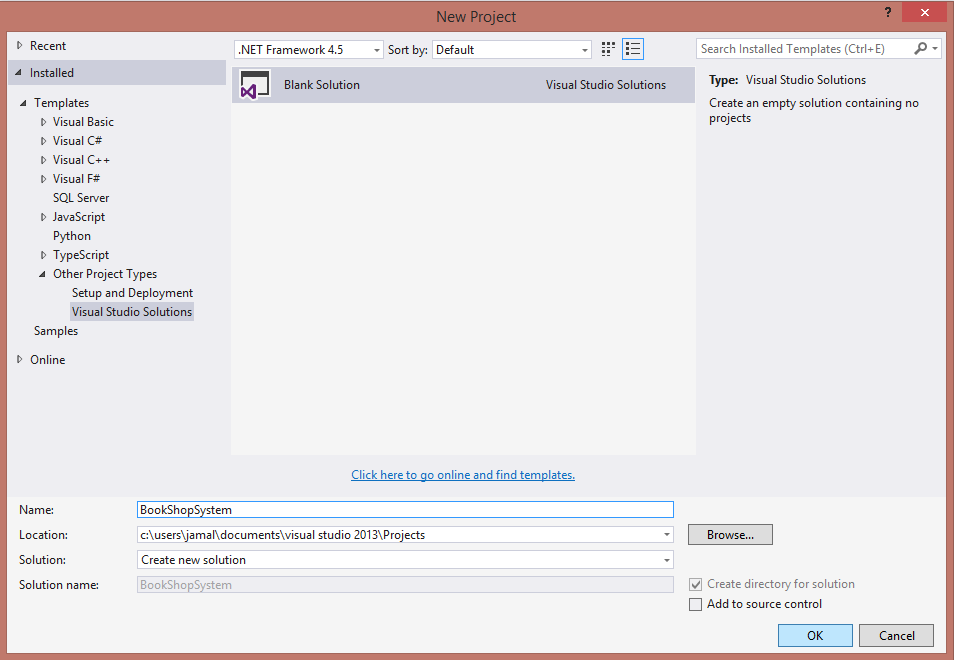
# Entity Framework Code First Exercise: Book Shop

This document defines the homework assignments from the ["Database Applications" Course @ Software University](https://softuni.bg/trainings/21/Database-Applications-Mar-2015). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Create a Database for Student System using Code First

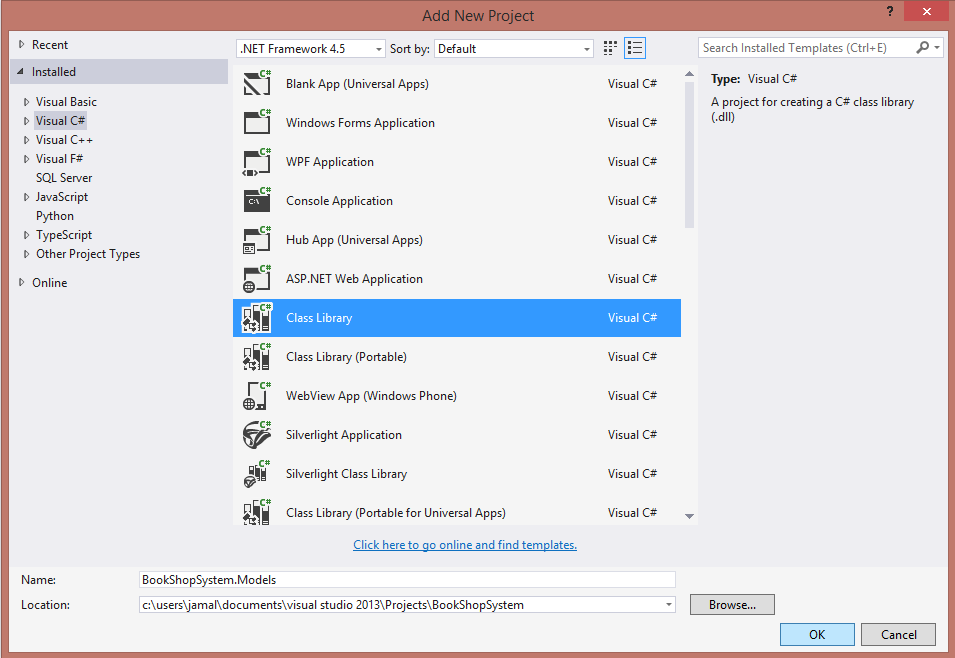
First of all, create a **Blank Solution** and name it accordingly - in our case, **BookShopSystem** sounds like a nice name.



We will separate code into different projects - **Models** (holding the classes Entity Framework will use as a basis for creating our database), **Data** (holding Entity Framework specific logic) and **Client** (holding our main application).

### Step 1 - Model the Database

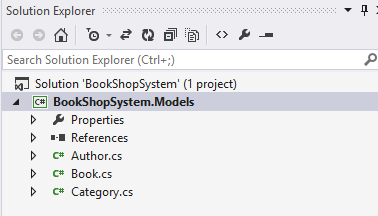
The **Models** project will not be an executable (i.e. it will only hold classes). Therefore, it will be a class library (.dll).



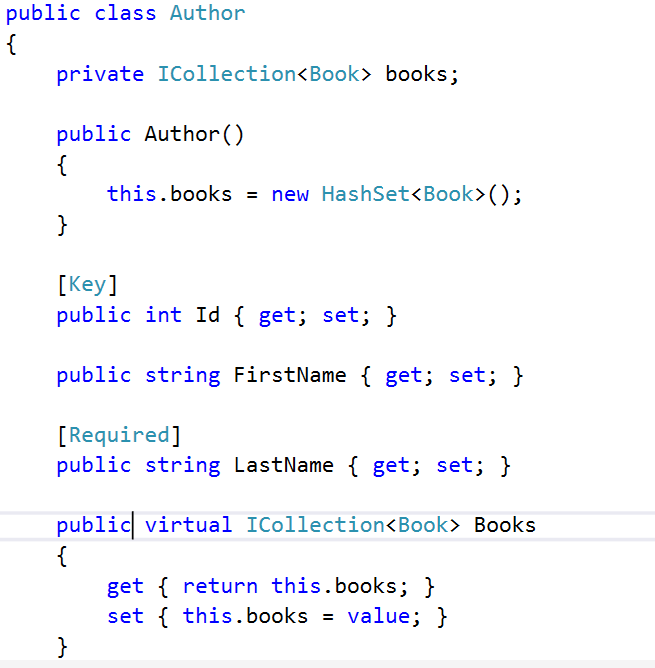
A book shop keeps **books**. A book can have one **author** (for the sake of simplicity) and many **categories**. Let's create a class for each of the main tables.

