Module 11: Introduction to Databases and Entity Framework

In this module, you will learn about databases and how to use Entity Framework to create databases for your C# projects. To continue with the rest of the chapter, please create a new C# project in Visual Studio.

1) Databases

So far, we have used very limited amount of data and we used lists and arrays to store and manipulate the data. With this approach, we could store little data in the memory and when we close the application all data created in runtime were automatically destroyed. However, many applications require storing large amounts of data and keeping the data for longer periods of time (until they are intentionally deleted). In other words, in many applications, we need the data to *persist*. In such cases, we use databases to store and manipulate data in a structured way. For example, in a course management program, you may need to keep record of hundreds of courses and students in a university and allow users to list, delete, update, and search the data. Surely, a database needs to be created for such a program.

We use database management systems to design and implement databases. Once the database is created, you can include it in your C# project. We will install SQL Server Data Tools to Visual Studio, which will provide as with an embedded database management option inside Visual Studio.

Open Visual Studio Installer from the Start menu.

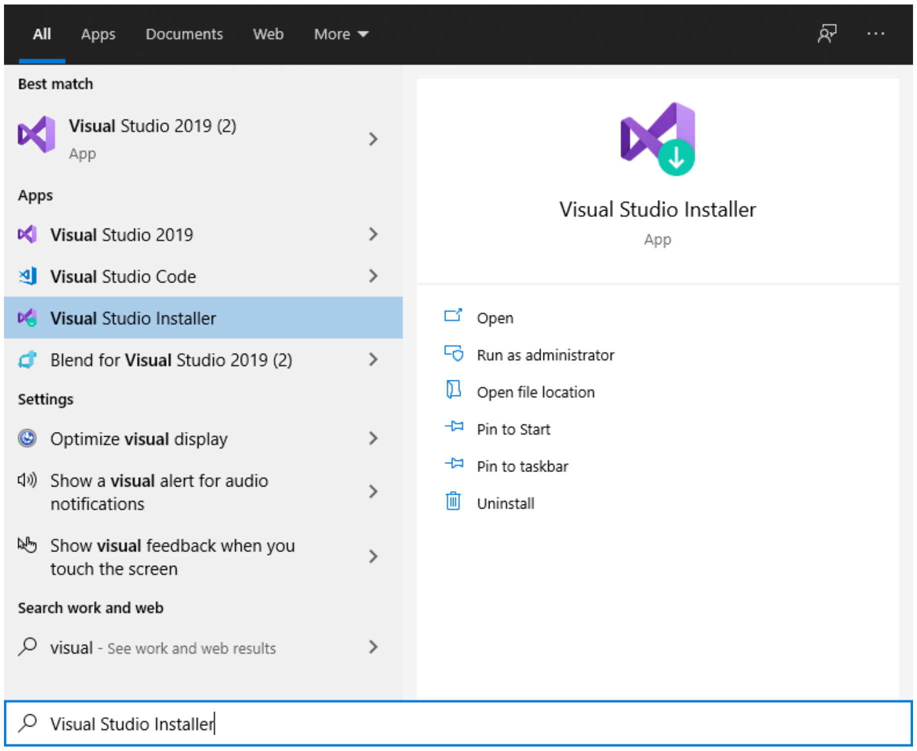
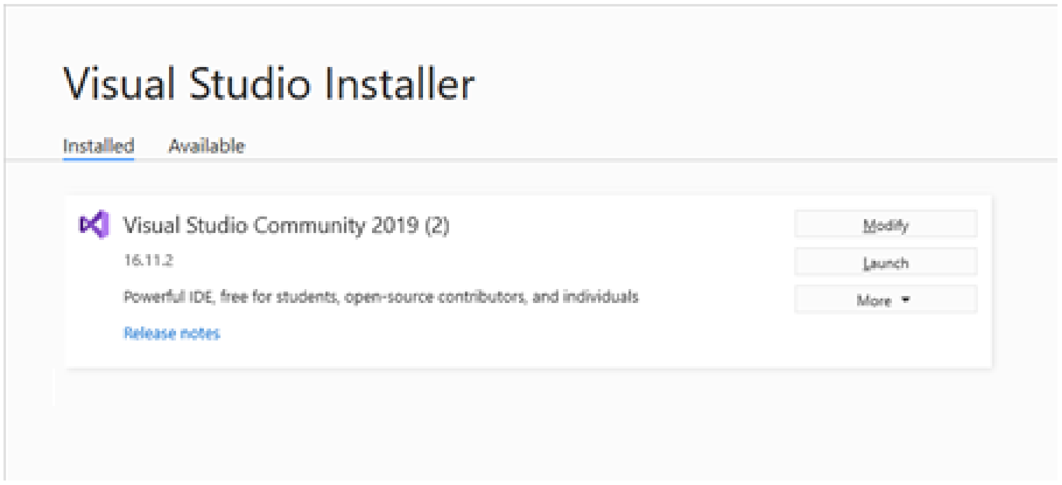


Figure 1. Opening Visual Studio Installer.

In this window as shown below, please click on **Modify**.



Visual Studio Installer

Installed Available

Visual Studio Community 2019 (2)

16.11.2

Powerful IDE, free for students, open-source contributors, and individuals

Release notes

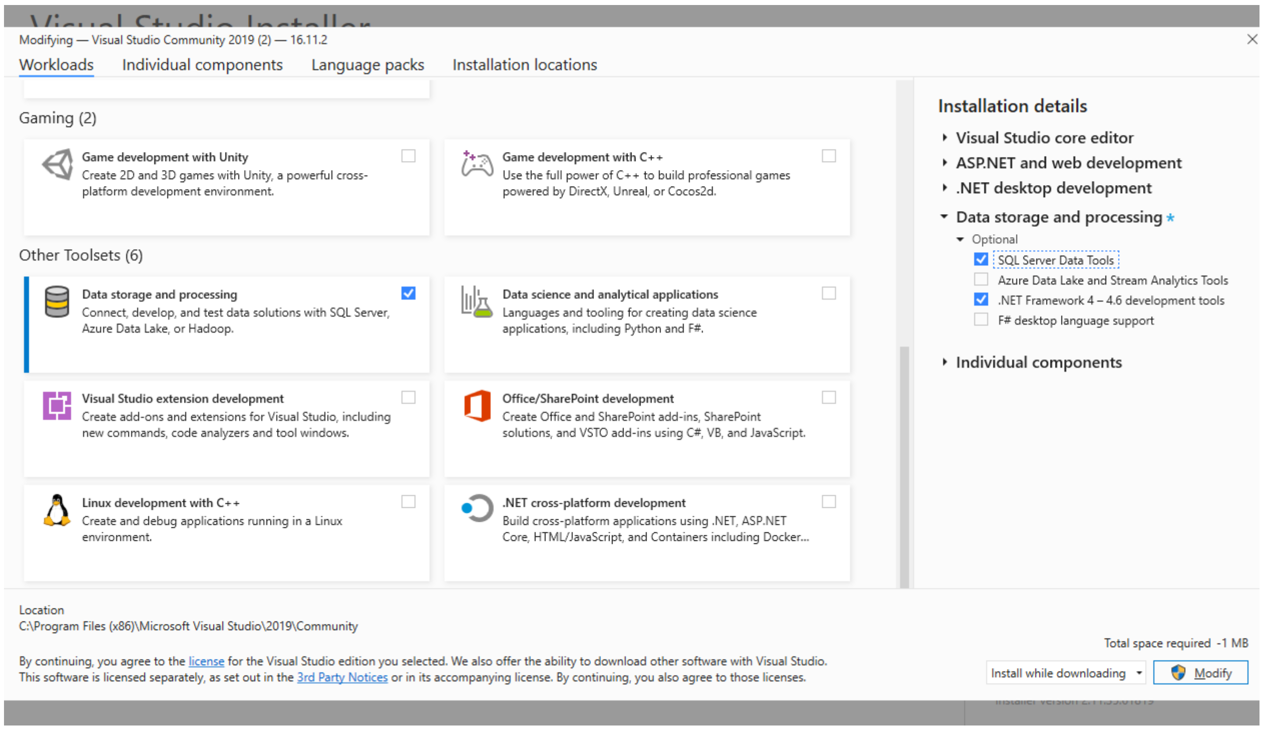
Modify

Launch

More ▾

Figure 2. Visual Studio Installer window.

In the next window, check the **Data storage and processing** item. On the right hand-side, some related options will be shown. Mark those that are shown with red arrow in the image below. Then, click on the **Modify** button.



Modifying — Visual Studio Community 2019 (2) — 16.11.2

Workloads Individual components Language packs Installation locations

Gaming (2)

Game development with Unity
Create 2D and 3D games with Unity, a powerful cross-platform development environment.

Game development with C++
Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Other Toolsets (6)

[✓] Data storage and processing
Connect, develop, and test data solutions with SQL Server, Azure Data Lake, or Hadoop.

Data science and analytical applications
Languages and tooling for creating data science applications, including Python and F#.

Visual Studio extension development
Create add-ons and extensions for Visual Studio, including new commands, code analyzers and tool windows.

Office/SharePoint development
Create Office and SharePoint add-ins, SharePoint solutions, and VSTO add-ins using C#, VB, and JavaScript.

Linux development with C++
Create and debug applications running in a Linux environment.

.NET cross-platform development
Build cross-platform applications using .NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker...

Installation details

- Visual Studio core editor
- ASP.NET and web development
- .NET desktop development
- Data storage and processing *
  - Optional
    - [✓] SQL Server Data Tools
    - [ ] Azure Data Lake and Stream Analytics Tools
    - [✓] .NET Framework 4 – 4.6 development tools
    - [ ] F# desktop language support
- Individual components

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required -1 MB

Install while downloading ▾ Modify

Figure 3. Selecting the Data storage and processing options in the Visual Studio Installer window.

