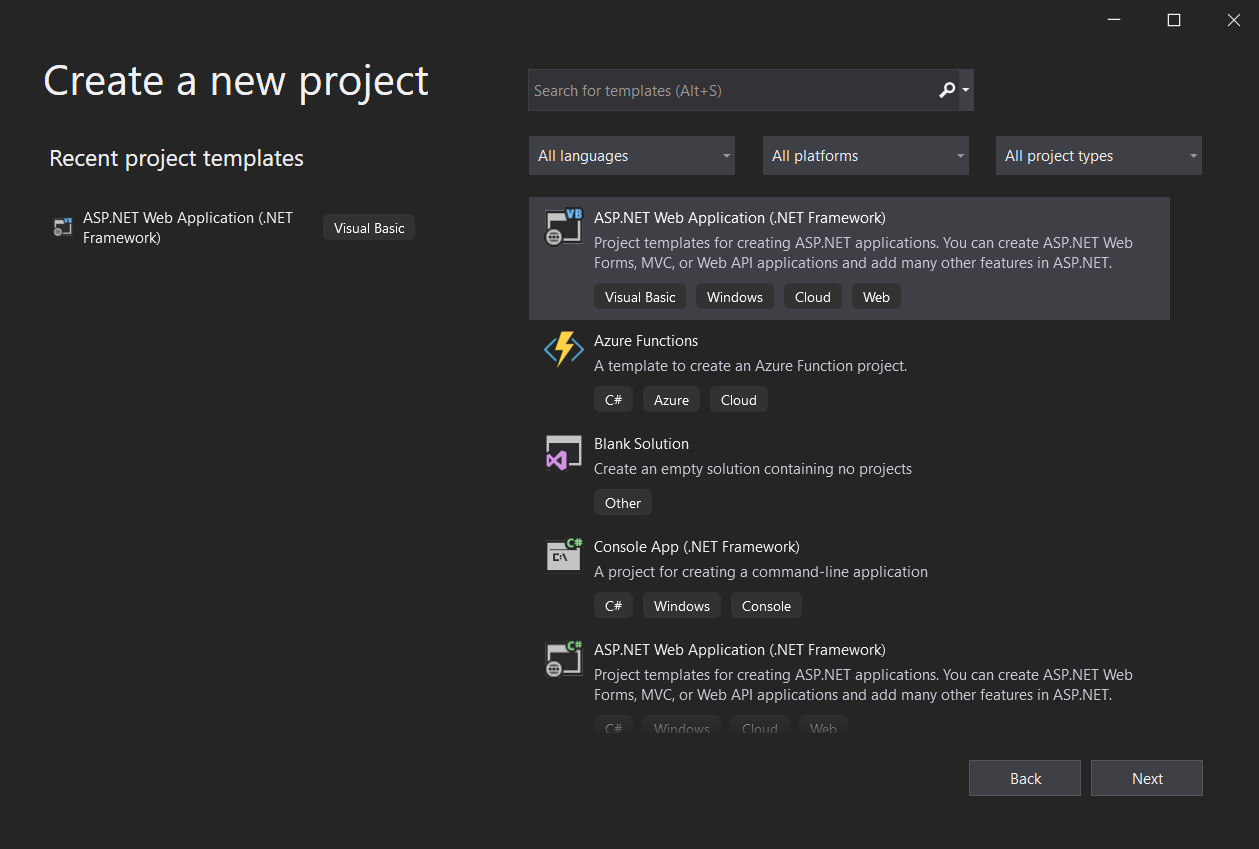
Step 1. Open Visual Studio and Create a New Project:

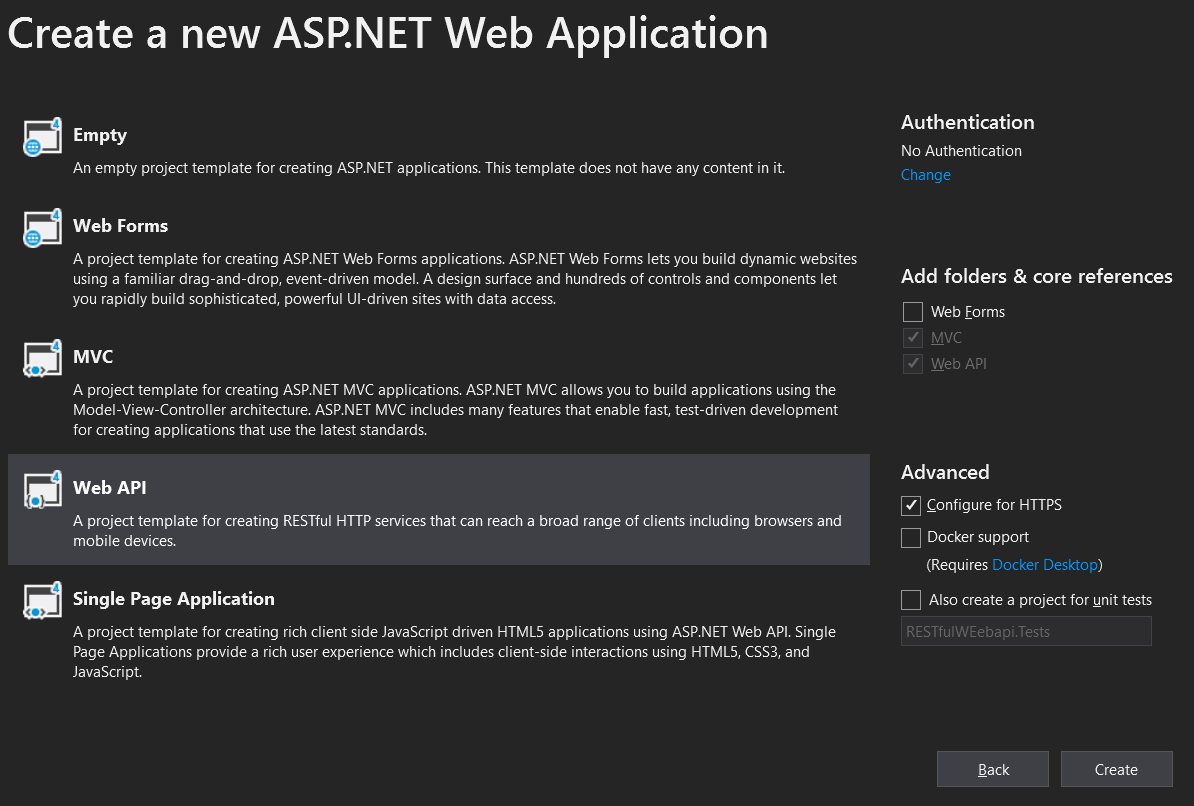
.

1. In the "Create a new project" dialog, select "ASP.NET Core Web Application" from the list.
2. Choose a name and location for your project.
3. Select the appropriate framework (e.g., ASP.NET Core 3.1 or later).
4. Choose the "Web Application (Model-View-Controller)" template.
5. Click "Create."



2. Configure the Project:

1. In the "Create a new ASP.NET Core web application" dialog:
2. Select "Configure for HTTPS" if you want to enable secure connections.
3. Choose the authentication type (Individual User Accounts, Windows Authentication, No Authentication).
4. Click "Create."



Step 3. Open Server Explorer:

1.In Visual Studio, go to the "View" menu.

Select "Server Explorer" from the dropdown.

2. Connect to a Database:

In the Server Explorer, right-click on "Data Connections."

Select "Add Connection."

3. Choose Data Source:

In the "Add Connection" dialog, choose the appropriate data source. For a SQL Server database, select "Microsoft SQL Server."

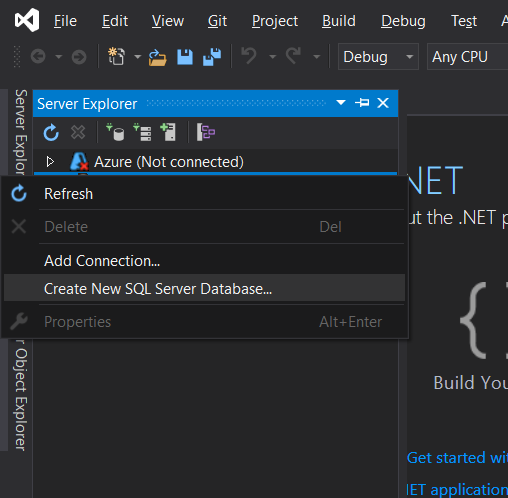
Enter the Server name and choose the authentication method (Windows Authentication or SQL Server Authentication).