* **Book** - id, title (between 1..50 symbols), description (optional, up to 1000 symbols), edition type (**Normal**, **Promo** or **Gold**), price, copies, release date (optional)
* **Author** - id, first name (optional) and last name
* **Category** - id, name

Assume everything **not market optional** is mandatory.



The **classes** should describe with **properties** each of the **table columns**.

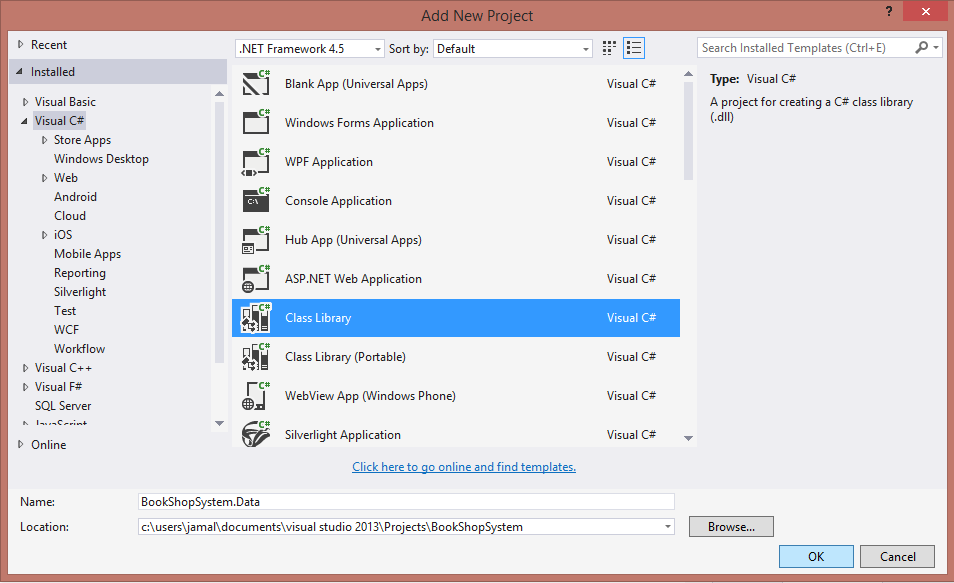


Add constraints as described above using attributes (e.g. **Required**, **MinLength**, **MaxLength**, etc). Do the same for the **Book** and **Category** models. Make sure you add **navigation properties** when there are relations.

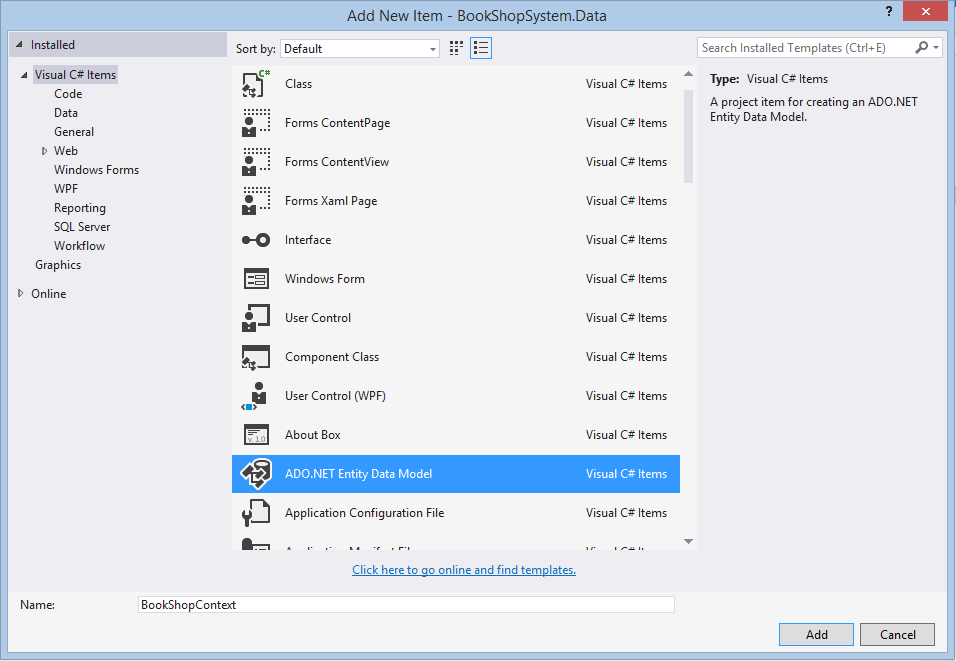
**Important** **Note**: Always make **navigation properties** **virtual**, so Entity Framework can **override** them and apply **lazy loading**.