Once the installation is complete you should be able to see the **SQL Server Object Explorer** item under the **View** menu as shown below. Please click on it.

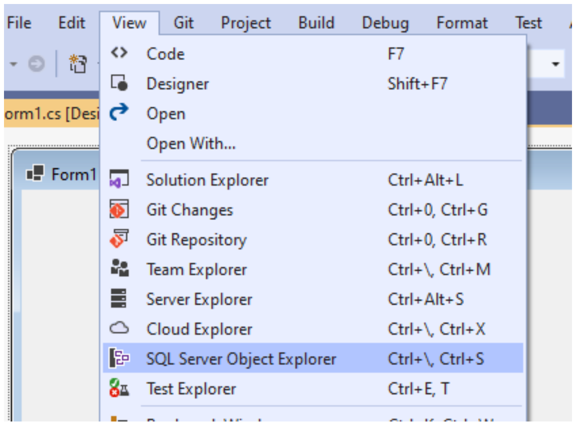

File Edit View Git Project Build Debug Format Test
Code F7
Designer Shift+F7
Open
Open With...
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+R
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Test Explorer Ctrl+E, T
orm1.cs [Desi
Form1

Figure 4. SQL Server Object Explorer item under the View menu

The following window should appear in Visual Studio. In this window, if you see (localdb) under SQL Server, then you are good with continuing the rest of the chapter.

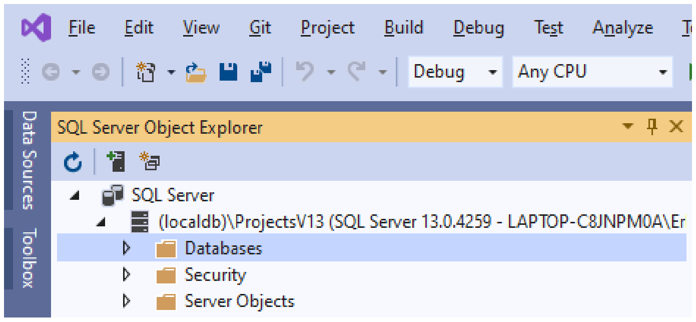

File Edit View Git Project Build Debug Test Analyze
Debug
Any CPU
Data Sources
Toolbox
SQL Server Object Explorer
SQL Server
(localdb)\ProjectsV13 (SQL Server 13.0.4259 - LAPTOP-C8JNPM0A\Er
Databases
Security
Server Objects

Figure 5. SQL Server Object Explorer window

2) Creating databases and tables

To create a new database, please right click on the Databases item and from the popup menu choose **Add New Database**, as shown below.

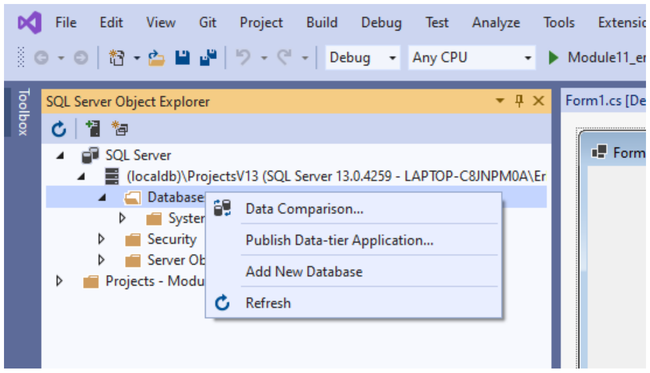
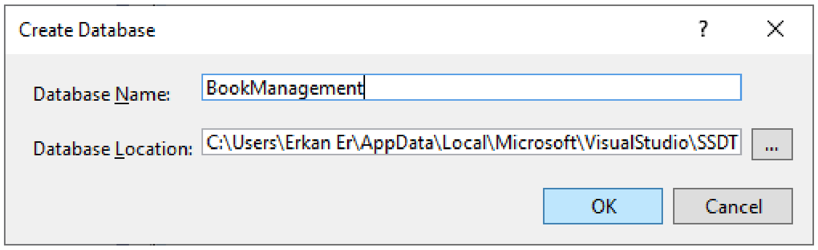

File Edit View Git Project Build Debug Test Analyze Tools Extensio
Debug
Any CPU
Module11_er
Toolbox
SQL Server Object Explorer
Form1.cs [De
Form
SQL Server
(localdb)\ProjectsV13 (SQL Server 13.0.4259 - LAPTOP-C8JNPM0A\Er
Database
Syster
Security
Server Ob
Projects - Modu
Data Comparison...
Publish Data-tier Application...
Add New Database
Refresh

Figure 6. Adding a new database

The following popup window should appear. Please enter **BookManagement** as the Database Name. You can leave the location as it is.



Database Name:	BookManagement
Database Location:	C:\Users\Erkan Er\AppData\Local\Microsoft\VisualStudio\SSDT

Figure 7. Entering a name for the database

As shown in Figure8 below, the database should be created and listed under the *Databases* folder inside SQL Server Object Explorer. In SQL, a database organizes and stores data via tables. A table stores data in rows and columns similar to a spreadsheet. Columns hold different piece of information about items stored in each row. For example, in a Books table, there might be columns to store id, title and page information about books. Each row would correspond to a distinct book item.

Let's create **Books** table. To do so, please right click on the **Tables** folder and choose **Add New Table** item from the popup menu.

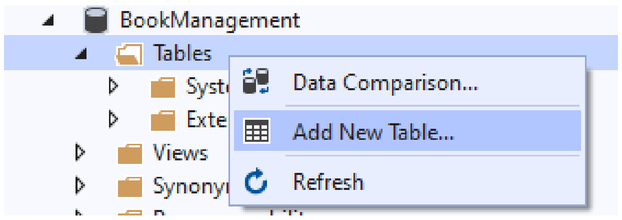
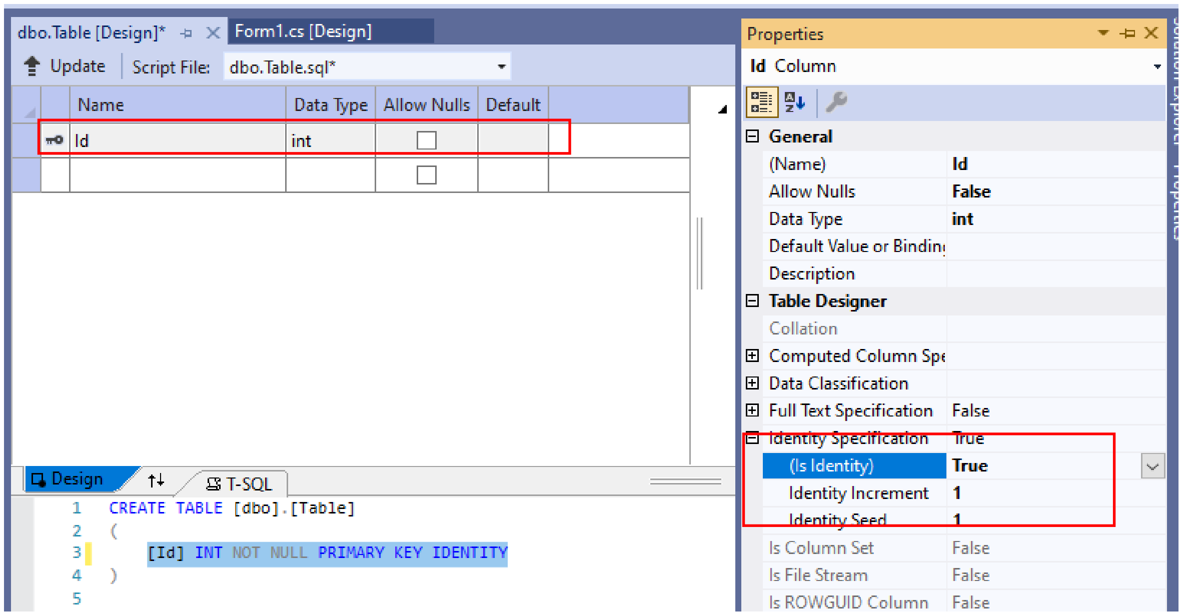


Figure 8. Adding a new table menu item



Name	Data Type	Allow Nulls	Default
Id	int	☐	
		☐	

```
CREATE TABLE [dbo].[Table]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY
)
```

General	
(Name)	Id
Allow Nulls	False
Data Type	int
Default Value or Bindin	
Description	
Table Designer	
Collation	
Computed Column Spe	
Data Classification	
Full Text Specification	False
Identity Specification	True
(Is Identity)	True
Identity Increment	1
Identity Seed	1
Is Column Set	False
Is File Stream	False
Is ROWGUID Column	False

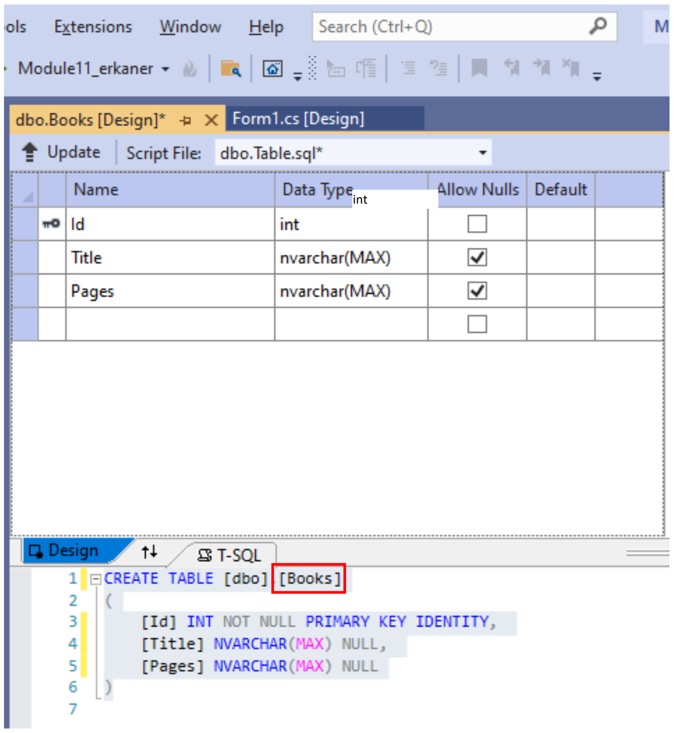
Figure 9. Adding a new table

The window shown in Figure 9 above should be displayed. In this window, at the top part you will see the design view while just at the bottom you will see the SQL code view. You can use either design or code views to configure the table. Any changes in one of them should be automatically reflected in the other one.