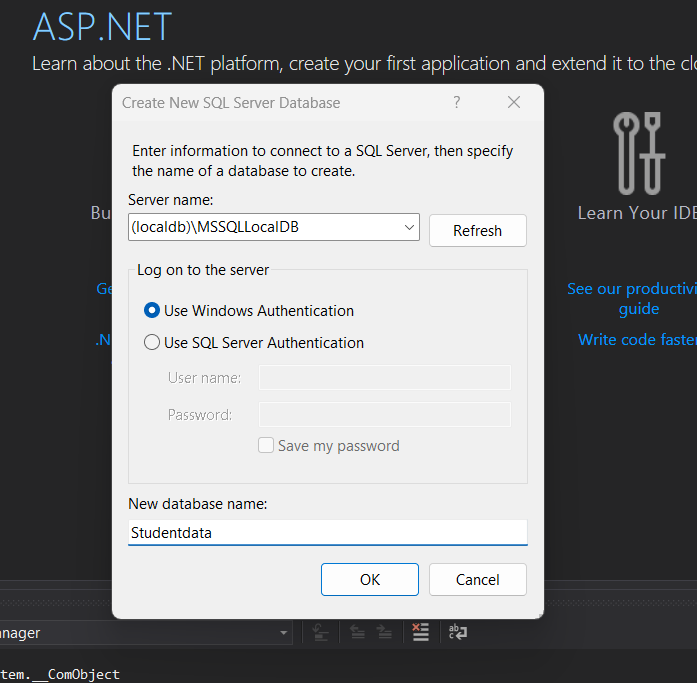
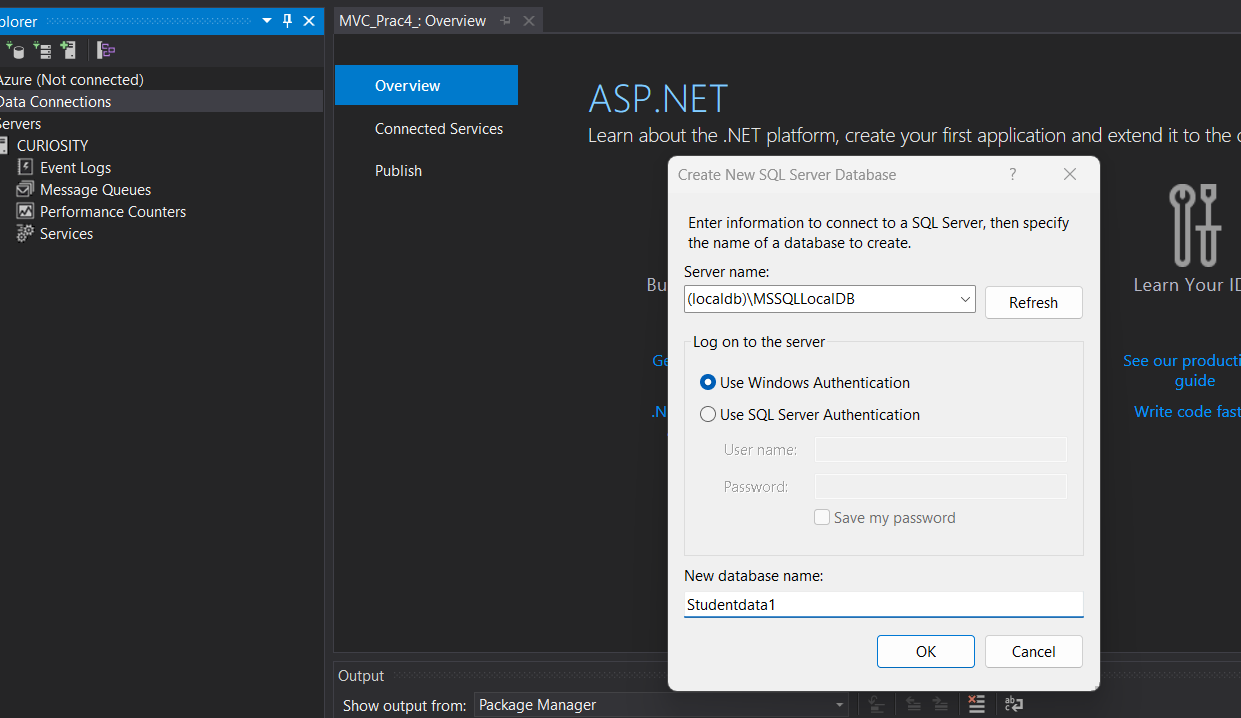
Click "Connect."