### Step 2 - Create the Data Layer

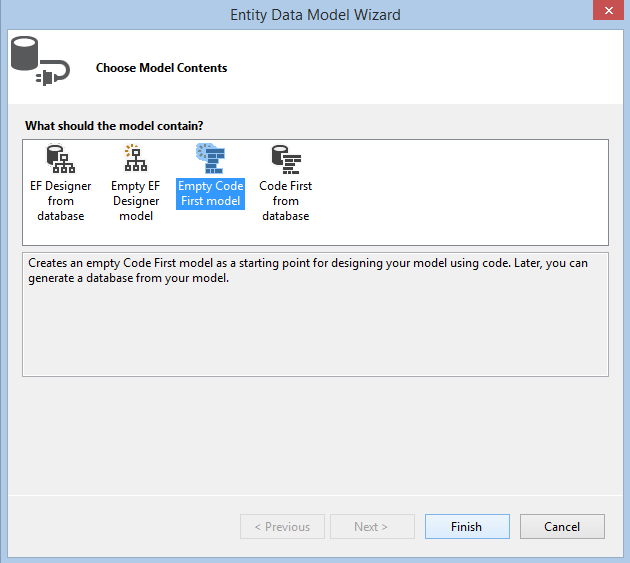
Once the models are done, our next step is to write the so-called **Data Layer**. Let's create it in a separate project.



In the newly created project, add a new ADO.NET Entity Data Model. Name it **BookShopContext**.

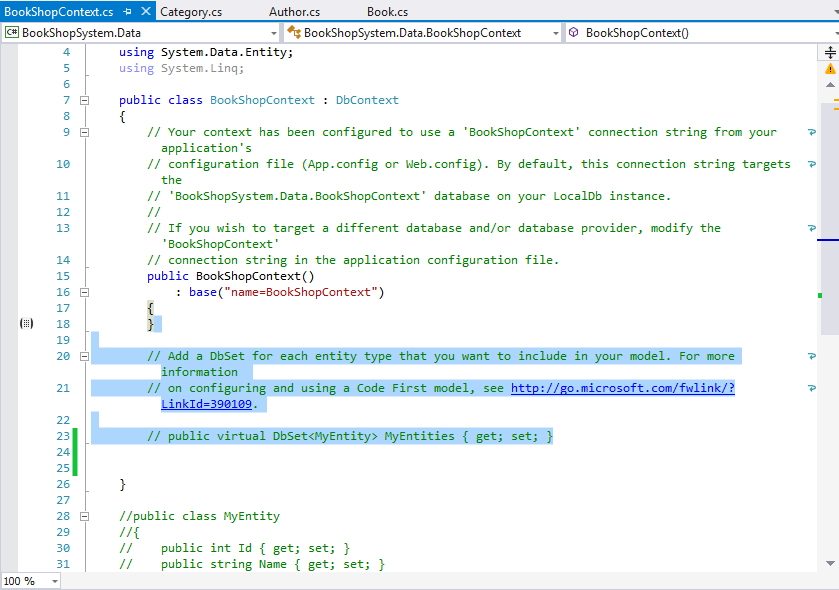


From the Model Wizard, select **Empty Code First Model** (meaning we will model our database from the code we write alone).

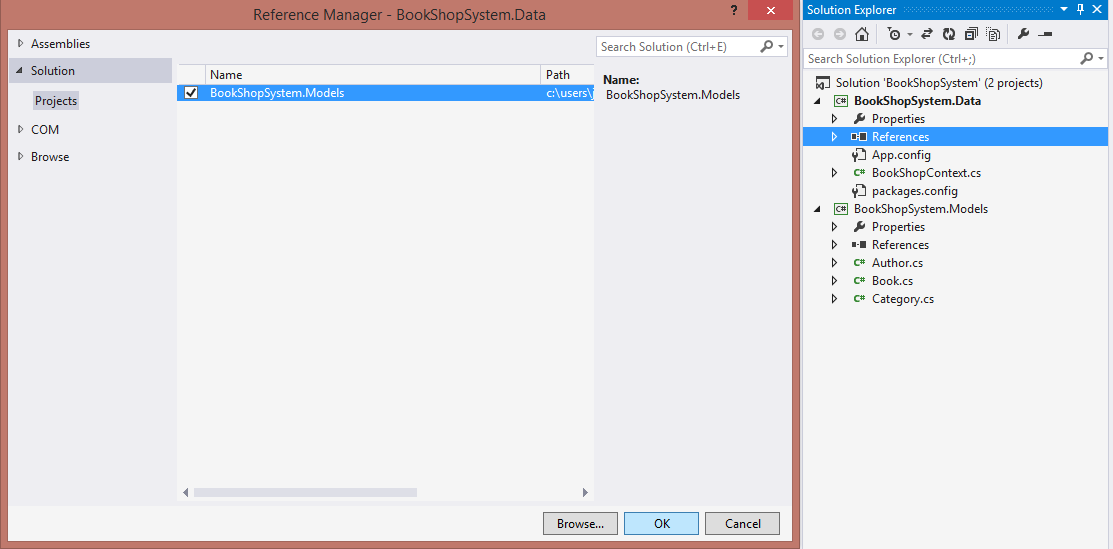


The Wizard should generate a **BookShopContext** class in our Data project. The idea behind the context is that it **manages the connection to the database** for us and reveals all data as **DbSet<T>** - generic collections which represent the data stored in our database. Thanks to it, we can easily work on that data with standard CRUD operations - just like we work with any C# collections!

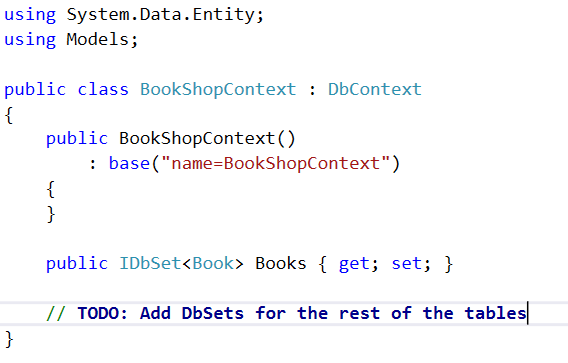
Delete the auto-generated comments and let's start writing.



Right-click **References** and add a reference to our previous project, so we can use the models.