Most tables should have a **primary key**, which is a column that holds a unique value to identify a specific item or row in a table (such as a book). **Id** column has been automatically created and set as the primary key for the Books table (see the **key icon** next to the row). Id column's data type is int. We will set its **Identity Seed** and **Identity Increment** properties to 1 since we want Id to be 1 for the first book record added, and to be automatically incremented by 1 for next book records, which means users do not have to figure out a valid Id each time a new record is added. This is a very typical configuration for primary keys.

However, you can choose your own custom primary key. For example, for Books table, ISBN is a good candidate for primary key since every book has a unique ISBN. In this case, the user should provide a unique ISBN for each new book entered. Otherwise, the system will throw an error and will not allow adding the new record without a valid value for the primary key.

Next, in the SQL window at the bottom, please change [Table] to [**Books**] and add two more columns: **Title** and **Pages** (see Figure 10). *Title* will have **nvarchar(MAX)** and *Pages* will have **int** as data types. You can consider nvarchar as a data type to store string data. Other data types are beyond the scope of this course. The code inside the SQL window (at the bottom) should be automatically updated based on the changes in the design of the table.



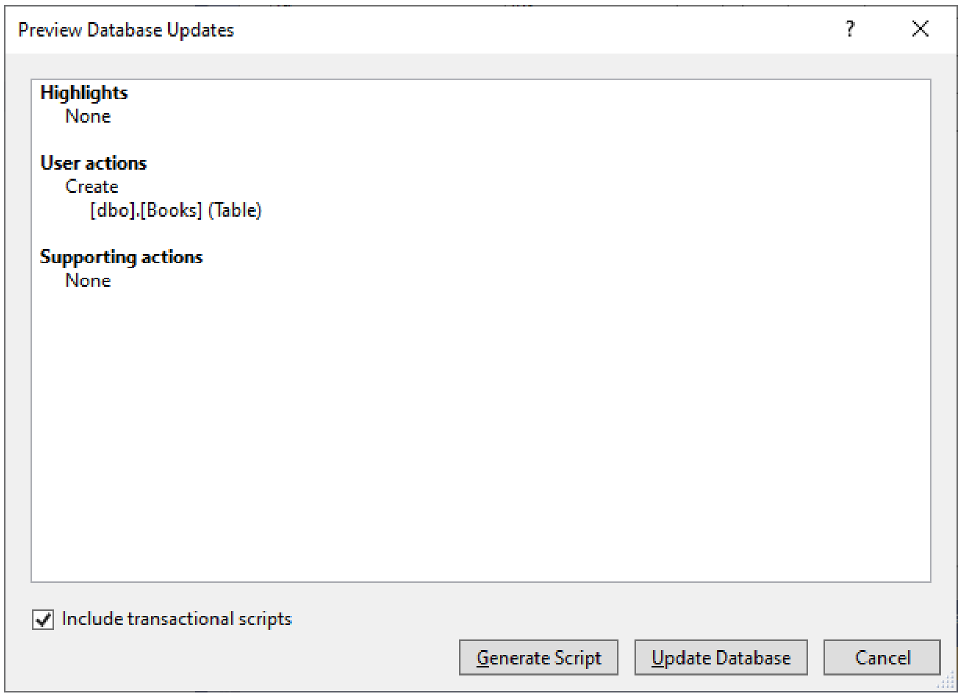
ols Extensions Window Help Search (Ctrl+Q)
Module11_erkaner
dbo.Books [Design]*
Form1.cs [Design]
Update Script File: dbo.Table.sql*
Design
T-SQL

Name	Data Type	Allow Nulls	Default
Id	int	☐	
Title	nvarchar(MAX)	✓	
Pages	nvarchar(MAX)	✓	
		☐	

```
CREATE TABLE [dbo].[Books]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [Title] NVARCHAR(MAX) NULL,
    [Pages] NVARCHAR(MAX) NULL
)
```

Figure 10. Adding new columns to Books table

To complete the creation of the table, please click on **Update** button on the top of the table design view. The following window (see Figure 11) should appear. Please click on **Update Database** button.



Preview Database Updates

Highlights
None

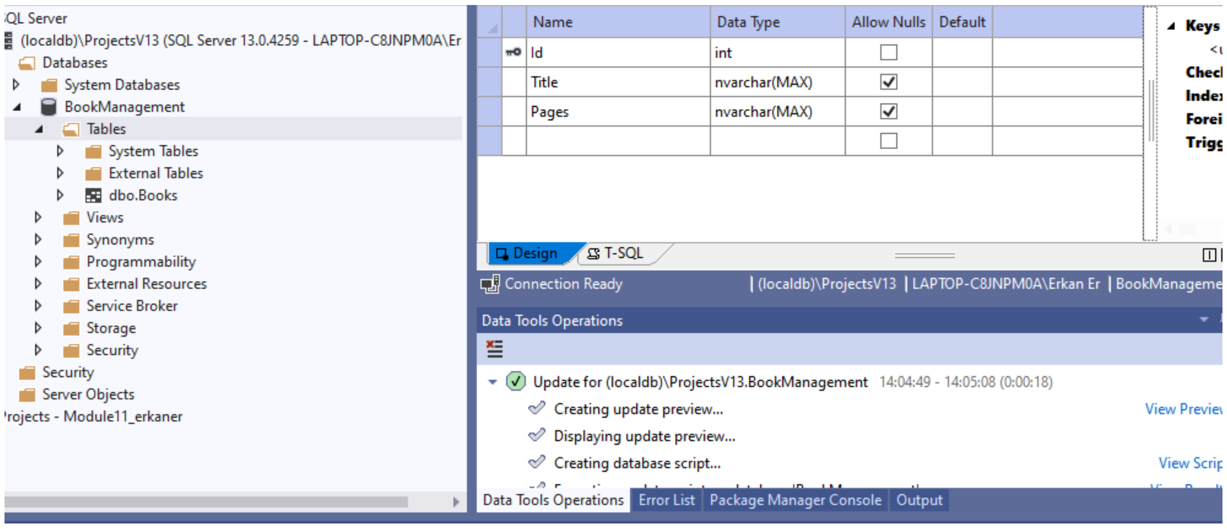
User actions
Create
[dbo].[Books] (Table)

Supporting actions
None

☑ Include transactional scripts

Figure 11. Updating the database to create the Books table

After this operation, the Books table should appear in the SQL Explorer window (see Figure 12).



```
QL Server
  (localdb)\ProjectsV13 (SQL Server 13.0.4259 - LAPTOP-C8JNPM0A\Er
    Databases
      System Databases
      BookManagement
        Tables
          System Tables
          External Tables
          dbo.Books
        Views
        Synonyms
        Programmability
        External Resources
        Service Broker
        Storage
        Security
    Security
    Server Objects
rojects - Module11_erkaner
```

	Name	Data Type	Allow Nulls	Default
🔑	Id	int	☐	
	Title	nvarchar(MAX)	✓	
	Pages	nvarchar(MAX)	✓	
			☐	

```
Connection Ready    | (localdb)\ProjectsV13 | LAPTOP-C8JNPM0A\Erkan Er | BookManageme

Data Tools Operations
Update for (localdb)\ProjectsV13.BookManagement   14:04:49 - 14:05:08 (0:00:18)
    Creating update preview...                                         View Previe
    Displaying update preview...
    Creating database script...                                        View Scrip
```

Figure 12. The Books table is added.

We will add some manual books data. To do so, please right click on the Books table, and choose **View Data** from the popup window, as shown in Figure 13.

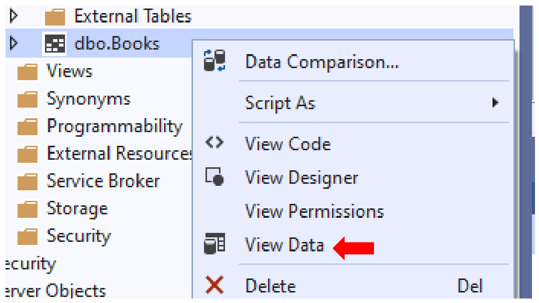
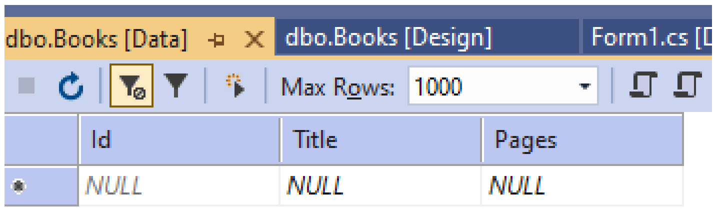


Figure 13. Accessing the data hold in the Books table

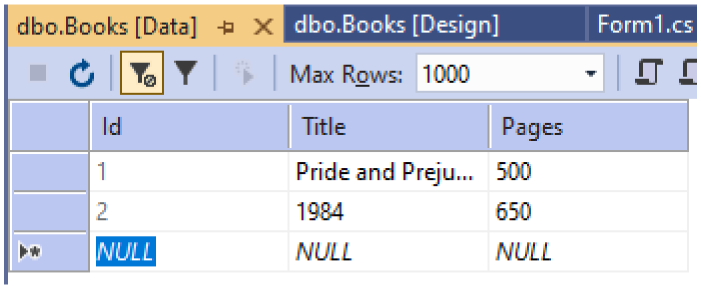
You should be able to display the Books table like a spreadsheet (see Figure 14). Id column cannot be edited since it will be automatically generated as new book records are added to the table.



