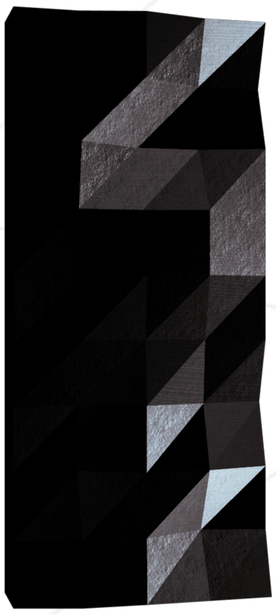

Java - Module 05

SQL/JDBC

Summary:
In this module you will use the key mechanisms to work with PostgreSQL DBMS via JDBC.

Version: 1.00

Contents

Chapter I

General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- You mus use the latest LTS version of Java. Make sure that compiler and interpreter of this version are installed on your machine.
- You must use both JVM and GraalVM to run your code.
- You can use IDE to write and debug the source code (we recommend IntelliJ Idea).
- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each exercise, make sure you follow the generally accepted Oracle standards
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output
- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter II

Rules of the Day

- Use the PostgreSQL DBMS in all tasks.
- Connect the up-to-date version of JDBC driver.
- To interact with the database, you may use classes and interfaces of the java.sql package (implementations of corresponding interfaces will be automatically included from the archive containing the driver).

Chapter III

Exercise 00: Tables and Entities

	Exercise 00
Tables and Entities	
Turn-in directory : *ex*00/	
Files to turn in : `Chat-folder`	
Allowed functions : `n/a`	

Today we will be implementing Chat functionality. In this chat, user can create or choose an existing chatroom. Each chatroom can have several users exchanging messages.

Key domain models which both SQL tables and Java classes must be implemented for are:

```
User
 ├── User ID
 ├── Login
 ├── Password
 ├── List of created rooms
 └── List of chatrooms where a user socializes
Chatroom
 ├── Chatroom id
 ├── Chatroom name
 ├── Chatroom owner
 └── List of messages in a chatroom
Message
 ├── Message id
 ├── Message author
 ├── Message room
 ├── Message text
 └── Message date/time
```

Create schema.sql file where you will describe `CREATE TABLE` operations to create tables for the project.

You should also create data.sql file with text data `INSERTS` (at least five in each table).

It is important to meet the following requirement! Let's assume that Course entity has a one-to-many relationship with Lesson entity. Their object-oriented relation should then look as follows:

```
class Course {
    private Long id;
    private List<Lesson> lessons;// there are numerous lessons in the course
    ...
}
class Lesson {
    private Long id;
    private Course course; // the lesson contains a course it is linked to
    ...
}
```

Additional requirements:

- To implement relational links, use one-to-many and many-to-many link types.
- Identifiers should be numeric.
- Identifiers shall be generated by `DBMS`.
- `equals()`, `hashCode()` and `toString()` shall be redefined correctly inside Java classes.

Exercise project structure:

```
Chat
|__src
|   |__main
|       |__java
|       |   |__fr.42.chat
|       |       |__models - domain knowledge models
|       |__resources
|           |__schema.sql
|           |__data.sql
|__pom.xml
```

Chapter IV

Exercise 01: Read/Find

	Exercise 01
Read/Find	
Turn-in directory : *ex*01/	
Files to turn in : `Chat-folder`	
Allowed functions : `n/a`	

`Data Access Object` (`DAO`, Repository) is a popular design pattern that allows to separate key business logic from data handling logic in an application.

Let's assume that we have an interface called `CoursesRepository` which provides access to course lessons. This interface may look as follows:

```
public interface CoursesRepository {
    Optional<Course> findById(Long id);
    void delete(Course course);
    void save(Course course);
    void update(Course course);
    List<Course> findAll();
}
```

You need to implement `MessagesRepository` with a `SINGLE Optional<Message> findById(Long id)` method and its `MessagesRepositoryJdbcImpl` implementation. This method shall return a `Message` object where author and chatroom will be specified. In turn, there is `NO NEED` to enter subentities (list of chatrooms, chatroom creator, etc.) for the author and the chatroom.