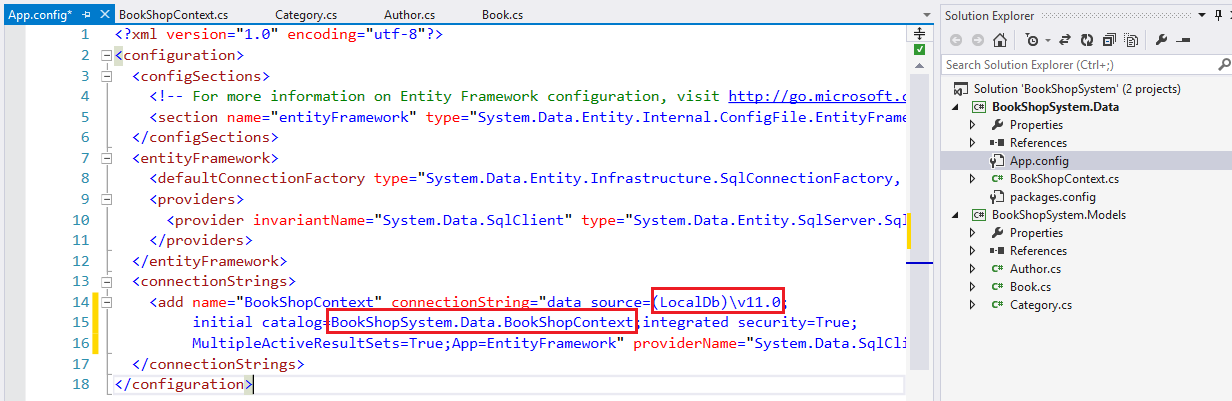


In the **BookShopContext** class, write in all necessary tables (as shown below) as **IDbSet<T> properties**, where **T** is the respective table. Make those properties public so we can access them outside the current project.



After that's done, there's one final thing left. In the **App.config** file there is a **<connectionStrings/>** tag which keeps information about the connection.

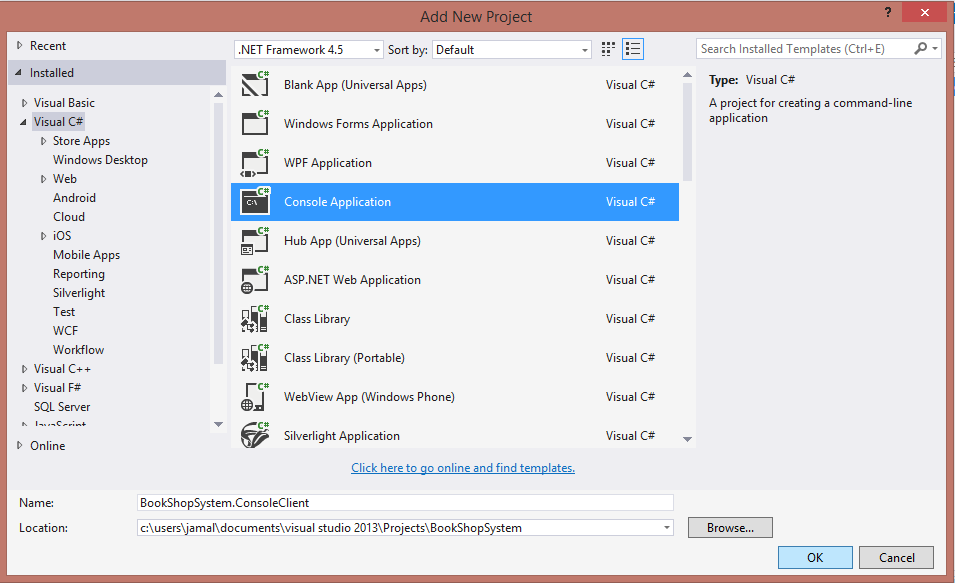
* The **name** attribute should correspond to the **connection string** in the **Context** **constructor**.
* Data source holds the target database (if you're using SQL Express it should be **.** or **localhost**. Otherwise, leave it **(LocalDb)\v11.0**.
* Initial catalog holds the name of the database - you can trim it down to **BookShopSystem** only.



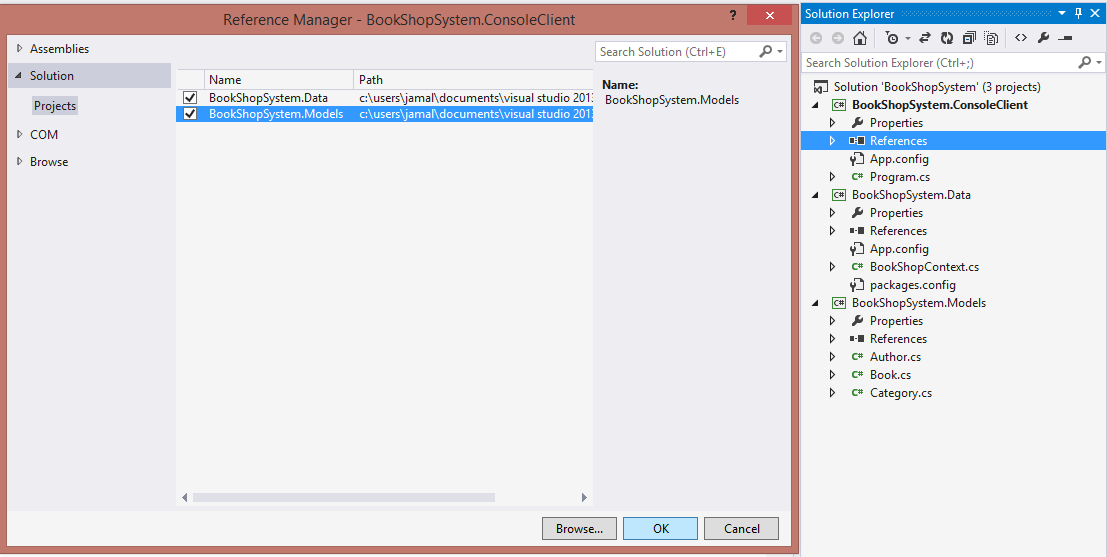
### Step 3 - Console Client

We now have our **models** (classes which will be mapped to database tables) and our **data layer.** It's time we write the console client and start Entity Framework for the first time.

