// Open the connection to the database

connection.Open();

Console.WriteLine("Connection opened successfully.");

}

catch (Exception ex)

{

Console.WriteLine("An error occurred: " + ex.Message);

}

}

**SqlCommand**

In ADO.NET with C#, commands are used to execute SQL queries and interact with a database. The main types of commands available in ADO.NET are:

**Text**: Used for SQL commands like SELECT, INSERT, UPDATE, or DELETE.

**StoredProcedure**: Used to execute stored procedures in the database.

**CommentType : CommentType.Text , CommentType.StoredProcedure**

The **SqlCommand** class provides various methods to execute commands and retrieve data, including:

**ExecuteNonQuery():** Executes commands that don’t return rows, such as INSERT, UPDATE, or DELETE.

**ExecuteScalar():** Executes a command that returns a single value. Aggregation data .

**ExecuteReader():** Executes commands that return multiple rows (result sets).

**Example of Each Command Type and Method**

Let's use a sample SQL Server database and SqlCommand to demonstrate these commands.

**1. Text Command with ExecuteNonQuery()**

This example inserts a new row into the database.

using System.Data.SqlClient;

string connectionString = "your\_connection\_string\_here";

using (SqlConnection connection = new SqlConnection(connectionString))

{

string insertQuery = "INSERT INTO Employees (Name, Age, Position) VALUES ('John Doe', 30, 'Developer')";

SqlCommand command = new SqlCommand(insertQuery, connection);

command.CommandType = CommandType.Text;

connection.Open();

int rowsAffected = command.ExecuteNonQuery();

Console.WriteLine($"{rowsAffected} row(s) inserted.");

}

**2. Stored Procedure with ExecuteReader()**

This example executes a stored procedure and retrieves data using ExecuteReader.

using System.Data.SqlClient;

string connectionString = "your\_connection\_string\_here";

using (SqlConnection connection = new SqlConnection(connectionString))

{

SqlCommand command = new SqlCommand("GetEmployees", connection);

command.CommandType = CommandType.StoredProcedure;

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine($"Name: {reader["Name"]}, Age: {reader["Age"]}, Position: {reader["Position"]}");

}

reader.Close();

}

**3. Text Command with ExecuteScalar()**

This example retrieves a single value (e.g., total number of employees).

using System.Data.SqlClient;

string connectionString = "your\_connection\_string\_here";

using (SqlConnection connection = new SqlConnection(connectionString))

{

string countQuery = "SELECT COUNT(\*) FROM Employees";

SqlCommand command = new SqlCommand(countQuery, connection);

command.CommandType = CommandType.Text;

connection.Open();

int employeeCount = (int)command.ExecuteScalar();

Console.WriteLine($"Total Employees: {employeeCount}");

}

**SqlDataReader**

In ADO.NET, the DataReader is an efficient way to retrieve data from a database in a forward-only, read-only manner. It is useful when you need to read data quickly without needing to update it or move backward in the result set. Here’s how you can use a DataReader in C#:

**Steps to Use DataReader in ADO.NET**

**Create a Connection**: Establish a connection to the database.

**Create a Command**: Define an SQL query or stored procedure.

**Execute the Command**: Use the ExecuteReader() method to retrieve the data.

**Read the Data**: Use the Read() method of DataReader to access each row.

using System;

using System.Data.SqlClient;

namespace DataReaderExample

{

class Program

{

static void Main(string[] args)

{

// Connection string to the database

string connectionString = "your\_connection\_string\_here";

// SQL query to execute

string query = "SELECT Id, Name, Age FROM Employees";

// Create a connection object

using (SqlConnection connection = new SqlConnection(connectionString))

{

try

{

// Open the connection

connection.Open();

// Create a command object

SqlCommand command = new SqlCommand(query, connection);

// Execute the command and obtain a DataReader

using (SqlDataReader reader = command.ExecuteReader())

{

// Read each row in the result set

while (reader.Read())

{

// Access data by column index or name

int id = reader.GetInt32(0); // Using column index

string name = reader.GetString(1); // Using column index

int age = reader.GetInt32(reader.GetOrdinal("Age")); // Using column name

// Display the data

Console.WriteLine($"ID: {id}, Name: {name}, Age: {age}");

}

}

}

catch (Exception ex)

{

Console.WriteLine("Error: " + ex.Message);

}

}

}

}

}