Hibernate Tip: Difference between JOIN, LEFT JOIN & JOIN FETCH

Question:

I saw JPQL queries using *JOIN*, *LEFT JOIN* and *JOIN FETCH* statement.

What are the differences between these 3 join statements?

Solution:

You might know the *JOIN* and *LEFT JOIN* statements from SQL. It supports clauses with the same name and a very similar syntax. The *JOIN FETCH* clause is specific to JPA.

Let's take a closer look at all 3 options.

JOIN:

In JPQL, you can define a *JOIN* statement based on a specified association between 2 entities. Your persistence provider, e.g., Hibernate, translates this into an SQL *JOIN* statement.

The SQL *JOIN* statement tells the database to combine the columns of 2 tables to a set which you can use within the same query. You can define a join condition that specifies which rows of each table shall be joined with each other. All rows which don't fulfill the condition are not part of the set.

In most cases, this condition compares the values of a primary key column of one table with the values of a foreign key column of an associated table. But you can also define more complex conditions using multiple columns of both tables.

In JPQL, you need to define join statements based on association mappings. This mapping provides the names of the foreign key and primary key columns. That makes the definition of the join statement easier, but you can't add any additional predicates.

Here's an example of a JPQL query that returns all *Author* entities who've written a *Book* which title contains the word "Hibernate". It joins the *Author* entity with the *Book* entity and uses the *title* attribute of the *Book* in the *WHERE* clause.

```
List<Author> authors = em.createQuery("SELECT a FROM
Author a JOIN a.books b WHERE b.title LIKE
'%Hibernate%'", Author.class).getResultList();
```

After you activated the logging of SQL statements, you can see that Hibernate generates the following statement for the JPQL query. It uses the defined many-to-many association to join the *Author* table with the association table *BookAuthor*. It then joins the association table with the *Book* table.

```
16:41:15,056 DEBUG [org.hibernate.SQL] -
    select
        author0_.id as id1_0_,
        author0_.firstName as firstNam2_0_,
        author0_.lastName as lastName3_0_,
        author0_.version as version4_0_
    from
        Author author0_
    inner join
        BookAuthor books1_
            on author0_.id=books1_.authorId
    inner join
        Book book2_
            on books1_.bookId=book2_.id
    where
        book2_.title like '%Hibernate%'
```

LEFT JOIN:

The *LEFT JOIN* statement is similar to the *JOIN* statement. The main difference is that a *LEFT JOIN* statement includes all rows of the entity or table referenced on the left side of the statement.

I use that in the following example to select all *Author*s with the *lastName* "Janssen" and their *Book*s. If the database contains a *Book* for a specific *Author*, the query returns it as the second element in the *Object[]*. Otherwise, that array element is *null*.

A simple *JOIN* statement would only return the *Author*s who have written a *Book*. The second element of the *Object[]* would never be null.

```
List<Object[]> authors = em.createQuery("SELECT a, b
FROM Author a LEFT JOIN a.books b WHERE a.lastName =
'Janssen'").getResultList();
```

Hibernate generates the following SQL statement for this query. It selects all columns mapped by the *Author* and *Book* entities and uses the defined association to create a left join between the *Book* and the *Author* tables.

```
16:54:10,510 DEBUG [org.hibernate.SQL] -
    select
        author0_.id as id1_0_0_,
        book2_.id as id1_1_1_,
        author0_.firstName as firstNam2_0_0_,
        author0_.lastName as lastName3_0_0_,
        author0_.version as version4_0_0_,
        book2_.publisherid as publishe5_1_1_,
        book2_.publishingDate as publishi2_1_1_,
        book2_.title as title3_1_1_,
        book2_.version as version4_1_1_
```

```
from
    Author author0_
left outer join
    BookAuthor books1_
        on author0_.id=books1_.authorId
left outer join
    Book book2_
        on books1_.bookId=book2_.id
where
    author0_.lastName='Janssen'
```

JOIN FETCH:

The *FETCH* keyword of the *JOIN FETCH* statement is JPA-specific. It tells the persistence provider to not only join the 2 database tables within the query but to also initialize the association on the returned entity. You can use it with a *JOIN* and a *LEFT JOIN* statement.

Let's change the first example and replace the *JOIN* statement with a *JOIN FETCH* statement.

```
List<Author> authors = em.createQuery("SELECT a FROM
Author a JOIN FETCH a.books b WHERE b.title LIKE
'%Hibernate%'", Author.class).getResultList();
```

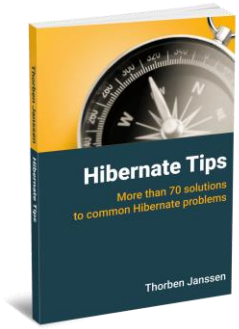
The JPQL query selects *Author* entities. But as you can see in the SQL statement, Hibernate now selects all columns mapped by the *Author* and the *Book* entity. Hibernate then maps the result set to *Author* and *Book* entities. It uses the *Book* entities to initialize the *books* attribute of each *Author* entity before it returns a *List* of *Author* entities.

```
16:57:53,384 DEBUG [org.hibernate.SQL] -
    select
        author0_.id as id1_0_0_,
        book2_.id as id1_1_1_,
        author0_.firstName as firstNam2_0_0_,
        author0_.lastName as lastName3_0_0_,
        author0_.version as version4_0_0_,
        book2_.publisherid as publishe5_1_1_,
        book2_.publishingDate as publishi2_1_1_,
        book2_.title as title3_1_1_,
        book2_.version as version4_1_1_,
        books1_.authorId as authorId2_2_0__,
        books1_.bookId as bookId1_2_0__
    from
        Author author0_
    inner join
        BookAuthor books1_
            on author0_.id=books1_.authorId
    inner join
        Book book2_
            on books1_.bookId=book2_.id
    where
        book2_.title like '%Hibernate%'
```

Learn more

JPQL is very similar to SQL and provides you with powerful query capabilities. You can learn more about it in my Ultimate Guide to JPQL Queries.

Hibernate Tips Book



Get more recipes like this one in my book Hibernate Tips: More than 70 solutions to common Hibernate problems.

It gives you more than 70 ready-to-use recipes for topics like basic and advanced mappings, logging, Java 8 support, caching and statically and dynamically defined queries.