ADO.NET (ActiveX Data Objects .NET)

ADO.NET (ActiveX Data Objects for .NET) is a **data access framework** in the .NET platform.

It provides classes that help applications **connect to a database**, **execute commands**, and **manage data** efficiently.

It acts as a **bridge between your C# application and the database** - allowing you to perform **CRUD operations** (Create, Read, Update, Delete).

Purpose of ADO.NET

The main goal of ADO.NET is to enable applications to:

- Communicate with **relational databases** like SQL Server, Oracle, MySQL, PostgreSQL, etc.
- Execute SQL queries and stored procedures.
- Retrieve data efficiently and store it temporarily in memory.
- Update the database with changes made in your C# objects or datasets.

Think of ADO.NET as a **bridge** that connects your C# program to the database and allows data to flow back and forth.

Architecture of ADO.NET

ADO.NET has two main parts:

Component Type	Description	Example Classes
Connected Architecture	Works with an **active connection** to the database	SqlConnection, SqlCommand, SqlDataReader
Disconnected Architecture	Works with data **offline** in memory, no continuous connection	DataSet, DataTable, DataAdapter

Core Objects of ADO.NET

SqlConnection

SqlConnection is a class in the **System.Data.SqlClient / Microsoft.Data.SqlClient** namespace that represents a **connection to a SQL Server database**.

It is the **starting point** for any database operation in ADO.NET - before you can run a query or retrieve data, you must first establish a valid connection to the database using SqlConnection.

- Opens a communication channel between your C# application and the SQL Server database.
- Works together with other ADO.NET objects like SqlCommand, SqlDataReader, and SqlDataAdapter.
- Maintains details like **server name**, **database name**, **authentication**, and **timeout settings** through a **connection string**.

```
using System.Data.SqlClient;
SqlConnection conn = new SqlConnection(
   "Data Source=.;Initial Catalog=StudentDB;Integrated Security=True");
conn.Open();
Console.WriteLine("Connection Opened!");
conn.Close();
```

1. Data Source / Server : Database server name or IP
2. Initial Catalog / Database : Database name
3. Integrated Security : Use Windows authentication (True/False)
4. User ID : SQL Server username
5. Password : SQL Server password
6. Connection Timeout : Timeout in seconds (default: 15)
7. Min Pool Size : Minimum connections in pool (default: 0)
8. Max Pool Size : Maximum connections in pool (default: 100)
9. Pooling : Enable connection pooling (default: true)
10. Encrypt : Encrypt connection (True/False)
11. TrustServerCertificate : Trust server certificate (True/False)

Example: `"Data Source=localhost,1433;Initial Catalog=Harmonious; User ID=sa;Password=Pass@word;Trust Server Certificate=True;Connection Timeout=30; Min Pool Size=5;Max Pool Size=50;"`

SqlCommand

SqlCommand is a class in the **System.Data.SqlClient** namespace used to **execute SQL queries, commands, and stored procedures** against a SQL Server database.

It works **in combination with** SqlConnection, using the open connection to send commands and retrieve results from the database.

```
string query = "INSERT INTO Students (Name, Age) VALUES ('John', 22)";
SqlCommand cmd = new SqlCommand(query, conn);
cmd.ExecuteNonQuery(); // For INSERT, UPDATE, DELETE
```

Command Types:

ExecuteNonQuery() → INSERT / UPDATE / DELETE
ExecuteScalar() → returns a single value
ExecuteReader() → reads multiple rows (used with SqlDataReader)

SqlDataReader (Connected Model)

SqlDataReader is a class in ADO.NET that provides a **fast, efficient, and forward-only** way to read data from a SQL Server database. It works in a **connected model**, meaning the database connection must remain open while reading data.