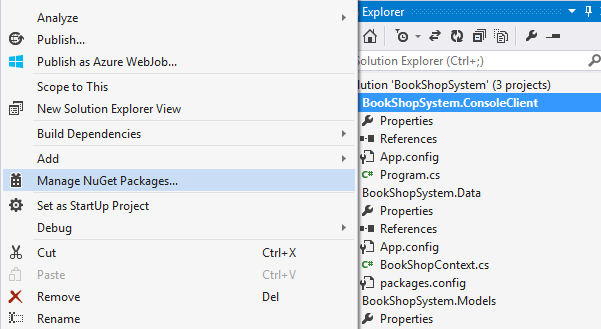
Create a new console application and name it **BookShopSystem.ConsoleClient**.



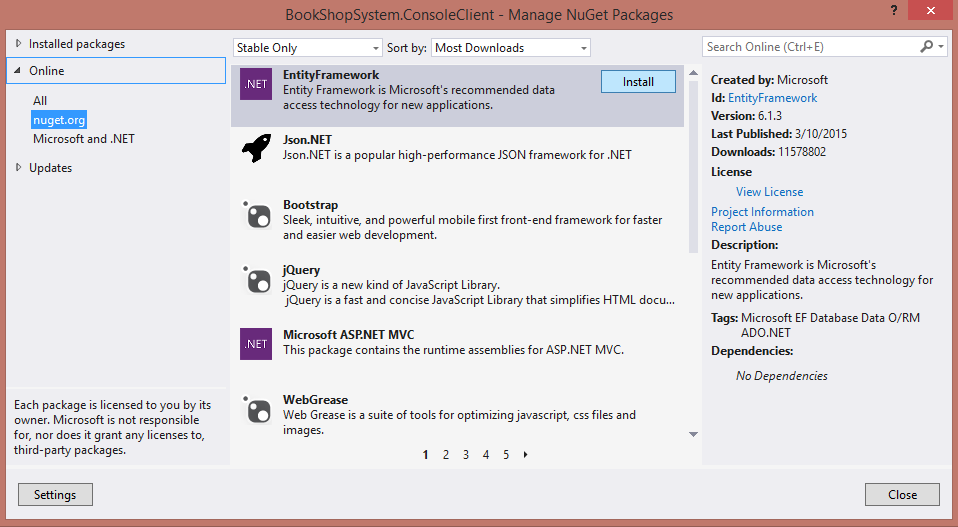
Reference the **Data** (so we can use the **BookShopContext**) and **Models** (so we can use the models).



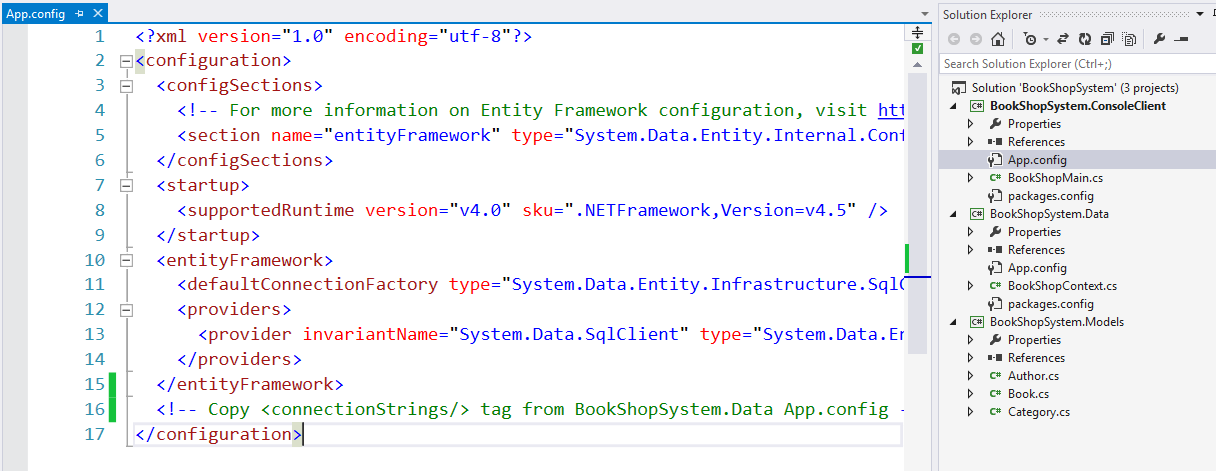
In order to use the **BookShopContext** outside the Data project, we need to **install Entity Framework** to this project as well.



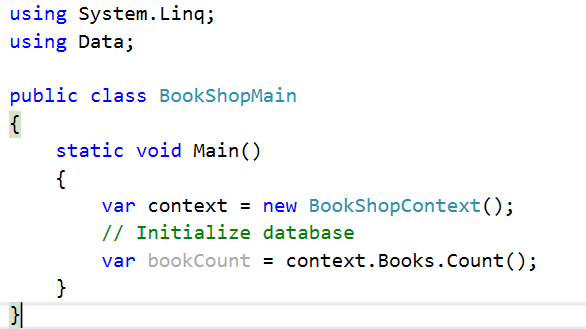
Search for **Entity Framework** in the **Nuget Package Manager** and install it.



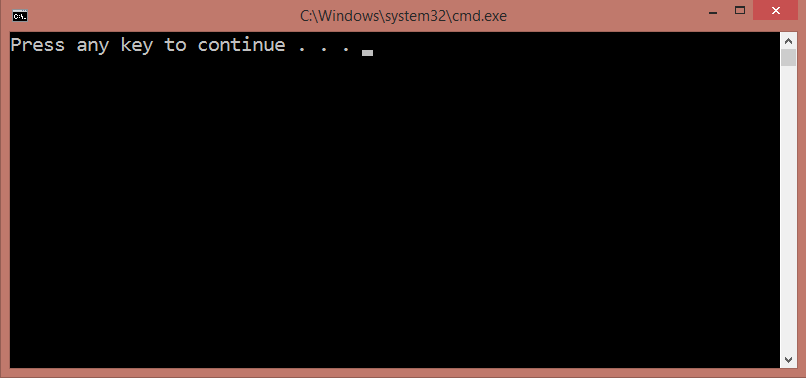
Last but not least, we need to edit the **ConsoleClient** **App.config** file and insert the **connection** **string** so it knows which database to use. We already configured that in the **Data App.config** file, so just go there and copy the **<connectionStrings/>** tag and paste it here.