4. Select or Create a Database:

Once connected, you will see a list of databases on the server.

You can either select an existing database or create a new one.





Step 5: Creating and designing the table

Step 1: Create a New Table

In the "SQL Server Object Explorer," expand your connected database node.

Right-click on "Tables" and choose "Add New Table."

Step 4: Design the Table

In the "Table Designer," you'll see columns such as "Column Name," "Data Type," etc.

Add a new column:

Click on the first empty row in "Column Name" and enter "ID."

Choose "int" as the data type.

Check the "PK" (Primary Key) box.

Add another column:

Click on the next empty row in "Column Name" and enter "Name."

Choose "nvarchar" as the data type.

Set the "Length" to an appropriate value.

Step 5: Input Data

Input the data into the table directly:

For ID, you can either manually input the values or allow the system to auto-increment.

For Name, input the values "OM," "X," "Y," "Z," and leave NULL for the last row.

Step 2: Save the Table

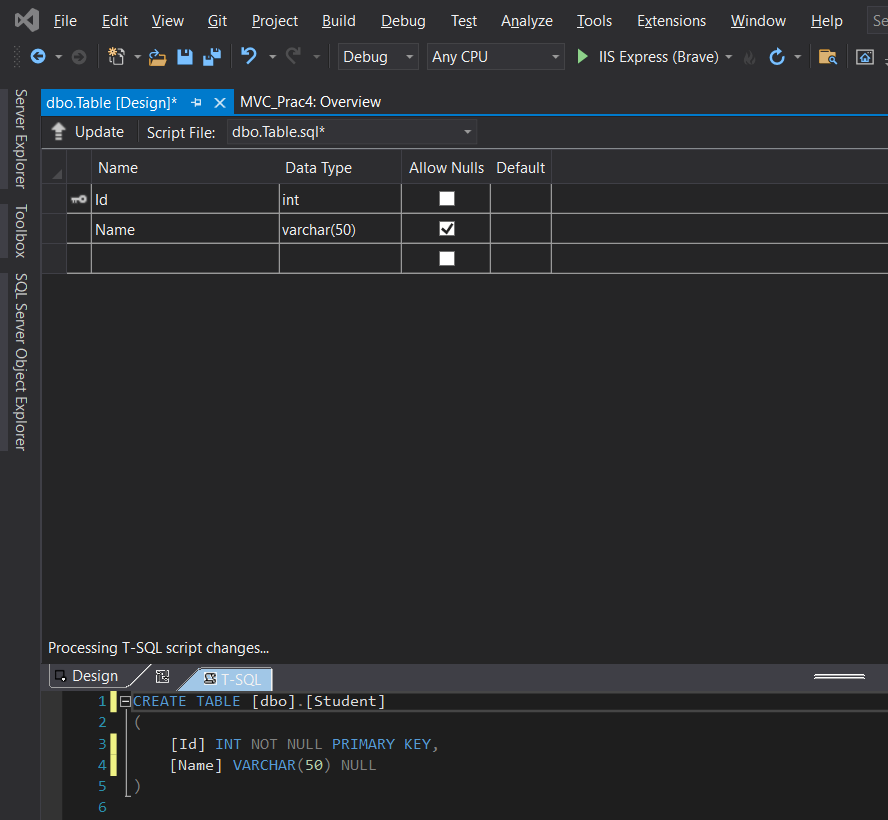
Click the "Update" button in the "Table Designer" toolbar (or press Ctrl + S).

