C# Database Workshop — ADO.NET, EF Core

This hands-on workshop guides you through connecting C# applications to SQL Server using both ADO.NET and Entity Framework Core. You'll build a console app and a minimal API that work with both a code-first and a database-first (Northwind) approach.

Part 1 — Console App (ADO.NET + EF Core)

Let's connect directly to a database and use EF Core with a code-first approach!

Step 1 – Create Project

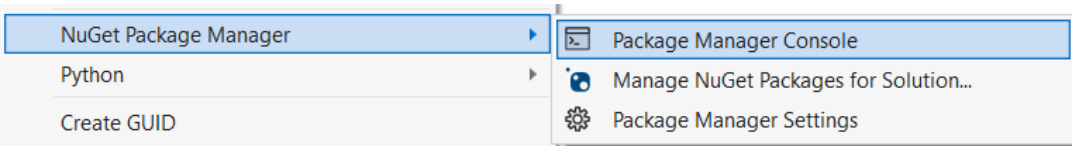
Create a new Console project in Visual Studio 2022 (.NET 8) called **DbWorkshopConsole**.
In VS 2022 → *File* → *New* → *Project*

Choose **Console App (.NET 6 or .NET 8)** → click **Next** → finish setup

Step 2 – Add NuGet Packages

Install Entity Framework NuGet packages

Install the following NuGet packages using the Package Manager Console:



Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools

Note: Make sure you are in the folder where your C# project is when executing the above commands

(You might have to install NET Framework for older projects in older apps but we use Use **EF Core** here)

Step 3 — Create your model classes

Let's say you have a Student entity:

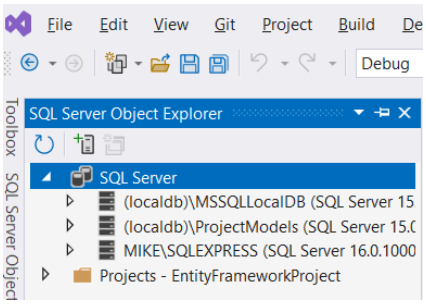
```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Step 4 — Create a DbContext

This class defines the connection and tables.

```
using Microsoft.EntityFrameworkCore;
public class SchoolContext : DbContext
{
    public DbSet<Student> Students { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder options)
    {
        options.UseSqlServer(
            "Server=(localdb)\\MSSQLLocalDB;Database=SchoolDB;Trusted_Connection=True;");
    }
}
```

LocalDB comes with Visual Studio. You can check it in **SQL Server Object Explorer**.



LocalDB is a lightweight, developer-only edition of SQL Server that is automatically installed with Visual Studio. It's designed for **local development**, not production but it's still SQL Server Express under the hood

Step 5 — Create the database (migrations)

In **Package Manager Console**, run:

Add-Migration InitialCreate

Update-Database

- **Add-Migration** creates a migration file describing your tables.
- **Update-Database** actually creates the database.

You'll now see **SchoolDB** under **(localdb)\MSSQLLocalDB**.
Note: You may need to Refresh the databases node to see the new **SchoolDB** database

Step 6 — Use EF in code

In your **Program.cs Main( )** method type:

using var db = new SchoolContext();

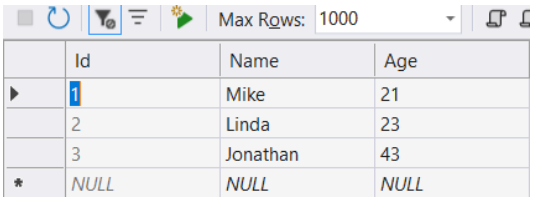
```
db.Students.Add(new Student { Name = "Mike", Age = 21 });
db.SaveChanges();