The implemented code must be tested in Program.java class. Example of the program operation is as follows (the output may differ):

```
   $> java Program
   Enter a message ID
-> 5
   Message : {
   id=5,
   author={id=7,login="user",password="user",createdRooms=null,rooms=null},
   room={id=8,name="room",creator=null,messages=null},
   text="message",
   dateTime=01/01/01 15:69
   }
   $>
```

Exercise project structure:

```
Chat
 |__src
 |   |__main
 |       |__java
 |       |   |__fr.42.chat
 |       |       |__models - domain knowledge models
 |       |       |__repositories - repositories
 |       |       |__app
 |       |           |__Program.java
 |       |__resources
 |           |__schema.sql
 |           |__data.sql
 |__pom.xml
```

- `MessagesRepositoryJdbcImpl` shall accept `DataSource` interface of java.sql package as a constructor parameter.
- For `DataSource` implementation, use `HikariCP` library—a pool of connections to the database which considerably expedite the use of storage.

Chapter V

Exercise 02: Create/Save

	Exercise 02
Create/Save	
Turn-in directory : *ex*02/	
Files to turn in : `Chat-folder`	
Allowed functions : `n/a`	

Now you need to implement `save(Message message)` method for `MessagesRepository`.

Thus, we need to define the following subentities for the entity we are saving :
a message author and a chatroom. It is also important to assign IDs that exist in the database to chatroom and author.

Example of save method use:

```
public static void main(String args[]) {
    ...
User creator = new User(7L, "user", "user", new ArrayList(), new ArrayList());
User author = creator;
Room room = new Room(8L, "room", creator, new ArrayList());
Message message = new Message(null, author, room, "Hello!", LocalDateTime.now());
MessagesRepository messagesRepository = new MessagesRepositoryJdbcImpl(...);
messagesRepository.save(message);
System.out.println(message.getId()); // ex. id == 11
}
```

- So, save method shall assign ID value for the incoming model after saving data in the database.
- If author and room have no ID existing in the database assigned, or these IDs are null, throw `Runtimeexception NotSavedSubEntityException` (implement this exception on your own).
- Test the implemented code in Program.java class.

Chapter VI

Exercise 03: Update

	Exercise 03
Update	
Turn-in directory : *ex*03/	
Files to turn in : `Chat-folder`	
Allowed functions : `n/a`	

Now we need to implement update method in `MessageRepository`. This method shall fully update an existing entity in the database. If a new value of a field in an entity being updated is null, this value shall be saved in the database.

An example of update method use:

```
public static void main(String args[]) {
    MessagesRepository messagesRepository = new MessagesRepositoryJdbcImpl(...);
    Optional<Message> messageOptional = messagesRepository.findById(11);
    if (messageOptinal.isPresent()) {
        Message message = messageOptional.get();
        message.setText("Bye");
        message.setDateTime(null);
        messagesRepository.update(message);
    }
    ...
}
```

- In this example, the value of the column storing the message text will be altered, whereas message time will be null.

Chapter VII

Exercise 04: Find All

	Exercise 04
Find All	
Turn-in directory : *ex*04/	
Files to turn in : `Chat-folder`	
Allowed functions : `n/a`	

Now you need to implement `UsersRepository` interface and `UsersRepositoryJdbcImpl` class using a `SINGLE List<User> findAll(int page, int size)` method.

This method shall return size—users shown in the page with page number. This `"piecewise"` data retrieval is called pagination. Thus, DBMS divides the overall set into pages each containing size entries. For example, if a set contains 20 entries with page = 3 and size = 4 , you retrieve users 12 to 15 (user and page numbering starts from 0).

The most complicated situation in converting relational links into object-oriented links happens when you retrieve a set of entities along with their subentities. In this task, each user in the resulting list shall have included dependencies—a list of chatrooms created by that user, as well as a list of chatrooms the user participates in.

Each subentity of the user `MUST` NOT include its dependencies, i.e. list of messages inside each room must be empty.

The implemented method operation should be demonstrated in Program.java.

`findAll(int page, int size) method shall be implemented by a SINGLE database query. It is not allowed to use additional SQL queries to retrieve information for each user.`

We recommend using CTE PostgreSQL.

UsersRepositoryJdbcImpl shall accept DataSource interface of java.sql package as a constructor parameter.