# Exercise: Spring Data Intro

This exercise is part of the [“Databases Frameworks” course @ SoftUni](https://softuni.bg/trainings/1635/databases-frameworks-hibernate-and-spring-data-june-2017).

## Bookshop System

Create database for **Bookshop System**. A bookshop keeps **books**. A book can have one **author** (for the sake of simplicity) and many **categories**. And each category can be placed on many books. 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), age restriction (**MINOR, TEEN or ADULT**)
* **Author** - id, first name (optional) and last name
* **Category** - id, name

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

The final schema of the database should look like that:



## Seed Data into the Database

Create **seedDatabase()** method in the ConsoleRunner **class**. That method will fill records in the database.

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

|  |
| --- |
| **Importing Books from File** |
| *//****TODO seed Authors from file authors.txt*** *//****TODO seed categories from file categories.txt*** BufferedReader booksReader = **new** BufferedReader(**new** FileReader(**"books.txt"**)); String line = booksReader.readLine(); **while**((line = booksReader.readLine()) != **null**){  String[] data = line.split(**"\\s+"**);   **int** authorIndex = random.nextInt(authors.size());  Author author = authors.get(authorIndex);  EditionType editionType = EditionType.*values*()[Integer.*parseInt*(data[0])];  SimpleDateFormat formatter = **new** SimpleDateFormat(**"d/M/yyyy"**);  Date releaseDate = formatter.parse(data[1]);  **int** copies = Integer.*parseInt*(data[2]);  BigDecimal price = **new** BigDecimal(data[3]);  AgeRestriction ageRestriction = AgeRestriction.*values*()[Integer.*parseInt*(data[4])];  StringBuilder titleBuilder = **new** StringBuilder();  **for** (**int** i = 5; i < data.**length**; i++) {  titleBuilder.append(data[i]).append(**" "**);  }  titleBuilder.delete(titleBuilder.lastIndexOf(**" "**), titleBuilder.lastIndexOf(**" "**));  String title = titleBuilder.toString();   Book book = **new** Book();  book.setAuthor(author);  book.setEditionType(editionType);  book.setReleaseDate(releaseDate);  book.setCopies(copies);  book.setPrice(price);  book.setAgeRestriction(ageRestriction);  book.setTitle(title);  *//****TODO add random categories for current book* bookService**.save(book); } |

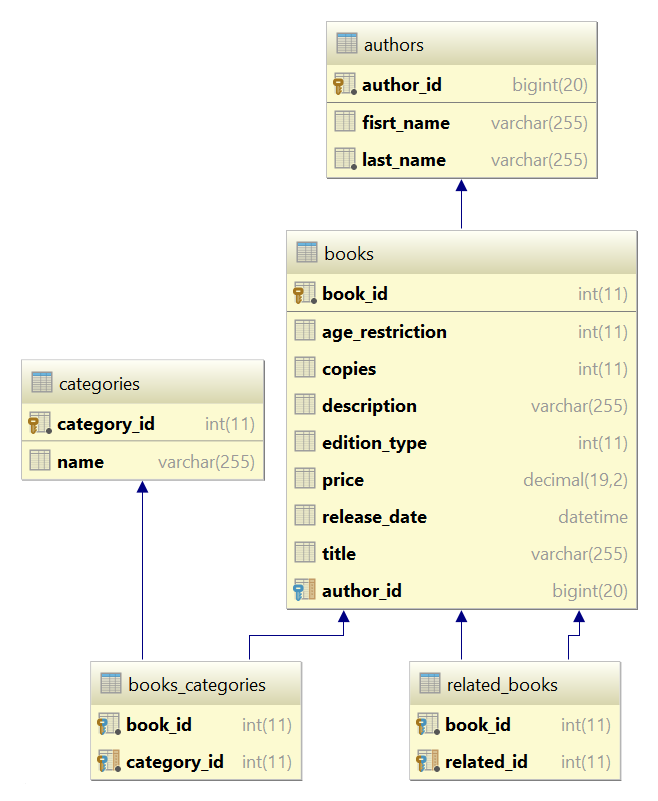
## Write Queries

Write the following programs that:

1. Get all **books** after the **year 2000**. Print only their **titles**.
2. Get all **authors** with at least **one book with release date before 1990**. Print their **first name** and **last name**.
3. Get all **authors**, ordered by the **number of their books** (descending). Print 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). Print the book's **title**, **release** **date** and **copies**.