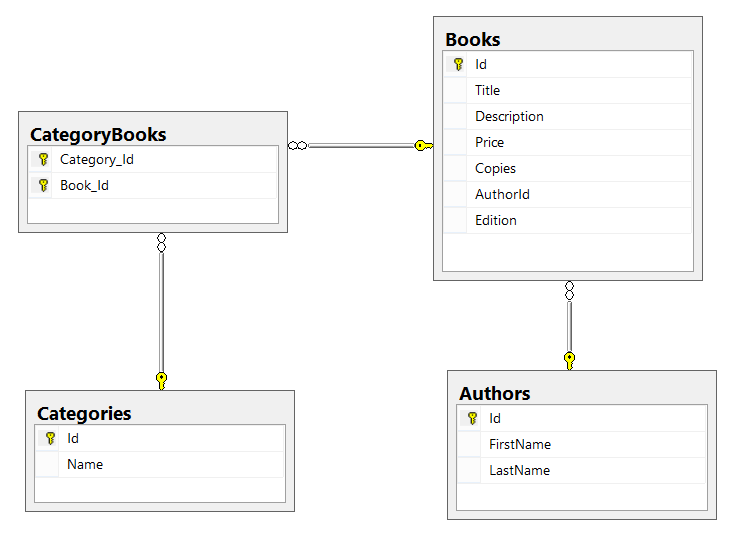
Finally, in order for Entity Framework to **create the database** for us, we need to execute any action through the **context**.



The application should finish without any exceptions.

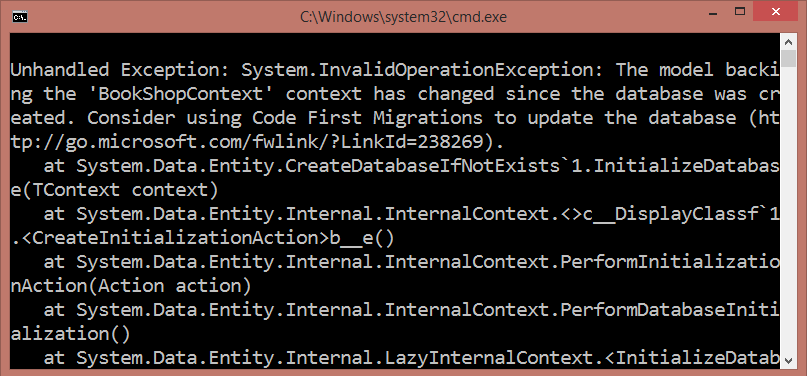


The generated relational **diagram** should be as follows:



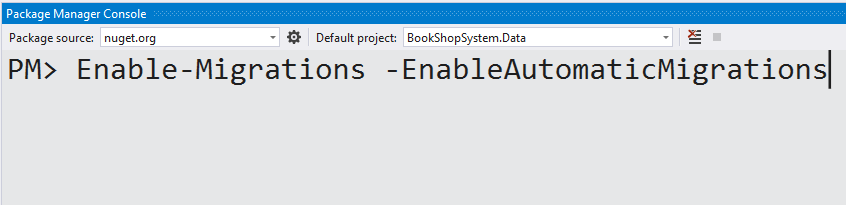
### Step 4 - Make Changes to the Local Model

Let's add a new property to the **Book** class - **AgeRestriction** (Minor, Teen or Adult). Run the program again. We expect EF to update the dabase according to the changes made to the models. However…

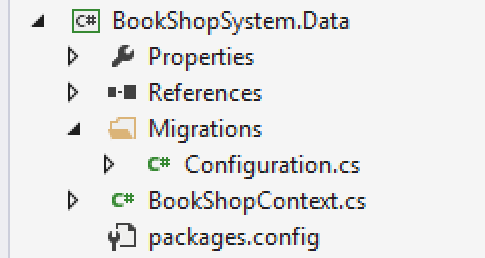


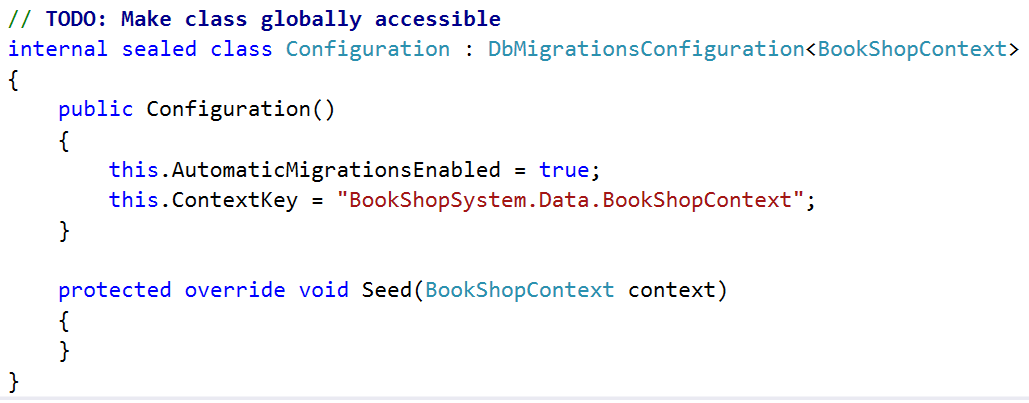
We get an exception and we are told to use **Code First Migrations**. **Migrations** update database according to the model by using a **strategy**. We need to configure Entity Framework to use a **migration strategy** (i.e. tell it what to do each time it **connects** to the database for the **first time**).

