# JPA- Queries 2 Workshop

## Content

In this workshop we will work with queries. We will look at a little more complex queries, for example when we want to navigate across relationships or when we are working with “many” relationships so we have to use join. These are the topics of this workshop:

- Navigating across relationships
- Working with collections: member of
- Using join

## Navigating across relationships

Here we want to find all the students for a specific tutor:

```
Query q2 =em.createQuery("select tutor.teachingGroup from Tutor as tutor
where tutor.name= 'Johan Smith'");
List<Student> studentsForJohan = q2.getResultList();
for (Student s : studentsForJohan) {
   System.out.println(s);
}
```

In the above, the Tutor class has all the information about the Student class, because we have established a @OneToMany relation from Tutor to Student class.

## Working with collections: member of

We want to see which tutor teaches a specific subject.

```
Subject programming = em.find(Subject.class, 3);
TypedQuery<Tutor> query= em.createQuery("from Tutor tutor where :subject
member of tutor.subjectsToTeach",Tutor.class);
query.setParameter("subject", programming);
List<Tutor>tutorsForProgramming = query.getResultList();
for(Tutor tutor : tutorsForProgramming) {
   System.out.println(tutor);
}
```

What we have done above, is that we got the data about the subject with id number 3 and stored it in a reference variable.
Then we created a query to get the tutors for this specified subject and we printed the result.

Run the code. This code will return the name of the teacher (teachers) for this subject.

## Using join

There are some occasions that we cannot use hql such as when we are working with ‘many’ relationships. In these situations we should use join.

If for example we want a list of all tutors who have a student who lives in ‘city 2’, because the tutor has many students, it is not easy to use hql to navigate from tutor to students.
Here we have to use *‘join’*. We have used ‘join’ before in sql. This is the same:

Remove all the query code from the main method. Add the following instead:

```
Query query2 = em.createQuery("from Tutor as tutor join tutor.teachingGroup
as student where student.address.city = 'city 2'");
List<Object[]> results = query2.getResultList();
for (Object[] item : results) {
  System.out.println(item[0] + "-------------------- "+ item[1]);
}
```

Now change the city of Jimi Hendriks to city 2 so that we have more students that have city 2 in their address.
Run the code.

## Using distinct

You get a pair of tutor/students as a result. So if a teacher has two students who live in city 2 we get the same teacher twice.

But here we are interested only in the tutor, and it is enough to get the name of the tutor only once. So we change the query to get only the tutor back.
And we add *distinct* to our sql statement to get the name of the tutor only once.

```
Query query3 = em.createQuery("select distinct tutor from Tutor as tutor
join tutor.teachingGroup as student where student.address.city = 'city
2'");
```

```
List<Tutor> results2 = query3.getResultList();
for (Tutor t : results2) {
      System.out.println(t);}
```

## Chaining the methods

Instead of calling the methods in separate lines, we can combine them and use ‘.’ and chain them.

```
String city = "city 2";
List<Tutor>results3 = em.createQuery("select distinct tutor from Tutor
tutor join tutor.teachingGroup student where student.address.city =
:city").setParameter("city", city).getResultList();
for(Tutor tutor:results3) {
   System.out.println(tutor);
}
```

We have chained together the series of methods into one single line.