Id	Title	Pages
NULL	NULL	NULL

Figure 14. Books table is initially empty

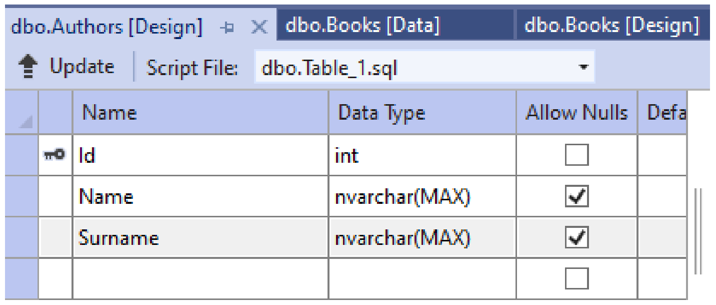
Please add several records by entering some Title and Pages values as shown in Figure 15. Id of the books should be automatically generated and displayed.



Id	Title	Pages
1	Pride and Preju...	500
2	1984	650
NULL	NULL	NULL

Figure 15. Adding two book records manually

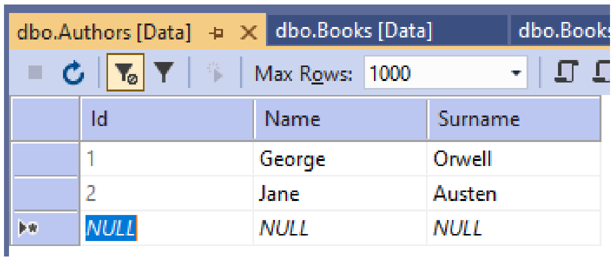
Next, we will create a new table called **Authors**. Please create this table with the columns shown in Figure 16. **Id** is the primary key for the Authors table. Please set its **Identity Seed** and **Identity Increment** properties to 1.



Name	Data Type	Allow Nulls	Defa
Id	int	☐	
Name	nvarchar(MAX)	✓	
Surname	nvarchar(MAX)	✓	
		☐	

Figure 16. Creating the Authors table

Next, we will add two author records to the Author table as shown in Figure 19 below.



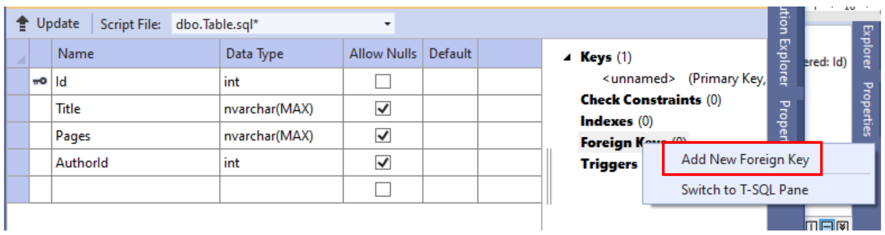
	Id	Name	Surname
	1	George	Orwell
	2	Jane	Austen
▶*	NULL	NULL	NULL

Figure 17. Adding two new authors to the Authors table

3) Creating foreign key

We will define a new column in the Books table, named `AuthorId`, to store the author of the books. This column will be a **Foreign Key** since it will refer to the `Id` column of the Authors table. In other words, through a foreign key, we associate the Book records with Author records.

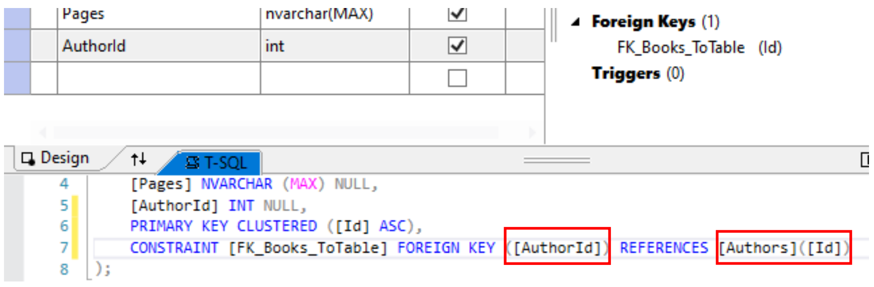
To add a foreign key, please selected the target column, and then right click on **Foreign Key** on the right-hand side of the window, as shown below.



	Name	Data Type	Allow Nulls	Default
🔑	Id	int	☐	
	Title	nvarchar(MAX)	✓	
	Pages	nvarchar(MAX)	✓	
	AuthorId	int	✓	
			☐	

Figure 18. Adding a Foreign Key

Then, do the changes marked below in the SQL code window to configure the Foreign Key constraint. As shown in the code line, **AuthorId** *references* the Id column of the Authors table.



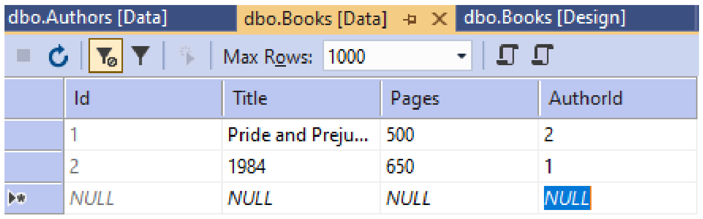
Name	Data Type	Allow Nulls
Pages	nvarchar(MAX)	✓
AuthorId	int	✓
		☐

```
    [Pages] NVARCHAR (MAX) NULL,
    [AuthorId] INT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_Books_ToTable] FOREIGN KEY ([AuthorId]) REFERENCES [Authors]([Id])
);
```

Figure 19. Configuring the Foreign Key constraint.

Since a foreign key is directly mapped with a primary key of another table, foreign key cannot be assigned with a value that does not exist in the referred primary key.

For example, we want to add author information for the books. We have two authors whose Ids are 1 and 2. We are allowed to enter only one of these two values in the AuthorId column of the Book table. If you enter 3, you should receive an error since there is no author with id 3.



	Id	Title	Pages	AuthorId
	1	Pride and Preju...	500	2
	2	1984	650	1
▶*	NULL	NULL	NULL	NULL

Figure 20. Adding AuthorId values in the Books table

3) Entity framework

Entity Framework (Code First) is basically a data-access technology that performs automatic mapping of your model (created in let’s say Visual Studio through set of classes) into a relational database. In other words, you implicitly design the underlying database for your project while working on the class structure of your project, which is referred to as the (domain) ***model***.

A **model** refers to the collection of the classes and the relationships among these classes. In **Domain-Driven Design** approach, the model itself is the core of your application development. Poor models will result in applications that are hard to develop and maintain. Whatever your model is (poor or great one), the entity framework is responsible for converting your model (i.e., *mapping*) into a relational database. So, the basic premise of Entity Framework is that programmers do not need to use a database management system to configure the database (create database and tables, define relationships, etc.), as we did previously above. Instead, programmers need to focus on the application itself and write the necessary code to build the domain model as per the application requirements.

Let’s image that we are building a Book Store application. Probably, in such an application, the model will have classes like Book, Author, Publisher, etc. To keep it simple, let’s create Book class, by using the following declaration:

```
public class Book
{
 public int Id { get; set; }
 public string Title { get; set; }
 public int Year { get; set; }
}
```

Our (very) simple model is ready. Now, it is time to let Entity Framework map our model (composed of only Book class) to a relational database. As you may expect, the resultant database will have a single table, called Book, with four columns (corresponding to each of the properties in Book class).

4) Installing and Configuring Entity Framework

To be able to use Entity Framework, we need to install it and then enable it with several configurations.

We will install Entity Framework by using NuGet Packages, which is defined as "a Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects that use the .NET Framework." (https://www.nuget.org/).

Please right-click on your project name in the Solution Explorer and choose **Manage NuGet Packages** as seen in the following figure.