First, we need to **enable automatic migrations**. Open the Package Manager Console and write the following command:



This will create a **Configuration.cs** class which enables automatic migrations and has a **Seed()** method.

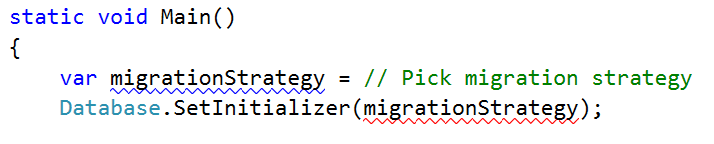




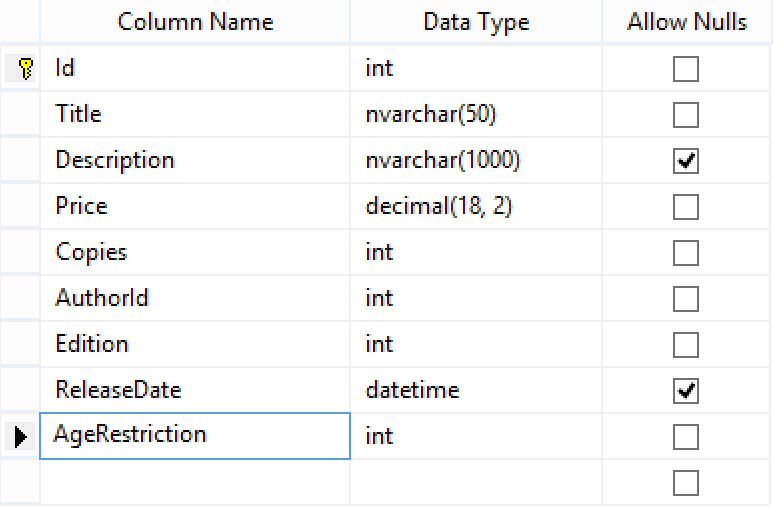
Finally, we need to the EF to apply a specific strategy to this configuration. The most used migrations are as follows:

* **DropCreateDatabaseIfModelChanges** - We lose all the data when the model changes
* **DropCreateDatabaseAlways** - Recreate the database each time we start the program
* **MigrateDatabaseToLatestVersion** - Updates the database in accordance to the changes we've made to the model

Pick the most suitible migration strategy and initialize it as shown below in the **Main()** method.



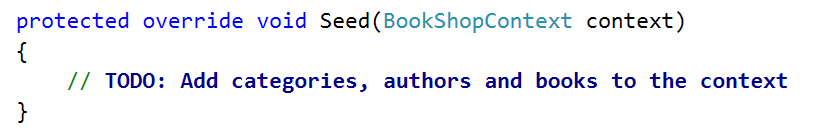
Once done, start the program again and no exception should occur. The design of the Books table should have a new column added.



### Step 5 - Seed Data into the Database

We have our database up and running. However, there is no data to work with. Let's seed some!

Go to the **Seed()** method of the **Configuration.cs** class we just created. That method will be called each time we start our application.



Use the provided **files** (**categories.txt**, **authors.txt**, **books.txt**) and import the data from them.

|  |
| --- |
| **Importing Books from File** |
| using (var reader = new StreamReader("books.txt"))  {  var line = reader.ReadLine();  line = reader.ReadLine();  while (line != null)  {  var data = line.Split(new [] {' '}, 6);  var authorIndex = random.Next(0, authors.Count);  var author = authors[authorIndex];  var edition = (EditionType) int.Parse(data[0]);  var releaseDate = DateTime.ParseExact(data[1], "d/M/yyyy", CultureInfo.InvariantCulture);  var copies = int.Parse(data[2]);  var price = decimal.Parse(data[3]);  var ageRestriction = (AgeRestriction) int.Parse(data[4]);  var title = data[5];  context.Books.Add(new Book()  {  Author = author,  Edition = edition,  ReleaseDate = releaseDate,  Copies = copies,  Price = price,  AgeRestriction = ageRestriction,  Title = title  });  line = reader.ReadLine();  }  } |

Randomly add categories to every book.

Only seed the database if it is empty.

### Step 6 - Write LINQ Queries

Now, let's leave something for you. Write the following queries with LINQ:

1. Get all **books** after the **year 2000**. Select only their **titles**.
2. Get all **authors** with at least **one book with release date before 1990**. Select their **first name** and **last name**.
3. Get all **authors**, ordered by the **number of their books** (descending). Select their **first name**, **last name** and **book count**.
4. Get all **books** from author **George Powell**, ordered by their **release date** (descending), then by **book title** (ascending). Select the book's **title**, **release** **date** and **copies**.
5. Get the most recent books by categories. The **categories** should be ordered by **total** **book count**. Only take the **top 3** most recent books from each category - ordered by **date** (descending), then by **title** (ascending). Select the **category name**, **total book count** and for each **book** - its **title** and **release date**.