// Read
var list = db.Students.ToList();
foreach (var s in list)
    Console.WriteLine($"{s.Id}: {s.Name} ({s.Age})");
```

(feel free to add a few more rows to the table)

Step 7 — View your data In Visual Studio:

- Open **View → SQL Server Object Explorer**
- Expand (localdb)\MSSQLLocalDB → Databases → SchoolDB
- Right-click Students → View Data

You'll see your table populated.



	Id	Name	Age
▶	1	Mike	21
	2	Linda	23
	3	Jonathan	43
*	NULL	NULL	NULL

Note: To remove a student try `db.Students.Remove(new Student { Id = 3 });`
`To remove the third student`

Step 8 — Common variations - Using scaffolding (reverse-engineer from existing DB)

If you already have a database type the following in the NuGet Console (all on one line).

For example, I have a database called **Northwind** on **SQLExpress** server. So I type

> **Scaffold-DbContext "Server=.\SQLExpress;Database=Northwind;Trusted_Connection=True;TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models**

(All on one line)
This will create a folder in your project called **Models.** Please make sure this folder exists and there are no error reported.

Using configuration file (optional)

You can also store connection strings in appsettings.json (for web apps) or app.config (for older console apps).

Just to recap, so far, you have: 👍🏻

1 Created a new C# project

2 Installed EF Core NuGet packages

3 Defined your model classes

4 Created a DbContext

5 Ran Add-Migration + Update-Database

6 Used EF methods (Add, Find, ToList, SaveChanges)

7 Viewed data in SQL Server Object Explorer

Now, let's connect to and use a database in **SQL Express** directly from **Visual Studio 2022** C# app.

Find or create your database

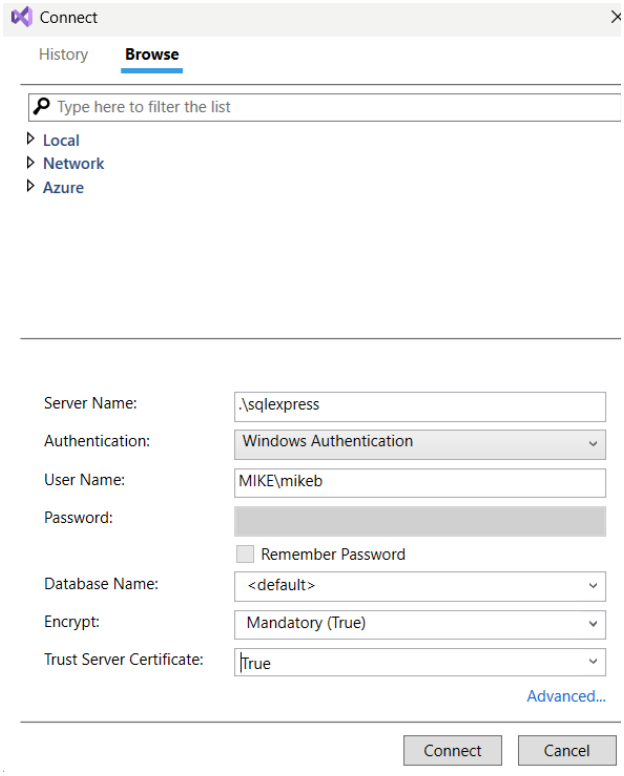
Option A — Using SQL Server Management Studio (SSMS)

1. Connect to **Server name:** .\SQLEXPRESS
   (or localhost\SQLEXPRESS)

Option B — Using Visual Studio (recommended for fewer screen swappint)

1. Open **View → SQL Server Object Explorer**
2. Click the "+ Add SQL Server" icon
3. Choose **Database Engine**
4. **Server name:** .\SQLEXPRESS
   (or localhost\SQLEXPRESS)

5. **Authentication:** Windows Authentication
6. Trust server Certificate = True
7. No need to change anything else, just click **Connect**



Step 3 — Connection string for C#

The connection string usually looks like this:

```
"Server=.\\SQLEXPRESS;Database=Northwind;Trusted_Connection=True;TrustServerCertificate=True"
```

or if using localdb (from VS default templates):

Server=(localdb)\MSSQLLocalDB;Database=TestDB;Trusted_Connection=True;

Step 4 — Access it with ADO.NET (raw)

If this code outputs “Connected!”, your connection works!

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

class Program
{
    static void Main()
    {
        string connString = "Server=.\\SQLEXPRESS;Database=Northwind;Trusted_Connection=True;
;TrustServerCertificate=True";

        using (SqlConnection conn = new SqlConnection(connString))
        {
            conn.Open();
            Console.WriteLine("Connected!");

            string sql = "SELECT COUNT(*) FROM customers";
            using (SqlCommand cmd = new SqlCommand(sql, conn))
            {
                int count = (int)cmd.ExecuteScalar();
                Console.WriteLine($"Rows: {count}");
            }
        }
    }
}
```

To read data type:

```
static void readDataExample()
{
    string connString =
"Server=.\\SQLEXPRESS;Database=Northwind;Trusted_Connection=True;;TrustServerCertificate=True";

    using (SqlConnection conn = new SqlConnection(connString))
    {
        conn.Open();
        Console.WriteLine("Connected!");

        string sql = "SELECT * FROM customers";
        using (SqlCommand cmd = new SqlCommand(sql, conn))
        {
            SqlDataReader reader =  cmd.ExecuteReader();
            while (reader.Read())
            {
                string city = reader["City"].ToString();
                Console.WriteLine($"{reader[0]} {reader[1]} {city}");
            }
        }
    }
}
```

Note: You can use Insert or delete using code like:

```
static void part4()
{
    string connString =
"Server=.\\SQLEXPRESS;Database=Northwind;Trusted_Connection=True;;TrustServerCertificate=True";

    using (SqlConnection conn = new SqlConnection(connString))
    {
        conn.Open();
```

```
        Console.WriteLine("Connected!");

        string sql = "DELETE FROM customers WHERE CustomerID='zzzzz'";
        using (SqlCommand cmd = new SqlCommand(sql, conn))
        {
            int rowsAffected = cmd.ExecuteNonQuery();
            if (rowsAffected > 0)
            {
                Console.WriteLine($"Number of row affected: {rowsAffected}");
            }
        }
    }
}
```