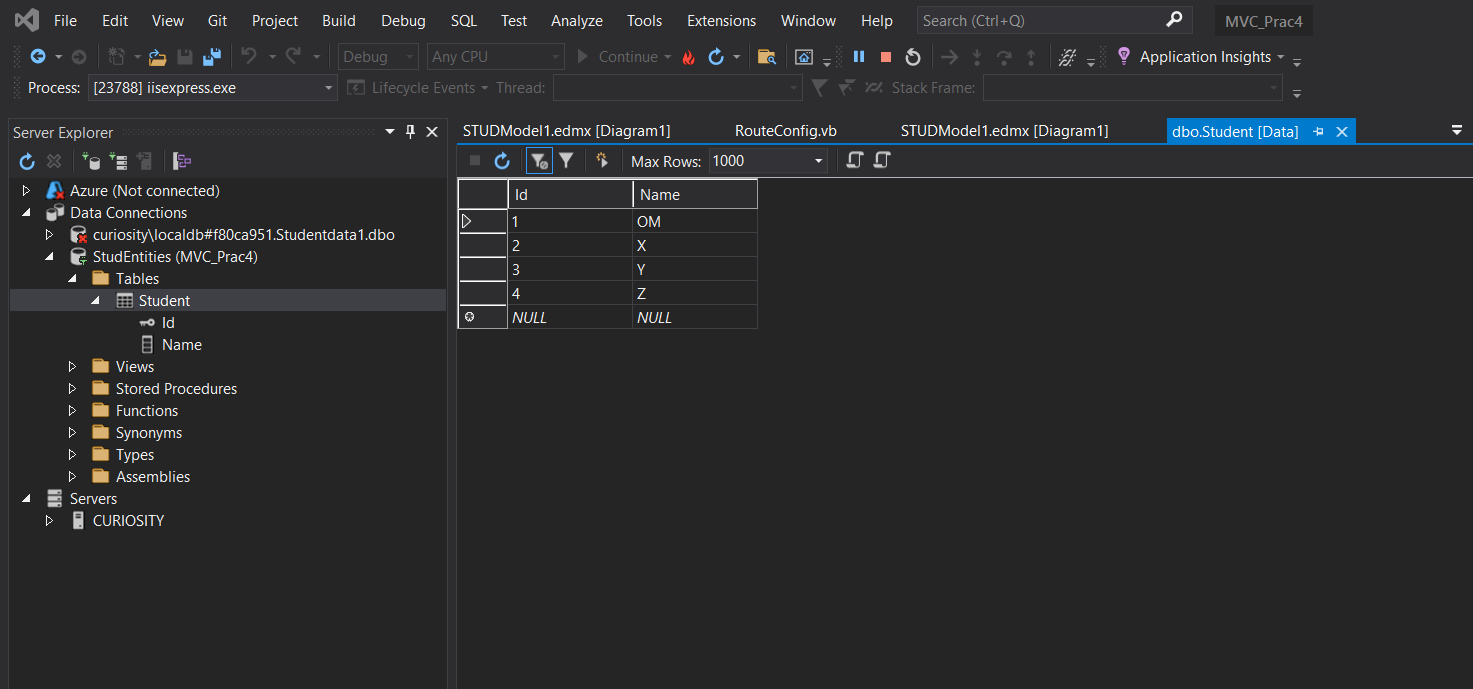
Provide a name for the table and click "OK."

Step 3: Check the Table

In the "SQL Server Object Explorer," refresh the "Tables" node.

You should see your newly created table with data.





Step 6: Adding New ADO.NET Entity Data Model

Step 1: Create a New ADO.NET Entity Data Model

Right-click on your project in Solution Explorer.

Select Add > New Item...

Choose "Data" from the left pane, and then choose "ADO.NET Entity Data Model."

Enter a name for your model (e.g., SchoolModel) and click "Add."

Step 2: Choose Data Connection

Choose "EF Designer from Database" and click "Next."

Select the data connection you previously established and click "Next."

Step 3: Choose Database Objects

Choose the tables or views you want to include in the model (e.g., the table you created with "ID" and "Name").