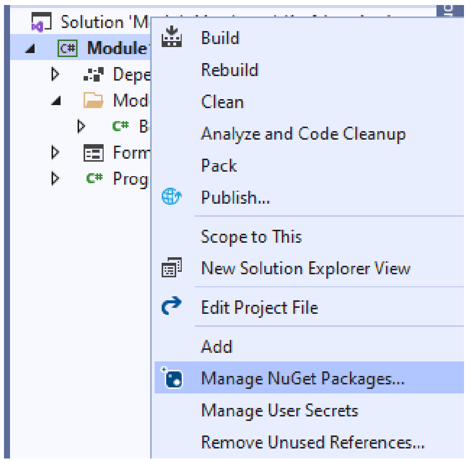
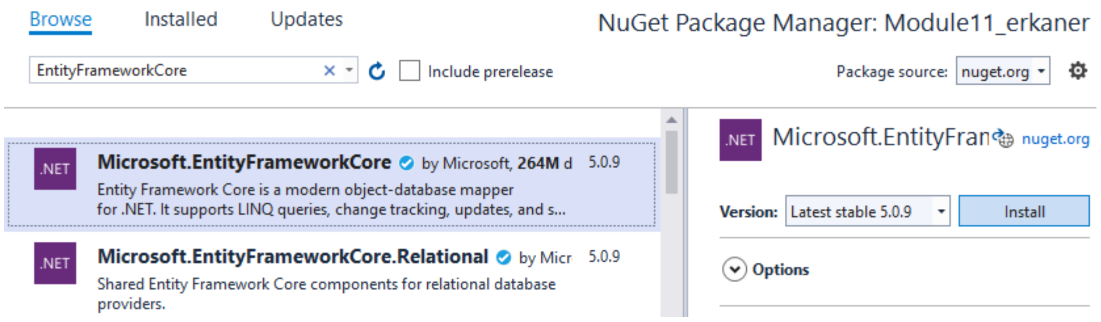


Figure 21. Manage NuGet Packages item

In the *Browse* tab, please search for **EntityFrameworkCore**. When Microsoft.EntityFrameworkCore appears in the list, please select it and the click Install. See Figure 22.

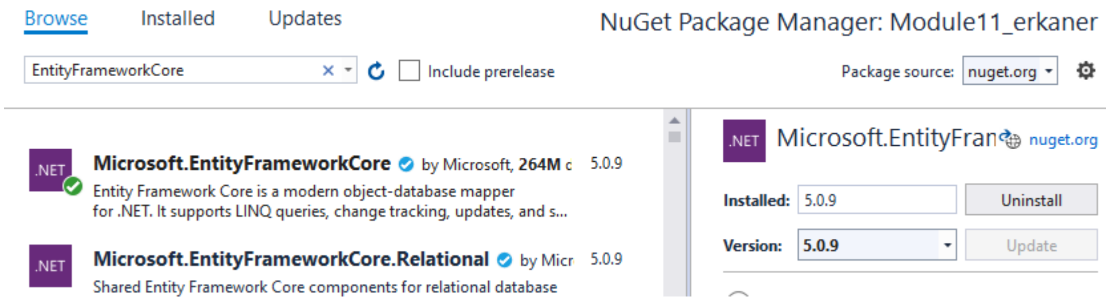

Browse
Installed
Updates
NuGet Package Manager: Module11_erkaner
EntityFrameworkCore
Include prerelease
Package source: nuget.org
.NET
Microsoft.EntityFrameworkCore by Microsoft, 264M d 5.0.9
.NET
Microsoft.EntityFrameworkCore.Relational by Micr 5.0.9
Microsoft.EntityFran nuget.org
Version: Latest stable 5.0.9
Install
Options

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and s...

Shared Entity Framework Core components for relational database providers.

Figure 22. Installing EntityFrameworkCore

If any additional windows appear for confirmation, please click **I Agree**. Once installed, a green checkbox should appear next to Microsoft.EntityFrameworkCore, as seen in Figure 23 below.

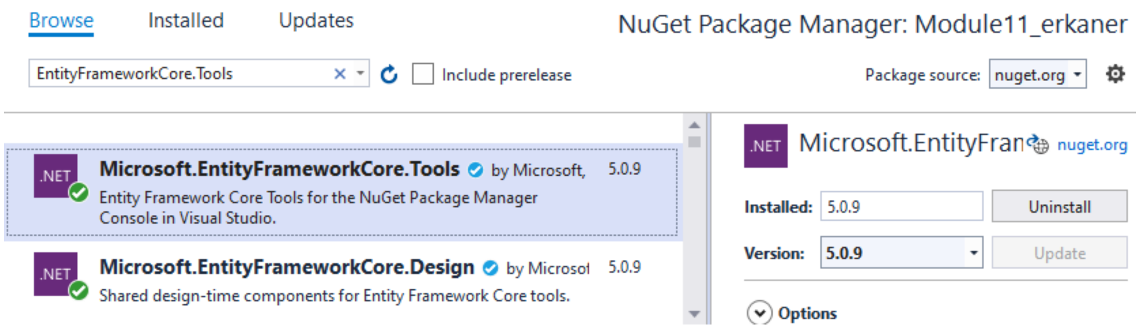

Browse
Installed
Updates
NuGet Package Manager: Module11_erkaner
EntityFrameworkCore
Include prerelease
Package source: nuget.org
.NET
Microsoft.EntityFrameworkCore by Microsoft, 264M c 5.0.9
.NET
Microsoft.EntityFrameworkCore.Relational by Micr 5.0.9
Microsoft.EntityFran nuget.org
Installed: 5.0.9
Uninstall
Version: 5.0.9
Update

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and s...

Shared Entity Framework Core components for relational database

Figure 23. EntityFrameworkCore is installed.

Similarly, please install *EntityFrameworkCore.***Tools** and *EntityFrameworkCore.***Design**, as shown in Figure 24 below.



EntityFrameworkCore.Tools

Package source: nuget.org

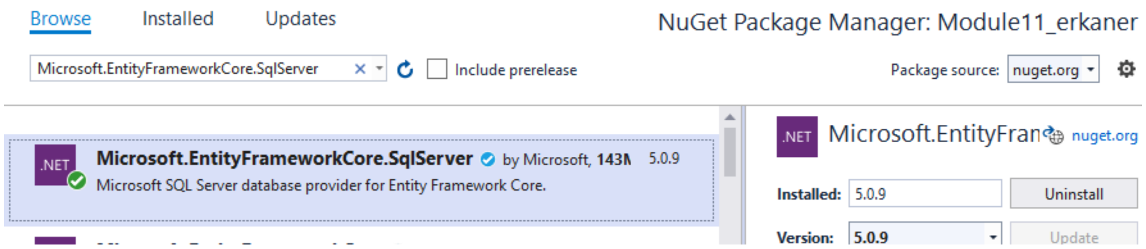
Microsoft.EntityFrameworkCore.Tools by Microsoft, 5.0.9
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Microsoft.EntityFrameworkCore.Design by Microsoi 5.0.9
Shared design-time components for Entity Framework Core tools.

Microsoft.EntityFran nuget.org
Installed: 5.0.9 Uninstall
Version: 5.0.9 Update

Figure 24. Installing other components of EntityFrameworkCore.

Last, you should install the *Microsoft.EntityFrameworkCore.***SqlServer** library, as shown in Figure 25.



Microsoft.EntityFrameworkCore.SqlServer

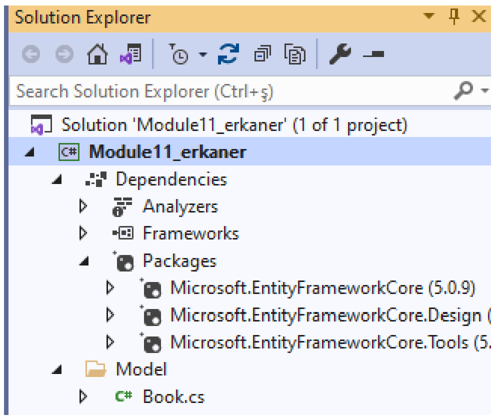
Package source: nuget.org

Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 143M 5.0.9
Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFran nuget.org
Installed: 5.0.9 Uninstall
Version: 5.0.9 Update

Figure 25. Installing other components of EntityFrameworkCore.SqlServer.

Now, EntityFramework is installed in your application. If you check the Dependencies section in the solution explorer you should see the EntityFramework related items (see Figure 26).



Solution 'Module11_erkaner' (1 of 1 project)
Module11_erkaner
Dependencies
Analyzers
Frameworks
Packages
Microsoft.EntityFrameworkCore (5.0.9)
Microsoft.EntityFrameworkCore.Design (
Microsoft.EntityFrameworkCore.Tools (5.
Model
Book.cs

Figure 26. EntityFrameworkCore package is listed in Solution Explorer.

5) Creating the DbContext file

To be able to use EntityFramework, you need to configure it. To do that, you need to add a new class (see Figure 27) that inherits from **`DbContext`** class (which already comes with EntityFramework installation). `DbContext` is the class by which you can explain your model to Entity Framework, which in turn maps your model to a relational database. DbContext is the core class which has the logic that drives Entity Framework Code First.

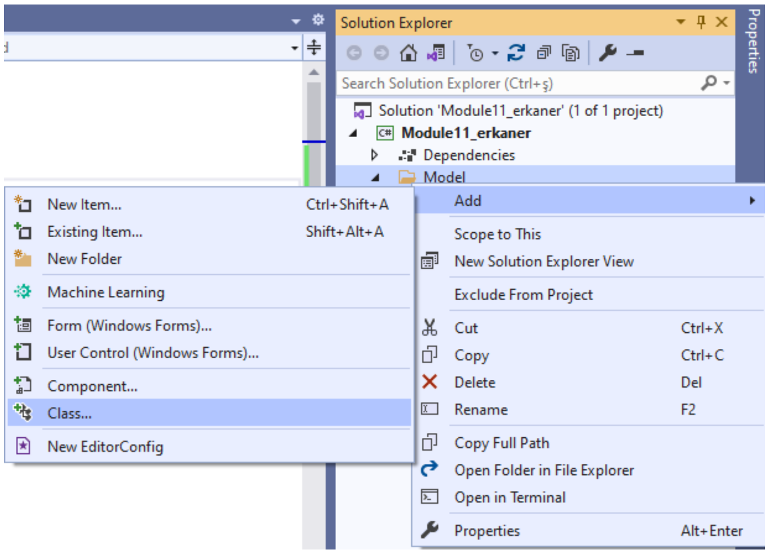


Figure 27. Adding a new class.

