Exercises: Hibernate Code First

This document defines the **exercise assignments** for the "Databases Advanced – Hibernate" course @ Software University.

1. Gringotts Database

Your task is to create table **wizard_deposits** using the Code First approach. The table should contain the following fields:

- **id** – Primary Key (number in range [1, 2^{31}-1].
- **first_name** – Text field with max length of 50 symbols.
- **last_name** - Text field with max length of 60 symbols. Required
- **notes** – Text field with max length of 1000 symbols
- **age** – Non-negative number. Required
- **magic_wand_creator** – Text field with max length of 100 symbols
- **magic_wand_size** – Number in range [1, 2^{15}-1]
- **deposit_group** - Text field with max length of 20 symbols
- **deposit_start_date** – Date and time field
- **deposit_amount** – Floating point number field
- **deposit_interest** - Floating point number field
- **deposit_charge** - Floating point number field
- **deposit_expiration_date** – Date and time field
- **is_deposit_expired** – Boolean field

Add several records to the database using Hibernate. Use the following example of wizard deposit creation and addition to the database.

```
WizardDeposit dumbledore = new WizardDeposit();
dumbledore.setFirstName("Albus");
dumbledore.setLastName("Dumbledore");
dumbledore.setAge(150);
dumbledore.setMagicWandCreator("Antoich Peverell");
dumbledore.setMagicWandSize(15);
Calendar calendar = Calendar.getInstance();
calendar.set(2016, 10, 20);
Date depositStart = calendar.getTime();
dumbledore.setDepositStartDate(depositStart);
calendar.set(2020, 10, 20);
Date depositEnd = calendar.getTime();
dumbledore.setDepositExpirationDate(depositEnd);
dumbledore.setDepositAmount(2000.24);
dumbledore.setDepositCharge(0.2);
dumbledore.setIsDepositExpired(false);

entityManager.persist(dumbledore);
entityManager.getTransaction().commit();
```

2. Create User

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

- **id** – Primary Key (number in range [1, 2^{31}-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

Seed some users in the database. Test if validation of the fields works as expected.

3. Hotel Database

Create database and create the following tables using the Code First approach:

- **employees** (id, first_name, last_name, title, notes)
- **customers** (account_number, first_name, last_name, phone_number, emergency_name, emergency_number, notes)
- **room_status** (room_status, notes)
- **room_types** (room_type, notes)
- **bed_types** (bed_type, notes)
- **rooms** (room_number, room_type, bed_type, rate, room_status, notes)
- **payments** (id, payment_date, account_number, first_date_occupied, last_date_occupied, total_days, amount_charged, tax_rate, tax_amount, payment_total, notes)
- **occupancies** (id, date_occupied, account_number, room_number, rate_applied, phone_charge, notes)

No relationships between tables are required. Make appropriate validations on different fields.

4. Sales Database

Create database for storing data about sales using the Code First approach. The database should have the following tables:

- **product** (id, name, quantity, price)
- **customer** (id, name, email, credit_card_number)
- **store_location** (id, location_name)
- **sale** (id, product_id, customer_id, store_location_id, date)

The relationships between tables are as follows:

- Sale has one product and a product can be sold in many sales
- Sale has one customer and a customer can participate in many sales
- Sale has one store location and one store location can have many sales

Hint

You can use the following format to design your model classes

- **Product**
  - int id
  - String name
  - Double quantity
  - BigDecimal price
  - Set<Sale> salesOfProduct
- **Customer**
  - int id
  - String name
  - String email
  - String creditCardNumber
  - Set<Sale> salesForCustomer
- **StoreLocation**
  - int id
  - String locationName
  - Set<Sale> salesInStore
- **Sale**
  - int id
  - Product product
  - Customer customer
  - StoreLocation storeLocation
  - Date date

5. Hospital Database

Congrats! You were hired as a Junior Database App Developer. But before starting to work you were required to provide some documents such as fit note from your GP. So, you go to him to get it. When you tell him, what do you need and what kind of job you are about to start. He told you that he was just looking for someone to make a software to help him managing and keeping data about his patients. He offered you to give you the fit note for free if you help him. You decided that's a great opportunity to save 20 leva and go out tonight with friends and also you would expand your portfolio with 1 project.

Your task is to design a database using the Code First approach. The GP needs to keep information about his patients. Each patient has first name, last name, address, email, date of birth, picture, information whether he has medical insurance or not and should keep history about all his visitations, diagnoses and prescribed medicaments. Each visitation has date and comments. Each diagnose has name and comments for it. Each medicament has name. Make sure all data is validated before inserting in the database.

Bonus Task

Make console based user interface so the doctor can use easily the database.

**Bonus Task

The console based user interface is a good start point but almost nobody use them nowadays (except BDZ Passenger Services). So, make a nice good looking graphical user interface for the program.

6. User Towns

It's time to modify the database we created in the 2nd task. 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.

7. 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.

8. Hospital Database Modification

Your GP bragged around in the hospital about the cool software you made for him. And now the hospital administration wants to modify your program so they can use it too. They want to store information about the doctors (name and specialty). Each doctor can perform many visitations and in each visitation, can be performed by only one doctor. Make the necessary changes in the database to satisfy the new needs of the hospital administration. When migrating to the new database schema make sure no data is lost. If you made some user interface (graphical or not), make changes in it be more adequate for the changes.

**Bonus Task

Make authentication system for doctors. Each doctor should be able to log in with his email and password. When the doctor is logged in he can see only information related about himself (his visitations, the patients he examined, etc.).

9. 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-.

```
@Email
private String email;
```

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

10. 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.

```
@Password(minLength = 6,
        maxLength = 40,
        containsDigit = true,
        containsLowercase = true)
private String password;
```

```
@Password(minLength = 4,
        maxLength = 30,
        containsDigit = true,
        containsLowercase = true,
        containsUppercase = true,
        containsSpecialSymbol = true,
        message = "Invalid password")
private String password;
```

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

11. 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

12. 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

13. 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