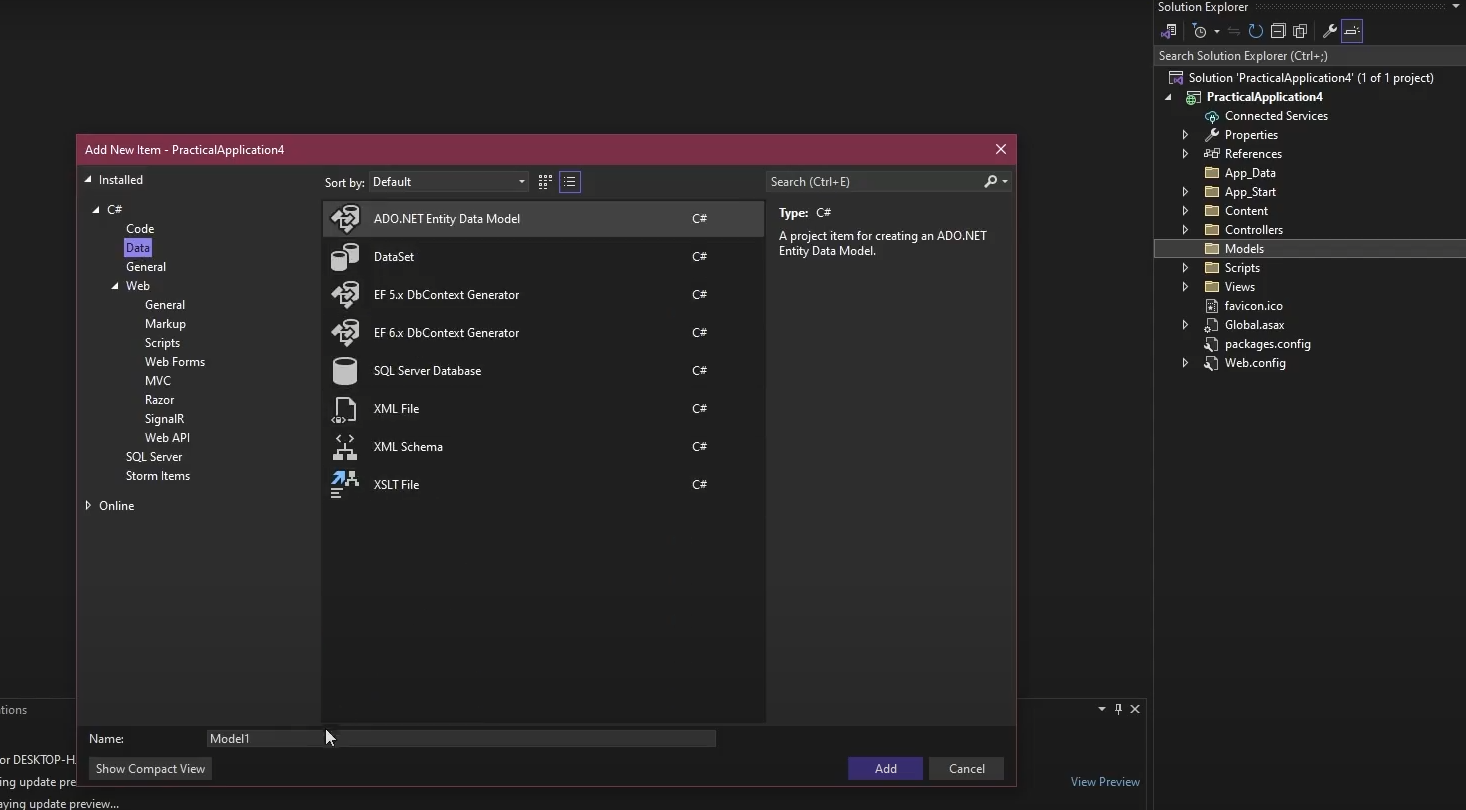
Click "Finish."

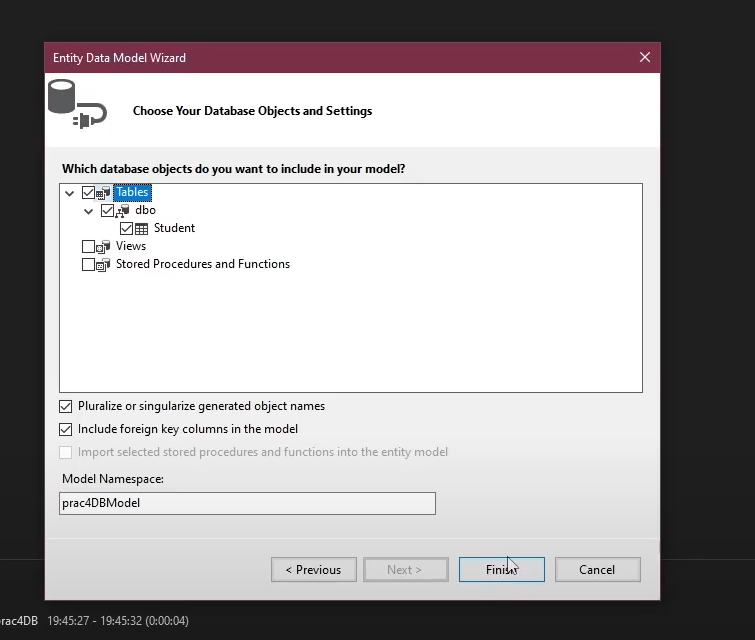
Step 4: Save and Generate Model

Save the model by clicking "Ctrl + S" or selecting File > Save All.