Please name the class file as BookDbContext (see Figure 28). Including the DbContext keyword at the end of the class name is a good practice.

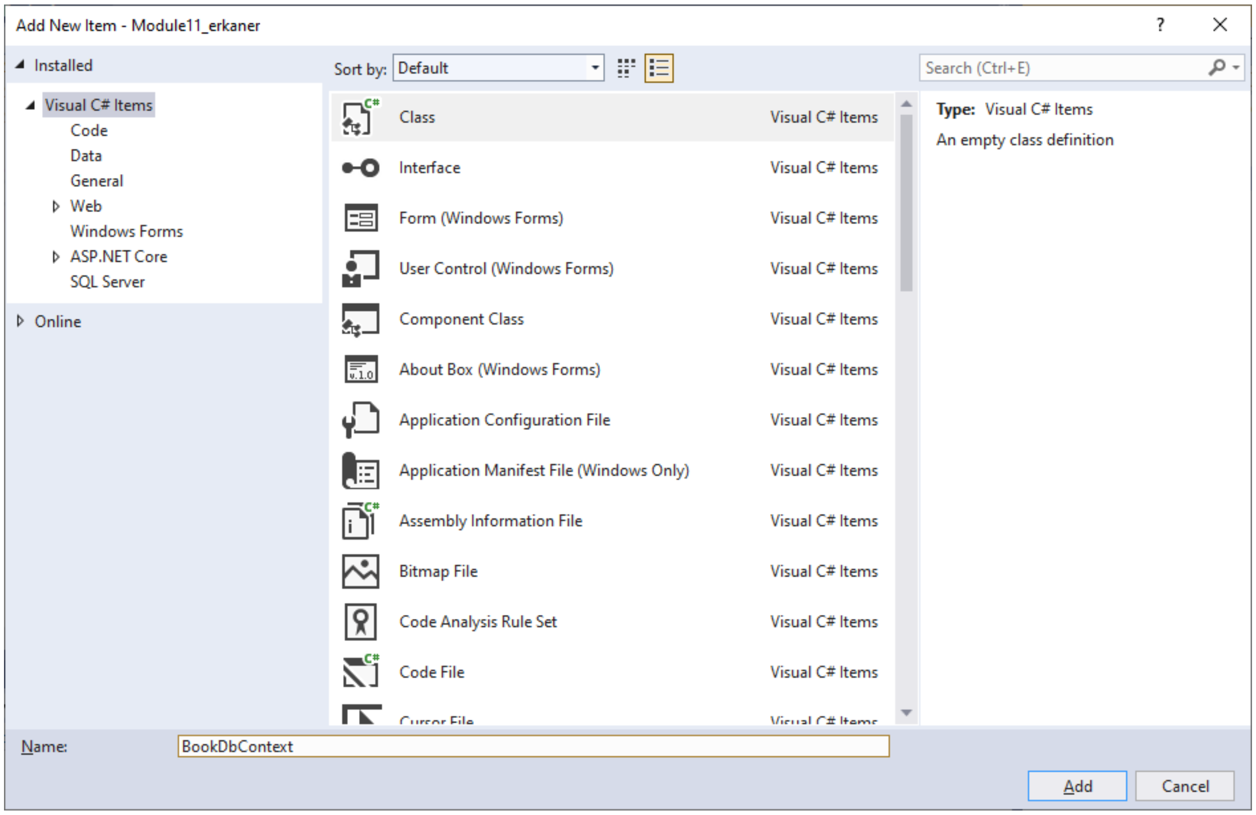
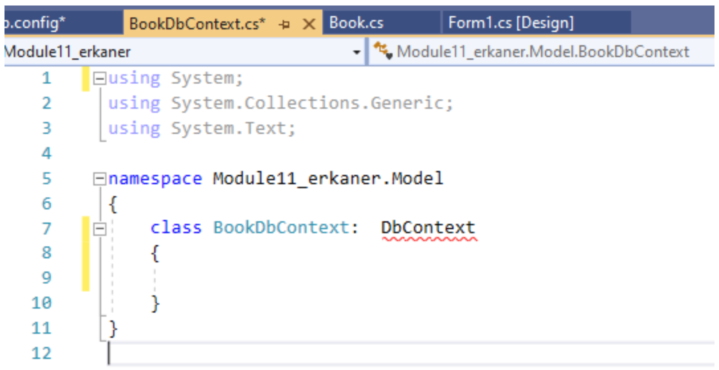

Add New Item - Module11_erkaner
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
ASP.NET Core
SQL Server
Online
Sort by: Default
Search (Ctrl+E)
Class Visual C# Items
Interface Visual C# Items
Form (Windows Forms) Visual C# Items
User Control (Windows Forms) Visual C# Items
Component Class Visual C# Items
About Box (Windows Forms) Visual C# Items
Application Configuration File Visual C# Items
Application Manifest File (Windows Only) Visual C# Items
Assembly Information File Visual C# Items
Bitmap File Visual C# Items
Code Analysis Rule Set Visual C# Items
Code File Visual C# Items
Cursor File Visual C# Items
Type: Visual C# Items
An empty class definition
Name: BookDbContext
Add
Cancel

Figure 28. Creating the BookDbContext class file.

As mentioned before, the `BookDbContext` class needs to inherit from `DbContext`. When inherited from `DbContext`, the `BookDbContext` class will be able to access all methods of `DbContext`, which we need to form the database based on the model of application.

Your code file should be displayed once it is created. Please write the code to inherit `BookDbContext` from `DbContext` as shown below. Initially, a jagged red line should appear under DbContext, indicating an error. See Figure 29 below.



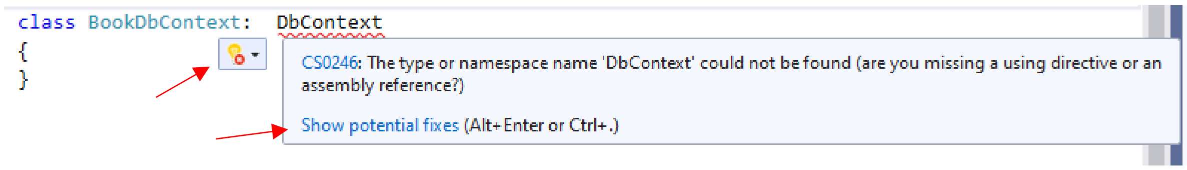
```
using System;
using System.Collections.Generic;
using System.Text;

namespace Module11_erkaner.Model
{
    class BookDbContext:  DbContext
    {

    }
}
```

Figure 29. Inheriting from DbContext.

Once you mouse over it, you should see the error messages displayed in a tooltip. The error message tells that DbContext class is not accessible because it is not properly imported with using statements. Visual Studio can automatically fix this error. Please click one of the items indicated with arrows in Figure 30.



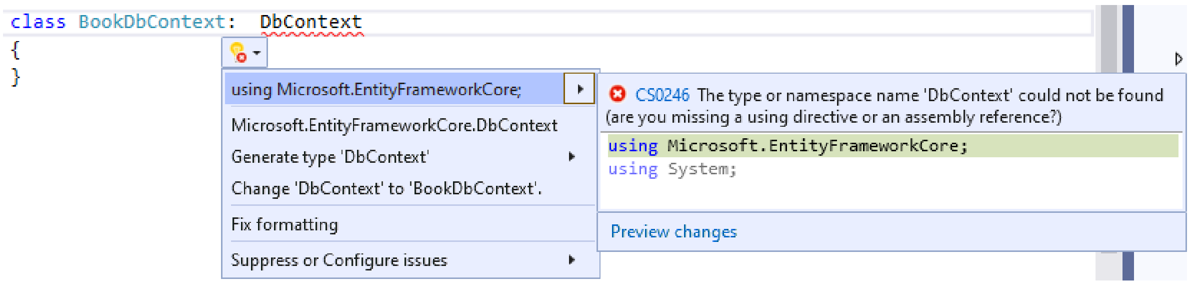
```
class BookDbContext:  DbContext
{
}
```

CS0246: The type or namespace name 'DbContext' could not be found (are you missing a using directive or an assembly reference?)

Show potential fixes (Alt+Enter or Ctrl+.)

Figure 30. Displaying the potential fix window.

The potential fixes should be displayed in a small popup window as shown in Figure 31. In this window, choose the first item (**using Microsoft.EntityFramework.Core**) and click on it twice to apply the automatic fix.



```
class BookDbContext:  DbContext
{
}
```

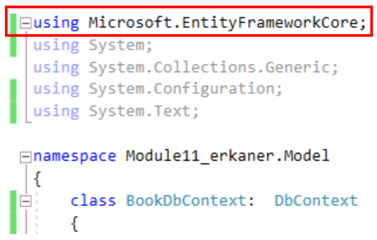
using Microsoft.EntityFrameworkCore;
Microsoft.EntityFrameworkCore.DbContext
Generate type 'DbContext'
Change 'DbContext' to 'BookDbContext'.
Fix formatting
Suppress or Configure issues
Preview changes

CS0246 The type or namespace name 'DbContext' could not be found (are you missing a using directive or an assembly reference?)

```
using Microsoft.EntityFrameworkCore;
using System;
```

Figure 31. Applying the potential fix.

The required using statement should be added at the top of the code file as shown in Figure 32.



```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Text;

namespace Module11_erkaner.Model
{
    class BookDbContext:  DbContext
    {
```

Figure 32. The using statement is added.

One important configuration in DbContext file that you will always have to do is setting the connection string. A connection string is what your C# application needs for establishing a connection to an existing (or to-be-created) database. To add a connection string, we will `override` the `OnConfiguring` method of `DbContext` and we will pass the connection string inside the `UseSqlServer` method.