Step 5 — Access it with Entity Framework Core

Create another class called **MyDbContext**

```
public class MyDbContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlServer(@"Server=.\SQLEXPRESS;Database=TestDB;Trusted_Connection=True;");
}
```

Then run the following in NuGet console to create the tables

Add-Migration Initial

Update-Database

Step 6 — Check data visually

In **Visual Studio → SQL Server Object Explorer**:

- Expand .\SQLEXPRESS → Databases → TestDB → Tables
- Right-click a table → *View Data*

Just to review what you've achieved: 👍🏻

Task	How
Start service	Services.msc → SQL Server (SQLEXPRESS)
Create DB	SSMS or Visual Studio SQL Server Object Explorer
Connect from code	Server=.\SQLEXPRESS;Database=YourDB;Trusted_Connection=True;
Use in ADO.NET	SqlConnection, SqlCommand
Use in EF Core	options.UseSqlServer(connectionString)

Step-by-step to connect EF to your existing SQL Express database.

(We'll use **Entity Framework Core**, since you're on Visual Studio 2022.)

Step 1 — Install EF Core packages

In **Visual Studio → Tools → NuGet Package Manager → Package Manager Console**, run:

Install-Package Microsoft.EntityFrameworkCore

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

These let EF talk to SQL Server and create migrations.

Step 2 — Create your model classes

Let's say your database has a table Students with columns Id, Name, and Age.

Create a class to match it:

```
public class Student
{
    public int Id { get; set; }  // must match the table's primary key
    public string Name { get; set; }
    public int Age { get; set; }
}
```

🧩 Step 3 — Create a DbContext

This is the bridge between your code and the SQL Express database.

```
using Microsoft.EntityFrameworkCore;

public class MyDbContext : DbContext
{
    public DbSet<Student> Students { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder options)
    {
        options.UseSqlServer(

"Server=.\\SQLEXPRESS;Database=TestDB;Trusted_Connection=True;TrustServerCertificate=True;");
    }
}
```

Notice the connection string includes `TrustServerCertificate=True`.

You can research this attribute at another time.

Step 4 — Run EF commands (optional)

If you want EF to **create** or **update** the database schema (code-first):

Add-Migration InitialCreate

Update-Database

But if your database already exists and you just want to use it (read/write), you can skip this step or use **Database-First Scaffolding** (see Step 6).

Step 5 — Use EF in code

Example: read and write to your Students table.

```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        using var db = new MyDbContext();

        // Add a new record
        db.Students.Add(new Student { Name = "Mike", Age = 21 });
        db.SaveChanges();

        // Read all
        var list = db.Students.ToList();
        foreach (var s in list)
            Console.WriteLine($"{s.Id}: {s.Name} ({s.Age})");
    }
}
```

Run that in your console app — it will insert and list rows.

Step 6 —Reverse engineer from an existing database

If you already have tables in SQL Express and don't want to re-create them, use **scaffolding**.

In NuGet Package Manager Console:

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

Add your EF classes

In Solution Explorer, right-click your project → **Add** → **Class...** → name it **NorthwindContext.cs**

```
using Microsoft.EntityFrameworkCore;

public class NorthwindContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder options)
        =>
options.UseSqlServer(@"Server=.\SQLEXPRESS;Database=Northwind;Trusted_Connection=True;Trus
tServerCertificate=True;");
}
```

Create a new class that corresponds to column names in your database table
Here, since we want to read from the customer table we type:

```
public class Customer
{
    public string CustomerID { get; set; } = string.Empty;
    public string CompanyName { get; set; } = string.Empty;
    public string ContactName { get; set; } = string.Empty;
    public string? City { get; set; }
}
```

Note: Not all fields are considered

Read data

```
using System;
using System.Linq;

internal class Program
{
    static void Main(string[] args)
    {
        using var db = new NorthwindContext();

        var customers = db.Customers.ToList();

        foreach (var c in customers)
            Console.WriteLine($"{c.CustomerID,-5} {c.CompanyName,-40} {c.City}");
    }
}
```

Read data using LINQ

```
using System;
using System.Linq;

internal class Program
{
    static void Main(string[] args)
    {
        using var db = new NorthwindContext();

        var customers = db.Customers
                          .OrderBy(c => c.CompanyName)
                          .Take(10)
                          .ToList();

        foreach (var c in customers)
            Console.WriteLine($"{c.CustomerID,-5} {c.CompanyName,-40} {c.City}");
    }
}
```

Please investigate other methods of Linq such as **Where** and **Select**

***** End *****