ADO.NET

Definition

ADO.NET (ActiveX Data Objects .NET) is a set of classes in the .NET Framework (and .NET Core/5+) that provide a bridge between **.NET applications** and **data sources** like SQL Server, Oracle, or XML files.

It belongs to the **System.Data namespace** and is the standard way to perform **data access** in C#.

Key Features

- **Connected access** → using SqlConnection, SqlCommand, SqlDataReader for real-time, forward-only, read-only access.
- **Disconnected access** → using DataSet, DataTable, DataAdapter so you can fetch data, work offline in memory, then reconnect later.
- **Interoperability** → works with multiple providers (SQL Server, OLE DB, ODBC, Oracle).
- **XML integration** → DataSet can read/write XML easily.
- **Strongly typed data** → through typed DataSets and DataTables.

Common Objects

- **SqlConnection** → connects to a database.
- **SqlCommand** → executes SQL queries or stored procedures.
- **SqlDataReader** → fast, forward-only data reader.
- **SqlDataAdapter** → fills DataSet / DataTable and updates DB.
- **DataSet / DataTable** → in-memory representation of data.

Example

```
using System.Data.SqlClient;

string cs = "Data Source=.;Initial Catalog=EmployeeDB;Integrated Security=True";

using (SqlConnection con = new SqlConnection(cs))
{
   con.Open();
   SqlCommand cmd = new SqlCommand("SELECT COUNT(*) FROM employee", con);
   int count = (int)cmd.ExecuteScalar();
   Console.WriteLine("Number of employees: " + count);
```

```
        }
```

👉 In short:
ADO.NET is the data access technology in .NET that lets you connect to databases, retrieve/manipulate data, and work with it in either connected or disconnected mode.

```
string cs = "Data Source=.;Initial Catalog=EmployeeDB;Integrated Security=True";
```

🔹 **Parts explained**

- **Data Source=.**
  - . means **local SQL Server instance** on your machine.
  - You could also write localhost or LAPTOP-85MTF1DV (your computer/instance name).
- **Initial Catalog=EmployeeDB**
  - The **database name** you want to connect to.
  - In your case, it's EmployeeDB.
- **Integrated Security=True**
  - This means use **Windows Authentication**.
  - Your current Windows account (the one you logged into Windows with) is used to connect.
  - No need to pass username/password explicitly.

🔹 **If you want SQL Server Authentication instead (with username & password)**

Replace with:

```
string cs = "Data Source=.;Initial Catalog=EmployeeDB;User ID=sa;Password=your_password;";
```

User ID=sa;Password=your_password; → provides SQL login credentials.

This is used if you've enabled **SQL Server Authentication** and created a login (e.g., sa).

✅ **Summary**

- Integrated Security=True → **Windows Authentication** (uses your Windows login, no explicit user/pass).

- User ID=...;Password=...; → **SQL Authentication** (you supply credentials).

Connect to a SQL Server by IP.

The key bits are the **server address, port, auth mode, and SSL**. Here are ready-to-use patterns and setup tips.

Connection string examples

SQL Server Authentication (most common for remote)

```
// fixed TCP port
string cs = "Data Source=203.0.113.10,1433;Initial Catalog=EmployeeDB;" +
          "User ID=appuser;Password=Str0ngP@ss!;" +
          "Encrypt=True;TrustServerCertificate=True;Connect Timeout=15;";
```

Named instance (avoid if possible over IP)

```
// requires SQL Browser (UDP 1434) and instance discovery
string cs = "Data Source=203.0.113.10\\SQLEXPRESS;Initial Catalog=EmployeeDB;" +
          "User ID=appuser;Password=Str0ngP@ss!;Encrypt=True;TrustServerCertificate=True;";
```

Better: give the instance a **static port** and use ip,port (first example). It avoids SQL Browser.

Windows (Integrated) Authentication (only if domain/NTLM works across network)

```
string cs = "Data Source=203.0.113.10,1433;Initial Catalog=EmployeeDB;" +
          "Integrated Security=True;Encrypt=True;TrustServerCertificate=True;";
```

With extra conveniences

```
string cs = "Data Source=203.0.113.10,1433;Initial Catalog=EmployeeDB;" +
          "User ID=appuser;Password=xxxx;Encrypt=True;TrustServerCertificate=True;" +
          "MultipleActiveResultSets=True;Application Name=MyWinFormsApp;";
```

Server & network checklist (do these on the remote box)

1. **Enable TCP/IP**
   SQL Server Configuration Manager → SQL Server Network Configuration → Protocols → enable **TCP/IP**.

2. **Set a static TCP port**
   Protocols → TCP/IP → Properties → IPAll → set **TCP Port = 1433** (or any open port). Restart the SQL Server service.

3. **Open firewall**
   On the server, allow inbound TCP **1433** (or your chosen port).
   If you must use a named instance, also allow UDP **1434** for SQL Browser.

4. **Authentication**
   Use **SQL logins** (create a login + user, grant rights), or ensure Windows auth works over the network/domain.

5. **Certificates / encryption**
   .NET 4.7+ defaults to Encrypt=True when possible. If you don’t have a trusted cert, set Encrypt=True;TrustServerCertificate=True (OK for labs; in production, install a proper TLS cert and use TrustServerCertificate=False).

Quick test before coding

- From your machine, open **SSMS**: Server name: 203.0.113.10,1433 → log in with the same credentials.
- Or use sqlcmd:
- sqlcmd -S 203.0.113.10,1433 -U appuser -P Str0ngP@ss!

Minimal C# test

```
using System;

using System.Data.SqlClient;


class TestConn {

  static void Main() {

    string cs = "Data Source=203.0.113.10,1433;Initial Catalog=EmployeeDB;" +

          "User ID=appuser;Password=Str0ngP@ss!;Encrypt=True;TrustServerCertificate=True;";

    using (var con = new SqlConnection(cs)) {

      con.Open();

      using (var cmd = new SqlCommand("SELECT @@SERVERNAME, DB_NAME()", con))

      using (var r = cmd.ExecuteReader()) {

        while (r.Read()) Console.WriteLine($"{r.GetString(0)} | {r.GetString(1)}");

      }

    }

  }

}
```

If it fails, the error message usually points to: unreachable port/firewall, wrong auth, or encryption mismatch. Share the exact error text and I’ll pinpoint the fix.