# Exercise: EF Code First - Book Shop Part II

This document defines the **exercise assignments** for the ["Databases Advanced – Entity Framework" course @ Software University.](https://softuni.bg/trainings/1442/databases-advanced-entity-framework-october-2016)

## Create a Database for Student System using Code First

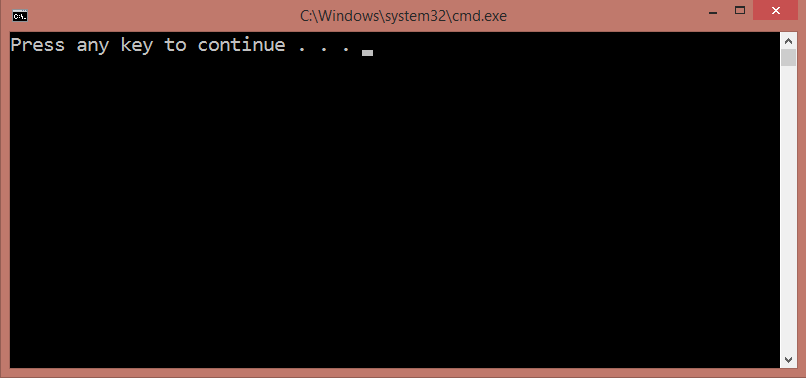
Now we shall continue from the point we have reached in Part I of the Book Shop Code First Exercise.

### 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.

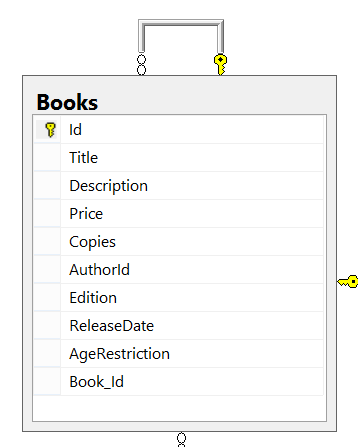
Go to the **Book** class and add a **navigational property RelatedBooks**. Make sure you instantiate it in the constructor.

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.

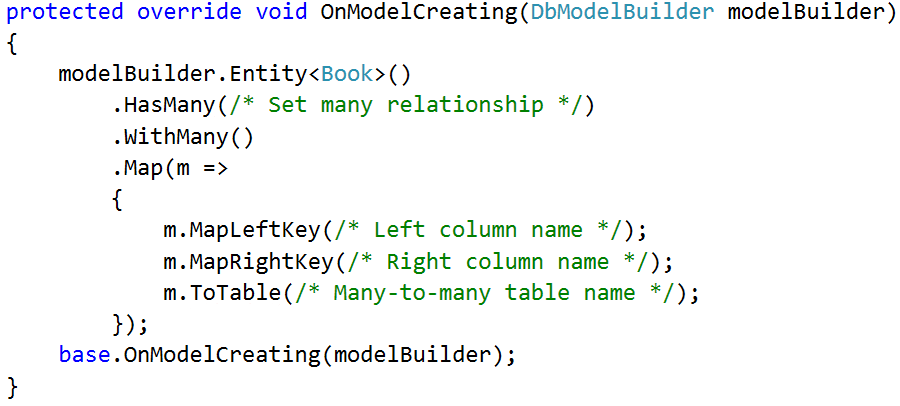


This is one of those cases where Entity Framework does not correctly update the DB schema after changes have been made to the model.

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.

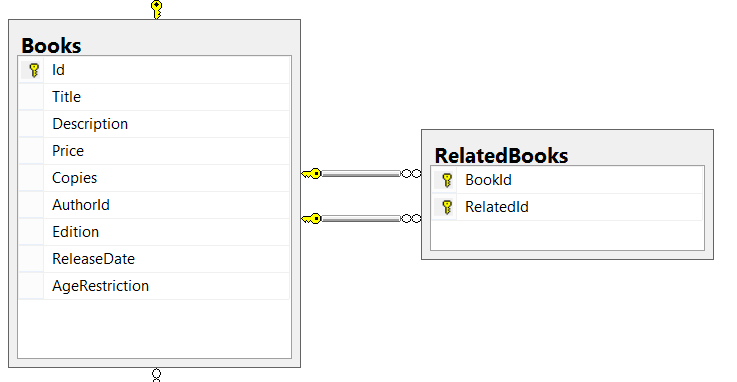
This method is called only once when entity framework starts for the first time (like **Seed()**). It receives a **DbModelBuilder** as argument. We will use that model builder to configure the desired relationship.

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 to 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 |