## 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.

The new database schema should now look as follows:

**Get 3 books** from the database and set them as **related**.

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| List<Book> books = (List<Book>) **bookService**.findBooks(); List<Book> threeBooks = books.subList(0, 3);  threeBooks.get(0).getRelatedBooks().add(threeBooks.get(1)); threeBooks.get(1).getRelatedBooks().add(threeBooks.get(0)); threeBooks.get(0).getRelatedBooks().add(threeBooks.get(2)); threeBooks.get(2).getRelatedBooks().add(threeBooks.get(0));  *//save related books to the database* **for** (Book book : threeBooks) {  **bookService**.save(book); }  **for** (Book book : threeBooks) {  System.***out***.printf(**"--%s\n"**, book.getTitle());  **for** (Book relatedBook : book.getRelatedBooks()) {  System.***out***.println(relatedBook.getTitle());  } } | --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 |

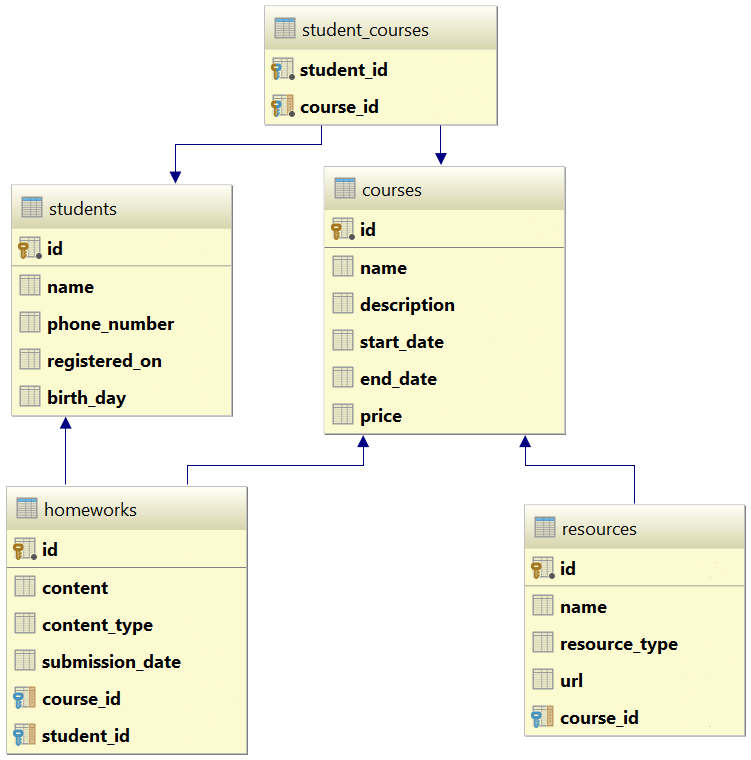
## Student System

Your task is to create a database for the **Student System**. Model the following tables:

* **Students**: id, name, phone number (optional), registration date, birthday (optional)
* **Courses**: id, name, description (optional), start date, end date, price
* **Resources**: id, name, type of resource (video / presentation / document / other), URL
* **Homework**: id, content, content-type (e.g. application/pdf or application/zip), submission date

Table relations:

* **Students** can be in **many course**s
* **Courses** can have **many students**
* **Courses** can have **many resources**
* **One course** can have **many homework submissions**
* **Homework submissions** have a **student**



## Seed Some Data in the Database

Write a **seed method** that fills the database with sample data (randomly generated).

Fill a few **students**, **courses**, **resources** and **homework submissions**.

Run the application several times to ensure that it seeds sample data **only** **once**.

## Working with the Database

Write a console application that works with the EF data layer and performs the following CRUD operations:

1. Lists **all students** and their **homework submissions**. Select only their **names** and for each homework - **content** and **content-type**.
2. List **all courses** with their corresponding **resources**. Select the **course name** and **description** and everything for each **resource**. Order the courses by start date (ascending), then by end date (descending).
3. List **all courses** with **more than 5 resources**. Order them by **resources count** (descending), then by **start date** (descending). Select only the **course name** and the **resource count**.
4. List all **courses** which were active on a **given date** (choose the date depending on the data seeded to ensure there are results), and for each course count the **number of students enrolled**.   
   Select the **course name**, **start** and **end date**, **course duration** (difference between end and start date) and **number of students enrolled**. Order the results by the **number of students** enrolled (in descending order), then by **duration** (descending).
5. For each student, calculate the **number of courses** he/she has enrolled in, the **total price** of these courses and the **average price per course** for the student.  
   Select the **student name**, **number of courses**, **total price** and **average price**. Order the results by **total price** (descending), then by **number of courses** (descending) and then by the **student's name** (ascending).

## Resource Licenses

Resources should now have many **licenses**. A **license** should have an **Id** and **Name**.

Make sure no data is lost after the update.

