

ORM + JPQL

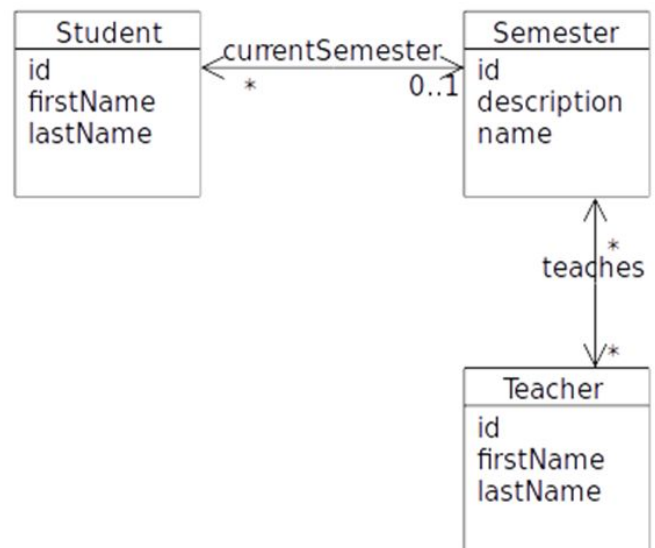
General part

- Explain the rationale behind the topic Object Relational Mapping and the Pros and Cons in using a ORM.
- Discuss how we usually have queried a relational database
- Discuss the methods we can use to query a JPA design and compare with what you explained above

Practical part

The exercise requires this [script](#) and you must set up a MySQL database as described below:

- Execute (from Workbench or whatever you prefer) the script given above and verify that the database *ExamPreparationJPQL* was created with tables and data.
- Create a new plain Maven Java Project with NetBeans to hold the code for this project.







The script created four tables (why FOUR) which simulates a very simple semester system with *students*, *teachers* and *semesters*, as sketched in this Object-model.


The script also inserted **6** students, **3** semesters and **3** teachers, and assigns relationships between them.

A) Use NetBeans to create a set of matching Entity Classes (see hints at the end). Make sure you understand what was created, and that you understand how classes and tables are related (almost a guaranteed discussion topic for the exam)

B) Investigate the generated Entity classes and observe the `NamedQueries` generated by the Wizard.

C) Create (preferable in a facade-class) Dynamic Queries (or if possible, use a named Query generated by the wizard) to solve the following problems:

1. **Find** all Students in the system
2. **Find** all Students in the System with the first name Anders
3. **Insert** a new Student into the system
4. **Assign** a new student to a semester (given the student-id and semester-id)
5.  Had the student above already been assigned to a semester he would suddenly be a member of two semesters, but still only have one reference back to the newest assigned semester. Fix this problem, preferably without losing historical information.
6. **Find** (using JPQL) all Students in the system with the last name And
7. Find (using JPQL) the total amount of all students
8.  **Find** (using JPQL) the total number of students, for a semester given the semester name as a parameter.
9.  **Find** (using JPQL) the total number of students in all semesters.
10.  **Find** (using JPQL) the teacher(s) who teaches on most semesters.

11.  Often (as in almost always) we don't want a result that matches an Entity class, but a result that matches a specific customer requirement for a specific request.

Create a new package, `mappers` and add a class `StudentInfo` to the package. Add these **public** properties to the class:


```
public String fullName;  
public long studentId;  
public String classNameThisSemester;  
public String classDescription;
```

Note: Since the Semester represents the current Semester, and there are only one (or zero) semesters for a Student, semester-info can easily be encapsulated inside the StudentInfo class.

Add a constructor that takes all the necessary arguments (create `fullName` from `firstname` and `lastname`)

Now create a method (using JPQL) with the following signature to return a list of all Students, encapsulated as `StudentInfo`'s.

```
List<StudentInfo> getStudentInfo ()
```

12.  Create a method, similar to the one above, but which returns a single `StudentInfo`, given a students id as sketched below:

```
StudentInfo getStudentInfo (long id)
```

Hints: For the last two → Read about *Result Classes* here:

<https://www.objectdb.com/java/jpa/query/jpql/select>

Hints: (How to create Entity Classes from an existing Database with NetBeans)

Start the Wizard: **EntityClasses from Database...**

On the page "Database Tables"

Select the Database Collection

Add All tables

Press Next

On the page "Entity Classes"

Rename package to entity

De-select "Generate JAXB Annotations"

Press Next

On the page "Mapping Options"

De-select "Use Column Names in Relationships"

Press Finish