JPA Queries

Part 3

Overview

This slide covers the following topics:

- Named Queries
- Report Queries
  - Report Query- Multiple fields
  - Report Query- Aggregation
- Native Sql
- JPQL

Named Queries

A Named Query is a predefined query identified by a meaningful name. It is typically defined in an XML configuration file (orm.xml).

Example Named Query (XML Configuration):

```
<named-query name="searchByName">
    <query>from Student as student where student.name =
:name</query>
</named-query>
```

This query is stored under META-INF/orm.xml.

Using Named Queries in Code

```
List<Student> results = em.createNamedQuery("searchByName", Student.class)
                          .setParameter("name", "Jimi Hendriks")
                          .getResultList();
for(Student student: results) {
    System.out.println(student);
}
```

Custom Mapping File: If the query file is named student-queries.xml, add this line in persistence.xml:

```
<mapping-file>META-INF/student-queries.xml</mapping-file>
```

Retrieve only the name of all students

If we want to get the name of the students, we write *a standard query* like this:

```
List<Student>students =em.createQuery("from Student")
                                                .getResultList();

for(Student s:students) {
   System.out.println(s.getName());
 }
```

It will work well and will give the name of all the students back. But what is the problem here?

Retrieve only the name of all students

The problem is that we only need the name of the students, nothing more.

By writing “from Student” not only it will get all the data about each student, but also it will join it to the tutor table and get all the data from this table too.

Think if we have 30000 students in our database, how much memory it would get to return this data (but we needed only the name of each student)

There is a better way to do this.

Report Queries

A Report Query retrieves only specific fields, improving performance and reducing memory usage.

So to get the name of all students, we can write "*select student.name*" directly in the sql statement.

```
Query q = em.createQuery("select student.name from Student student");
List<String>results = q.getResultList();
for(String name:results) {
  System.out.println(name);
}
```

Report Queries Multiple fields

With Report Query, we can retrieve multiple fields:

```
List<Object[]>results = em.createQuery("select student.name,
student.enrollmentID from Student student").getResultList();

for(Object[] obj:results) {
   System.out.println("Name: " + obj[0]);
   System.out.println("ID: " + obj[1]);
}
```

In this query, hibernate returns a list of Object arrays. In this example each element of this list will be an array of two elements.

Report Queries - Aggregation

Aggregation works with *count, min, max, sum* and *avg*.

So we can get some aggregated data on one or more columns.

For example we want total number of students in the student table:

```
long numberOfStudents = (Long)em.createQuery("select count(student)
from Student student").getSingleResult();
System.out.println("We have " + numberOfStudents + " students");
```

Native Sql

HQL works well in most cases, but for complex queries, native SQL can be used.

This is an example of using native sql:

```
List<Object[]> results = em.createNativeQuery("select s.name,
s.enrollmentid from student s").getResultList();
for(Object[] result: results) {
  System.out.println(result[0] + " ; " + result[1]);
}
```

Unlike HQL and JPQL, native SQL directly references database tables instead of mapped Java entities.

JPQL (Jakarta Persistence Query language)

JPQL is database-independent and works with JPA entities instead of database tables.

```
List<Object[]>results = em.createQuery("select student.name,
student.enrollmentID from Student student").getResultList();


long numberOfStudents = (Long)em.createQuery("select count(student)
from Student student").getSingleResult();
```

Classic HQL vs JPQL

Feature	HQL	JPQL
Query Definition	`from Student`	`select student from Student student`
Standard Compliance	Non-standard	Fully Portable
Alias Requirement	Optional	Required

Comparing Query Types in Hibernate

Query Type	Example
Regular Query	`session.createQuery("from Student")`
Named Query	`session.getNamedQuery("searchByName").setParameter("name", "Sara Higgins")`
Native Query	`session.createSQLQuery("select * from student s").addEntity(Student.class).list()`

Summary

Feature	Description
Named Queries	Predefined queries stored in XML files.
Report Queries	Efficiently retrieve specific fields.
Aggregation Queries	Use `count`, `sum`, `avg`, etc., for summary data.
Native SQL	Directly interacts with the database.
JPQL	Database-independent query language based on SQL.