## Create User

Your task is to create table **Users**. The table should contain the following fields:

* **id** – Primary Key (number in range [1, 231-1])
* **username** – Text with length between 4 and 30 symbols. Required.
* **password** – Required field. Text with length between 6 and 50 symbols. Should contain at least:
  + 1 lowercase letter
  + 1 uppercase letter
  + 1 digit
  + 1 special symbol (!, @, #, $, %, ^, &, \*, (, ), \_, +, <, >, ?)
* **email** – Required field. Text that is considered to be in format **<user>@<host>** where:
  + **<user>** is a sequence of letters and digits, where '**.**', '**-**' and '**\_**' can appear between them (they cannot appear at the beginning or at the end of the sequence).
  + **<host>** is a sequence of at least two words, separated by dots '**.**' (dots cannot appear at the beginning or at the end of the sequence)
* **profile\_picture** – Image file (.jpeg or .png) with size maximum of 1MB
* **registered\_on** – Date and time of user registration
* **last\_time\_logged\_in** – Date and time of the last time the user logged in
* **age** – number in range [1, 120]
* **is\_deleted** – Shows whether the user is deleted or not

## User Towns

Now the user should have **born town** and **currently living** in town. The town has **name** and **country** where is he placed. Migrate the database with the new schema of the table and make sure no data is lost when updating.

## User Names

Again, it’s modification time this time not so big. Add 2 new properties to the user **first name** and **last name**. Also, add one more property **fullName** that would return the concatenation of first and last name separated by a single space. That property must be generated only when we need it (there is no need to keep it in the database). Migrate the database with the new schema of the table and make sure no data is lost when updating.

## Friends

Let’s say that the **user can have many friends** that would be again other users (or in other words **many to many self-relationship**).

Make the necessary changes using Code First Migrations. Make sure no data is lost after the update.

## Albums

Previously 1 user was able to upload only 1 picture (just his/her profile picture). Now each user is capable of creating **personal albums**. Each album has **name, background color** and information whether **is public or not**. Each picture has **title**, **caption** and **path** on the file system. An album can contain many pictures and one picture can be present in many albums. Each user can have many albums but an album can have only one owner user.

Make the necessary changes using Code First Migrations. Make sure no data is lost after the update.

## \*Email Annotation

Make a validation annotation **@Email** that can be used on string fields. The property should check if the value of the property is valid. One email is valid if in format **<user>@<host>** where:

* **<user>** is a sequence of letters and digits, where '**.**', '**-**' and '**\_**' can appear between them. Examples of valid users: "**stephan**", "**mike03**", "**s.johnson**", "**st\_steward**", "**softuni-bulgaria**", "**12345**". Examples of invalid users: ''**--123**", ".....", "**nakov\_-**", "**\_steve**", "**.info**".
* **<host>** is a sequence of at least two words, separated by dots '**.**'. Each word is sequence of letters and can have hyphens '**-**' between the letters. Examples of hosts: "**softuni.bg**", "**software-university.com**", "**intoprogramming.info**", "**mail.softuni.org**". Examples of invalid hosts: "**helloworld**", "**.unknown.soft.**", "**invalid-host-**", "**invalid-**".
* Examples of **valid emails**: info@softuni-bulgaria.org, kiki@hotmail.co.uk, no-reply@github.com, s.peterson@mail.uu.net, info-bg@software-university.software.academy.
* Examples of **invalid emails**: --123@gmail.com, …@mail.bg, .info@info.info, \_steve@yahoo.cn, mike@helloworld, mike@.unknown.soft., s.johnson@invalid-.

emailannot

Use that annotation on the previous problems to validate any fields containing e-mail address.

### Hint

Use [Hibernate Validator](http://hibernate.org/validator/).

## \*Password Annotation

Make validation annotation **@Password** that can be used to validate string fields. The property should check if the value of the field is valid. In the constructor, the password should receive minimum and maximum length of the password. As optional parameters, we should be able to provide whether the password should contain lowercase letter, uppercase letter, digit or special symbol.

|  |  |
| --- | --- |
| C:\Users\Valio\AppData\Local\Microsoft\Windows\INetCacheContent.Word\annotattribute1.png | C:\Users\Valio\AppData\Local\Microsoft\Windows\INetCacheContent.Word\annotattribute.png |

Use that annotation on the previous problems to validate any fields containing password.

## Get Users by Email Provider

Write program that print all usernames and emails of users by given email provider.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| gmail.com | pesho123 pesho@gmail.com  vanko1 vanko1@gmail.com  goshko\_n00b gn00b@gmail.com |
| yahoo.co.uk | penbo pen@yahoo.co.uk  catLady stepheny.p@yahoo.co.uk |
| abv.bg | No users found with email domain abv.bg |

## Count Users with Bigger Pictures

Write a program that count the users with pictures bigger than given width.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 120 | 4 users have profile pictures wider than 120 pixels |
| 921 | 1 user has profile picture wider than 921 pixels |
| 999 | No users have profile picture wider than 999 pixels |

## Remove Inactive Users

Write a program that set is\_deleted field to true for all users that has not been logged in after given date. Print the number of user that has been set as deleted. Then delete all users that have been marked for removal.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 12 Oct 2004 | 10 users have been deleted |
| 10 Jul 2015 | 1 user has been deleted |
| 01 Nov 2016 | No users have been deleted |