We will not go into further details about `OnConfiguring method and DbContextOptionsBuilder` within the scope of this module and course. You can use the following code to configure the database connection in your future projects. You will need to update the *Database* name accordingly in your new projects. For this project, the database name will be `BookManagement_v1`.

```
class BookDbContext: DbContext
{
  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=BookManagement_v1");
  }
}
```

6) DbSet types

To ease your understand, you can just think that a DbContext corresponds to whole database. In this database, we need tables, right? For example, in our scenario we will need a table for recording instances of Book class. We can indicate the tables that we want in our database in DbContext class by using **`DbSet`** type. In other words, we need to use `DbSet` type to tell Entity Framework that `Book` class should be included when mapping the domain model to the database. Otherwise, Entity Framework **will not** create a corresponding table in the database for the `Book` class.

The Books object with `DbSet` type will also act as ***a proxy*** to the Books table in the database when we need to retrieve some book records, or when we need to create, update or delete a book entry. Below is the updated definition of the BookDbContext file:

```
class BookDbContext: DbContext
{
  public DbSet<Book> Books { get; set; } //a proxy to the Books table in the database

  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=BookManagement_v1");
  }
}
```

7) Using migrations to create the database for the first time

We have completed all required configurations and now we can ask EntityFramework to create the database based on the existing configurations and the domain model.

We need to use some simple commands of EntityFramework using **Package Manager Console**. To open this console, click on *View* menu, choose *Other Windows*, and then click on *Package Manager Console* in the submenu. This is illustrated in Figure 33 below.

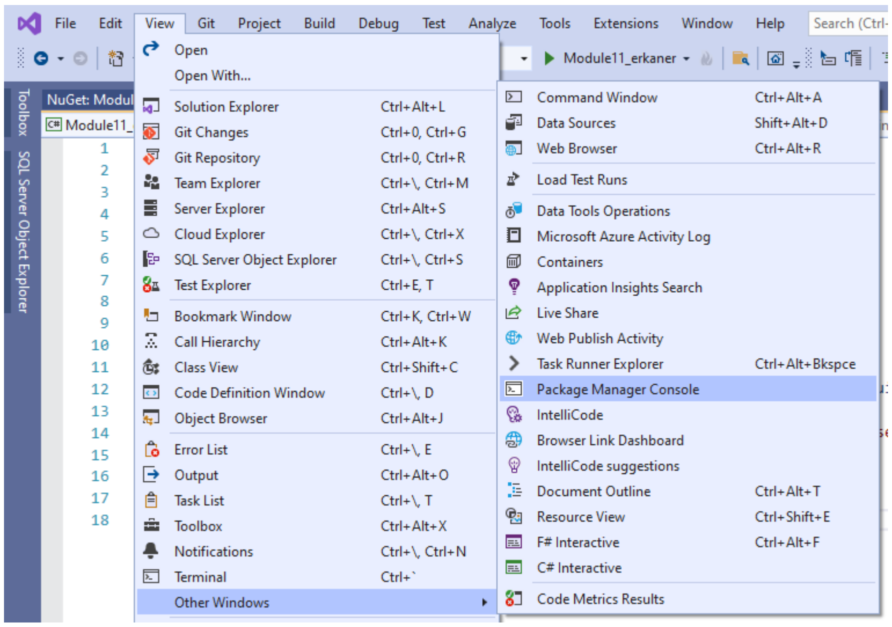
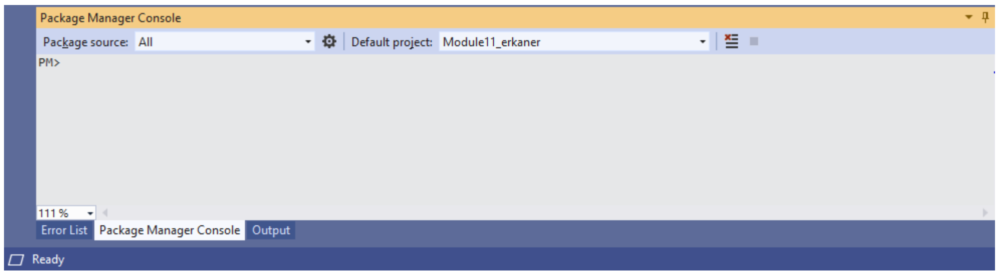

File
Edit
View
Git
Project
Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
Search (Ctrl-
Module11_erkaner
Toolbox
SQL Server Object Explorer
NuGet: Modul
Module11_
Open
Open With...
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+R
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Test Explorer Ctrl+E, T
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+\, Ctrl+N
Terminal Ctrl+`
Other Windows
Command Window Ctrl+Alt+A
Data Sources Shift+Alt+D
Web Browser Ctrl+Alt+R
Load Test Runs
Data Tools Operations
Microsoft Azure Activity Log
Containers
Application Insights Search
Live Share
Web Publish Activity
Task Runner Explorer Ctrl+Alt+Bkspce
Package Manager Console
IntelliCode
Browser Link Dashboard
IntelliCode suggestions
Document Outline Ctrl+Alt+T
Resource View Ctrl+Shift+E
F# Interactive Ctrl+Alt+F
C# Interactive
Code Metrics Results

Figure 33. Opening Package Manager Console.

Package Manager Console should appear at the bottom part of the Visual Studio (see Figure 34). As its name implies, this is a console window where you can write and execute some commands.



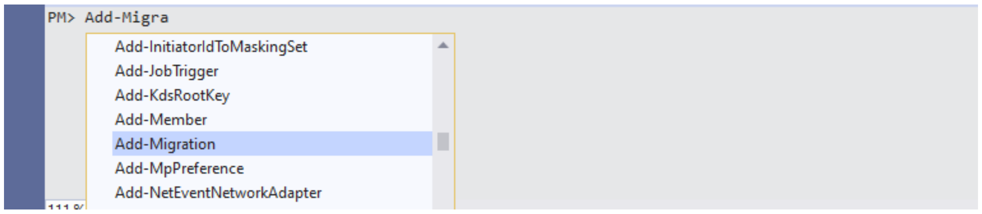
Package source: All
Default project: Module11_erkaner

```
PM>
```

Figure 34. Package Manager Console.

Entity Framework uses the ***migrations*** to build and update the database based on the changes and configurations in the domain model. Right now, we need migrations to create the database for the first time. Later, as we do changes in our domain model (e.g., adding a new property to a class), we need to create new migrations to update the database accordingly.

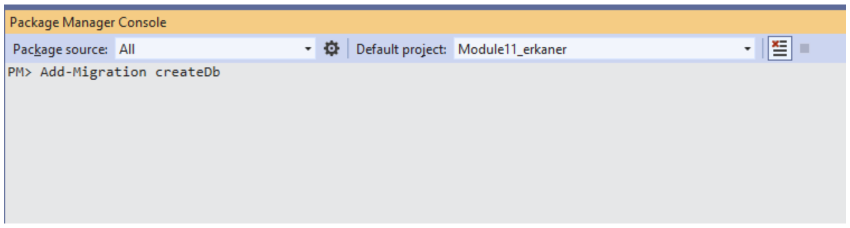
We will use Add-Migration command. Please type Add in the package manager console and press the tab key. Matching commands should be listed. This is a useful feature when you cannot remember the whole command. Please choose Add-Migration and press enter.



```
PM> Add-Migra
```

Figure 35. Finding the Add-Migration command.

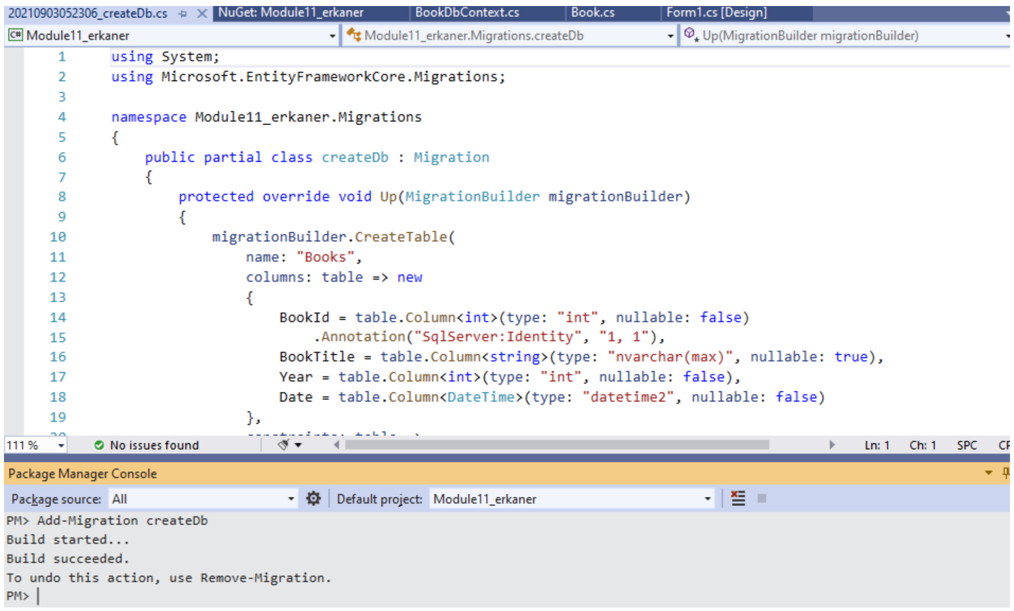
You need to provide a name for the migration. Please type **createDb** after Add-Migration. Your console should look like Figure 36.



```
PM> Add-Migration createDb
```

Figure 36. Adding a migration.