**Note**: Books may appear in several categories.

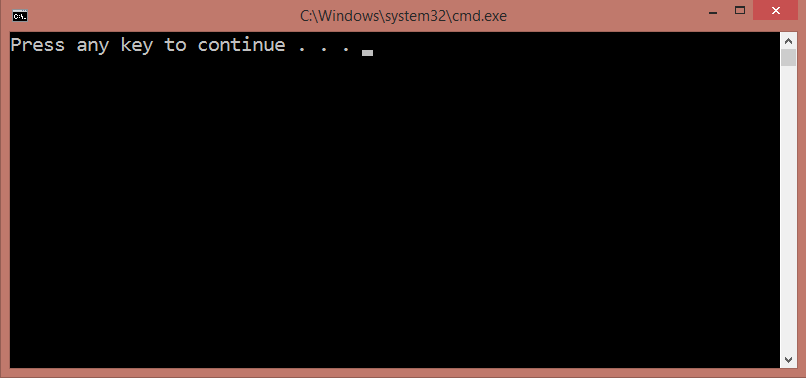
|  |
| --- |
| **Sample output** |
| --Mystery: 58 books  Brandy of the Damned (2015)  The Parliament of Man (2014)  Look Homeward (2014)  --Science Fiction: 50 books  Alien CornA (play) (2014)  How Sleep the Brave (2014)  Look Homeward (2014)  --Crime: 45 books  Look Homeward (2014)  The Doors of Perception (2013)  A Monstrous Regiment of Women (2013)  --Romance: 38 books  Alien CornA (play) (2014)  Look Homeward (2014)  The Wealth of Nations (2013)  --Fiction: 37 books  Great Work of Time (2014)  A Many-Splendoured Thing (2012)  The Last Temptation (2012)  --Thriller: 33 books  This Side of Paradise (2013)  No Highway (2013)  Fair Stood the Wind for France (2013)  --Historical: 33 books  The Wealth of Nations (2013)  Jesting Pilate (2011)  The Glory and the Dream (2011) |

### Step 7 - Related Books

Let's say at one point we decide that **books** should have **related books** - i.e. a book has many related books and each related book has related books as well.

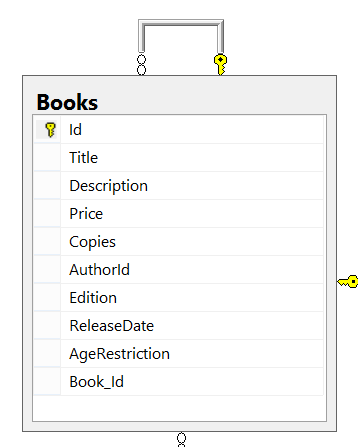
Go to the **Book** class and add a **navigational property RelatedBooks**. Make sure you instantiate it in the constructor.

Start the program again.



No errors. The migration has most likely been successful and we should see a many-to-many self-reference in the database schema.

However, that's not the case - the migration is successful but a book can have only 1 related book.

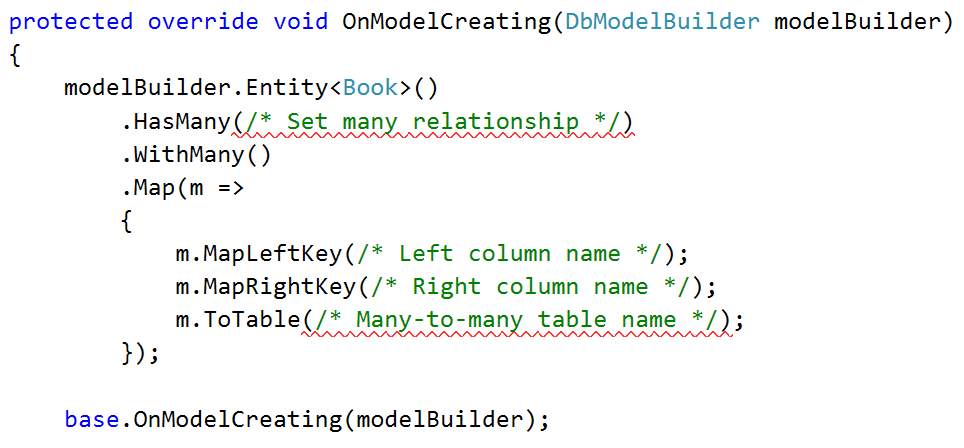


This is one of those cases where Entity Framework does not correctly update the DB schema after changes have been made to the model.

In such events, we need to manually configure the relationship with the so-called **Model Builder**. Go to the **BookShopContext** class and override the **OnModelCreating()** method.

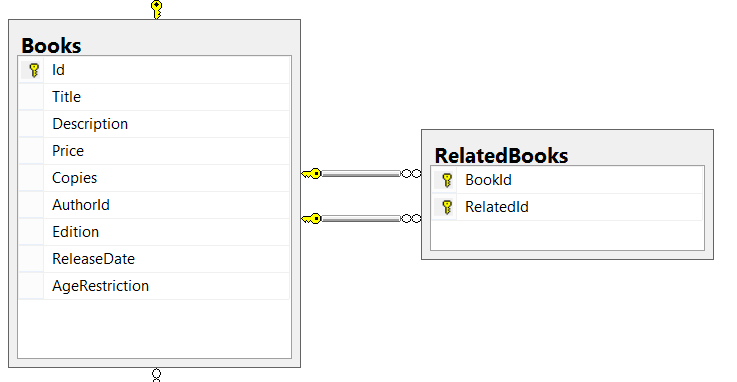
This method is called only once when entity framework starts for the first time (like **Seed()**). It receives a **DbModelBuilder** as argument. We will use that model builder to configure the desired relationship.

The Model Builder (also **fluid API**) will allow us to tell EF that a **Book** has **many books** (the related books) and those books have many as well. Then we will map that relationship to a **junction** (**many-to-many**) table where we manually set the **left key** (book id) and **right key** (related book id). Finally, we **name the table**.



At the end we call the **base** implementation of the method (it contains native EF configurations, so it's best we call it at the end).

Restart the program and if no exception is thrown, the new DB schema should now look as follows:



And finally, let's test the **RelatedBooks** functionality it. **Query 3 books** from the database and set them as **related**.

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| var books = context.Books  .Take(3)  .ToList();  books[0].RelatedBooks.Add(books[1]);  books[1].RelatedBooks.Add(books[0]);  books[0].RelatedBooks.Add(books[2]);  books[2].RelatedBooks.Add(books[0]);  context.SaveChanges();  // **TODO: Query the first three books**  // and get their names and their related book names  foreach (var book in booksFromQuery)  {  Console.WriteLine("--{0}", book.Title);  foreach (var relatedBook in book.RelatedBooks)  {  Console.WriteLine(relatedBook);  }  } | --Absalom  A che punto A" la notte  After Many a Summer Dies the Swan  --A che punto A" la notte  Absalom  --After Many a Summer Dies the Swan  Absalom |