It is **read-only**, so you can only retrieve data - not modify it. This makes it ideal for quickly fetching large volumes of data in scenarios like displaying query results or generating reports.

```
string query = "SELECT * FROM Students";
SqlCommand cmd = new SqlCommand(query, conn);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    Console.WriteLine($"{reader["Name"]} - {reader["Age"]}");
}
reader.Close();
```

SqlDataAdapter & DataSet (Disconnected Model)

SqlDataAdapter is a class in ADO.NET that acts as a **bridge between the database and a** DataSet, allowing data to be retrieved from or updated to the database without keeping the connection open.

DataSet is an **in-memory data store** that holds **tables, rows, and columns**, enabling applications to work with data in a **disconnected mode**. This allows data manipulation, filtering, and updates offline, which can later be synchronized with the database using the SqlDataAdapter.

```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Students", conn);
DataSet ds = new DataSet();
da.Fill(ds, "Students");

foreach (DataRow row in ds.Tables["Students"].Rows)
{
    Console.WriteLine($"{row["Name"]} - {row["Age"]}");
}
```

SQL Injection (SQLi) - Description

SQL Injection is a type of security vulnerability that occurs when an application **incorporates untrusted user input directly into an SQL query**.

- The attacker supplies specially crafted input that alters the SQL statement the application sends to the database.
- Instead of the application's intended query, the database executes the attacker-controlled SQL.
- This can allow data theft (sensitive rows), data modification or deletion, authentication bypass, or in some cases remote command execution depending on database privileges.

SqlParameter

SqlParameter is a class in ADO.NET used to **pass values safely to SQL queries or stored procedures**.

It is primarily used to **prevent SQL Injection attacks** by separating SQL code from data. Instead of concatenating user input directly into queries, you use SqlParameter objects to supply values dynamically, ensuring that input is treated as **data only**, not executable SQL.

```
string query = "INSERT INTO Students (Name, Age) VALUES (@name, @age)";
SqlCommand cmd = new SqlCommand(query, conn);
cmd.Parameters.AddWithValue("@name", "Alice");
cmd.Parameters.AddWithValue("@age", 21);
cmd.ExecuteNonQuery();
```

Connected vs. Disconnected Architecture

Feature	Connected	Disconnected
Classes	SqlConnection, SqlCommand, SqlDataReader	SqlDataAdapter, DataSet
Connection	Open until data read	Opens once, then works offline
Performance	Faster for small data	Better for bulk / cached data
Use Case	Real-time reads	Offline processing or batch updates

Best Practices

Always close the connection (conn.Close() or using block).
Use parameterized queries to avoid SQL Injection.
Use blocks to auto-dispose connections.
Keep queries optimized - avoid fetching unused columns.
Prefer DataReader for large sequential reads.

Simple Program Example

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

class AdoDemo
{
   static void Main()
   {
      string cs = "Data Source=.;Initial Catalog=StudentDB;Integrated
Security=True";
      using (SqlConnection conn = new SqlConnection(cs))
      {
         conn.Open();

         // Insert
         SqlCommand insertCmd = new SqlCommand(
            "INSERT INTO Students (Name, Age) VALUES (@name, @age)",
conn);
         insertCmd.Parameters.AddWithValue("@name", "Ravi");
         insertCmd.Parameters.AddWithValue("@age", 23);
         insertCmd.ExecuteNonQuery();

         // Select
         SqlCommand selectCmd = new SqlCommand("SELECT * FROM Students",
conn);
         SqlDataReader dr = selectCmd.ExecuteReader();
         while (dr.Read())
         {
```

```
            Console.WriteLine($"{dr["Name"]} - {dr["Age"]}");
        }
    }
  }
}
```

Stored Procedure

A Stored Procedure is a precompiled collection of SQL statements (like SELECT, INSERT, UPDATE, DELETE, or logic blocks) stored inside the SQL Server database.

Precompiled - runs faster than executing raw SQL from your C# code.
Supports **input, output, and return parameters**.

```
CREATE PROCEDURE GetNotificationByID
@NotificationId INT
AS
BEGIN
      SELECT * FROM [notification].[Notification] WHERE NotificationId =
@NotificationId
END
```