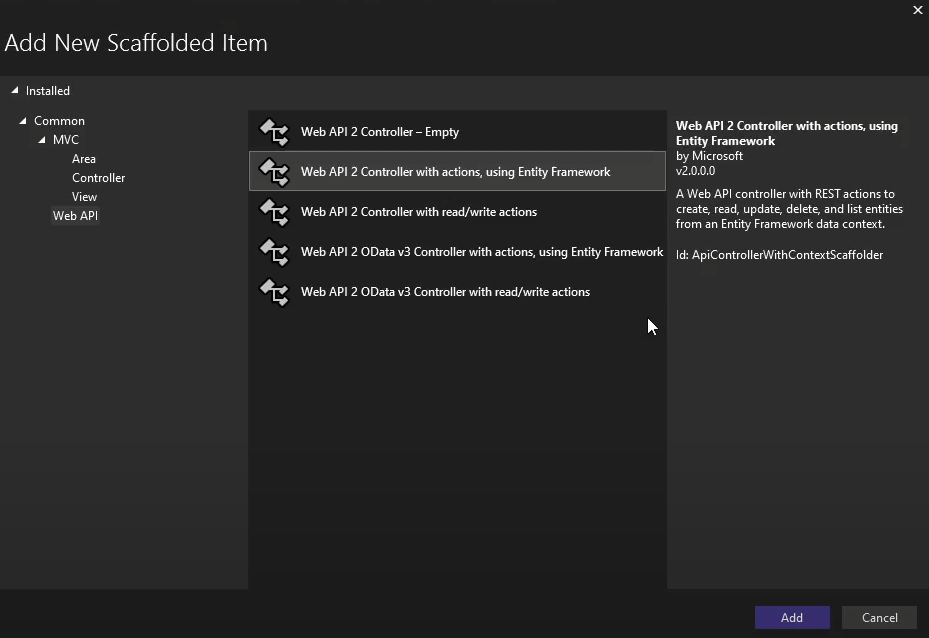
The Entity Data Model will be generated based on the selected database objects.

Now, you have an ADO.NET Entity Data Model in your project. You can use this model to perform CRUD operations on your database through Entity Framework.





Adding Scaffolding item,



﻿using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using System.Web.Http.Description;

using PracticalApplication6.Models;

namespace PracticalApplication6.Controllers

{

public class StudentController : ApiController

{

private prac4DBEntities db = new prac4DBEntities();

// GET: api/Student

public IQueryable<Student> GetStudents()

{

return db.Students;

}

// GET: api/Student/5

[ResponseType(typeof(Student))]

public IHttpActionResult GetStudent(int id)

{

Student student = db.Students.Find(id);

if (student == null)

{

return NotFound();

}

return Ok(student);

}

// PUT: api/Student/5

[ResponseType(typeof(void))]

public IHttpActionResult PutStudent(int id, Student student)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != student.Id)

{

return BadRequest();

}

db.Entry(student).State = EntityState.Modified;

try

{

db.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!StudentExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return StatusCode(HttpStatusCode.NoContent);

}

// POST: api/Student

[ResponseType(typeof(Student))]

public IHttpActionResult PostStudent(Student student)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

db.Students.Add(student);

try

{

db.SaveChanges();

}

catch (DbUpdateException)

{

if (StudentExists(student.Id))

{

return Conflict();

}

else

{

throw;

}

}

return CreatedAtRoute("DefaultApi", new { id = student.Id }, student);

}

// DELETE: api/Student/5

[ResponseType(typeof(Student))]

public IHttpActionResult DeleteStudent(int id)

{

Student student = db.Students.Find(id);

if (student == null)

{

return NotFound();

}

db.Students.Remove(student);

db.SaveChanges();

return Ok(student);

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

private bool StudentExists(int id)

{

return db.Students.Count(e => e.Id == id) > 0;

}

}

}