After running the Add-Migration command, you should see the migration file automatically opened in VS (see Figure 37). This file contains all changes to be made to the database.



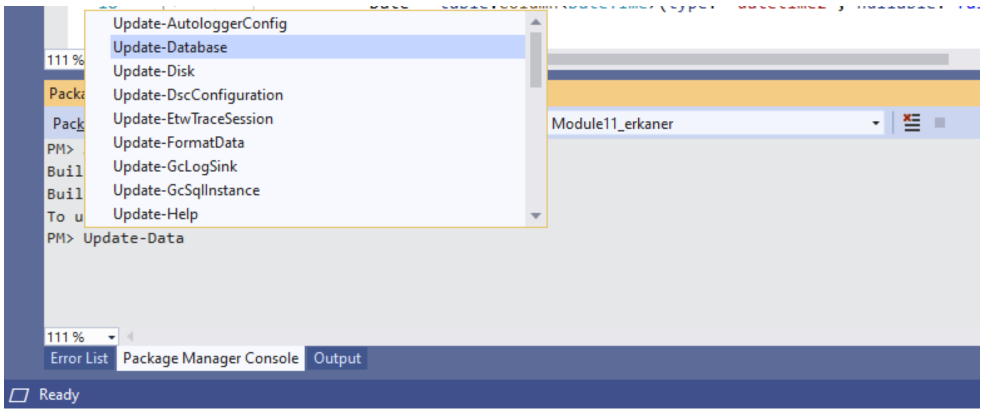
```csharp
using System;
using Microsoft.EntityFrameworkCore.Migrations;

namespace Module11_erkaner.Migrations
{
    public partial class createDb : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Books",
                columns: table => new
                {
                    BookId = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    BookTitle = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Year = table.Column<int>(type: "int", nullable: false),
                    Date = table.Column<DateTime>(type: "datetime2", nullable: false)
                },
```

```
PM> Add-Migration createDb
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

Figure 37. The using statement is added.

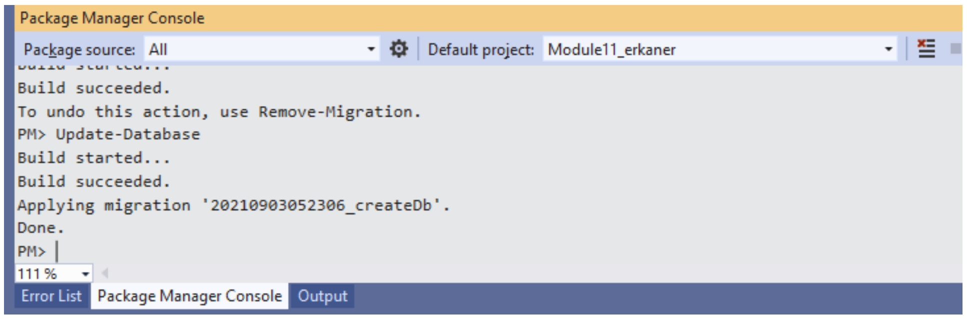
To apply the changes in the migration file, please run the **Update-Database** command and press enter as shown in Figure 38.



```
PM>
Buil
Buil
To u
PM> Update-Data
```

Figure 38. The using statement is added.

Once the command is executed, you should obtain the following output as shown in Figure 39. If you see **Done** in the output without any error messages, then the database should be created successfully.



```
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database
Build started...
Build succeeded.
Applying migration '20210903052306_createDb'.
Done.
PM>
```

Figure 39. The using statement is added.

Then, please check the SQL Server Object Explorer window. The **BookManagement_v1** database should be available in the Databases folder, and the database should have the **Books** table as seen in Figure 40.

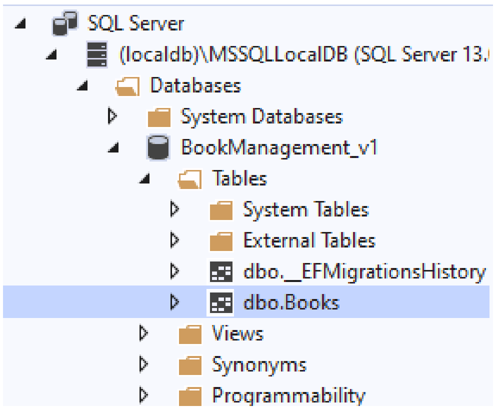


Figure 40. The using statement is added.

7) Creating a simple application to create and list books

We will create a simple application to allow user to create and list books. Please add a button (named as `btn_addBook`) and listbox control (named as `lst_books`) to the form as shown in Figure 41.

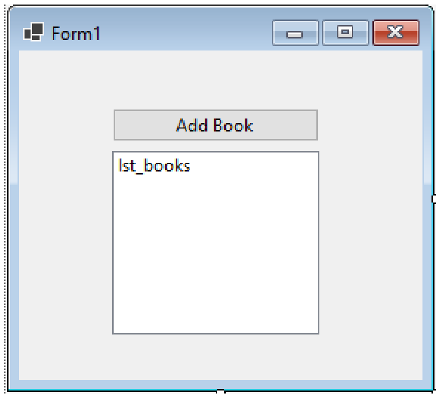


Figure 41. The using statement is added.

When the button is clicked, we want to add a sample book record to the database. We need to first create a book object with valid title and year values. Next, we need to create an instance of `BookDbContext`, named as **`_db`**. Through `_db`, we can access the **Books** collection (which is a `DbSet` type as defined inside the BookDbContext file). Since it is a `DbSet`, we can call the `Add` method and pass the book object. Last, we need to call the `SaveChanges` method to ask EntityFramework to execute this add operation. The complete code is shown in Figure 42.

```
private void btn_addBook_Click(object sender, EventArgs e)
{
    Book book = new Book() { BookTitle = "Tutunamayanlar", Year = 1972 };
    BookDbContext _db = new BookDbContext();
    _db.Books.Add(book);
    _db.SaveChanges();
}
```

Figure 42. The using statement is added.

If you go ahead and view the data in the Books table (see Figure 43), you should see the new book record that we have just added (see Figure 44).

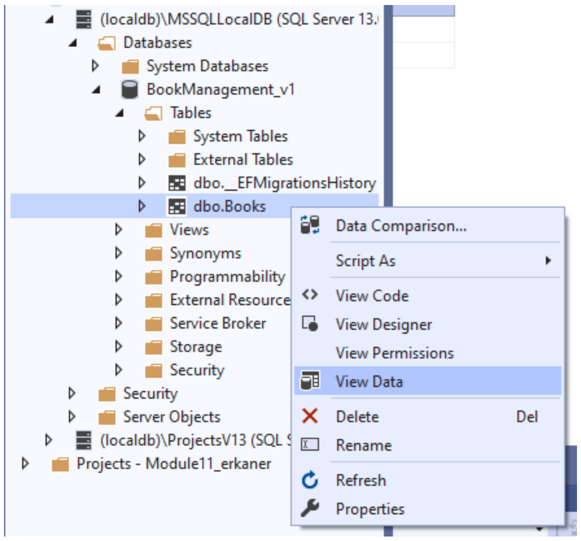
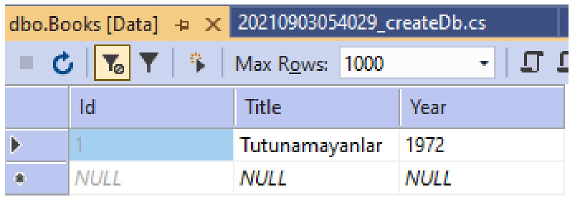


Figure 43. Viewing the data in the Books table.


dbo.Books [Data] 20210903054029_createDb.cs

Max Rows: 1000

	Id	Title	Year
▶	1	Tutunamayanlar	1972
✱	NULL	NULL	NULL

Figure 44. New book record is added to the Books table.

Now, we want to display the existing book records in the listbox. For this purpose, we will define a method called `BindBooks`. Inside this method, we will create an instance of BookDbContext, named as **_db**. To fetch the book records from the database we need to call `_db.Books.ToList()` and assign it to the `DataSource` property of `lst_books`. We should also set the ValueMember and DisplayMember properties properly. The complete definition of the `BindBooks` method is displayed below.

```
public void BindBooks()
{
    BookDbContext _db = new BookDbContext();

    lst_books.DataSource = _db.Books.ToList();
    lst_books.ValueMember = "Id";
    lst_books.DisplayMember = "Title";
}
```

Figure 45. Defining the BindBooks method.

Now, we can call this method in the form load to list the books when the application is run. We should also call this method when a new book is created (so that it can be immediately listed). The related code is shown below.

```
private void btn_addBook_Click(object sender, EventArgs e)
{
    Book book = new Book() { Title = "Tutunamayanlar", Year = 1972 };
    BookDbContext _db = new BookDbContext();
    _db.Books.Add(book);
    _db.SaveChanges();

    BindBooks();
}

private void Form1_Load(object sender, EventArgs e)
{
    BindBooks();
}
```

Figure 46. Calling the BindBooks method.

If you run your application, the single book record that we have created should be displayed in the listbox as shown in Figure 47.

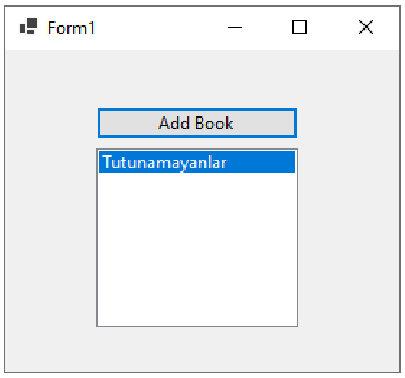


Figure 47. The book records are listed.