Difference between JOIN and JOIN FETCH in Hibernate

Please help me understand where to use a regular JOIN and where a JOIN FETCH.

For example, if we have these two queries

```
FROM Employee emp
JOIN emp.department dep
```

and

```
FROM Employee emp
JOIN FETCH emp.department dep
```

Is there any difference between them? If yes, which one to use when?

java hibernate join hql fetch

edited Aug 4 '16 at 1:09

asked Jul 2 '13 at 16:50

abbas
1,524 1 15 21

1 you can find it here link read **14.3. Associations and joins** – Angga Jul 2 '13 at 18:27

3 I have gone through that documententation but still don't know where should I use a JOIN and where a JOIN FETCH. – abbas Jul 2 '13 at 20:06

1 If you have @oneToOne mapping set to FetchType.LAZY and you use second query(because you need Department objects to be loaded as part of Employee objects) what Hibernate will do is, it will issue queries to fetch Department objects for every individual Employee object it fetches from DB. Later in the code you might access Department objects via Employee to Department single-valued association and Hibernate will not issue any query to fetch Department object for the given Employee. Remember Hibernate still issues queries equal to the number of Employees it has fetched. – Bunti Dec 14 '15 at 22:48

To assist in the doc hunt ~ Fetching Strategies – Eddie B Nov 15 '16 at 18:55

4 Answers

In this two queries, you are using JOIN to query all employees that have at least one department associated.

But, the difference is: in the first query you are returning only the Employes for the Hibernate. In the second query, you are returning the Employes **and** all Departments associated.

So, if you use the second query, you will not need to do a new query to hit the database again to see the Departments of each Employee.

You can use the second query when you are sure that you will need the Department of each Employee. If you not need the Department, use the first query.

I recomend read this link if you need to apply some WHERE condition (what you probably will need): How to properly express JPQL "join fetch" with "where" clause as JPA 2 CriteriaQuery?

Update

If you don't use `fetch` and the Departments continue to be returned, is because your mapping between Employee and Department (a `@OneToMany`) are setted with `FetchType.EAGER` . In this case, any HQL (with `fetch` or not) query with `FROM Employee` will bring all Departments. Remember that all mapping *ToOne (`@ManyToOne` and `@OneToOne`) are EAGER by default.

edited May 23 at 12:34

Community♦
1 1

answered Apr 30 '14 at 0:37

Dherik
2,033 1 22 41

1 Which behaviour will be if we execute statement without fetch and get result. Then within session we will treat to department? – gstackoverflow Nov 16 '15 at 10:17

1 @gstackoverflow, yes – Dherik Feb 1 '16 at 16:14

in this link i mentioned before on the comment, read this part :

> A "fetch" join allows associations or collections of values to be initialized along with their parent objects using a single select. This is particularly useful in the case of a collection. **It effectively overrides the outer join and lazy declarations of the mapping file** for associations and collections.

this "JOIN FETCH" will have it's effect if you have (fetch = FetchType.LAZY) property for a collection inside entity(example bellow).

And it is only effect the method of "when the query should happen". And you must also know this:

> hibernate have two orthogonal notions : when is the association fetched and how is it fetched. It is important that you do not confuse them. We use fetch to tune performance. We can use lazy to define a contract for what data is always available in any detached instance of a particular class.

when is the association fetched --> your "FETCH" type

how is it fetched --> Join/select/Subselect/Batch

In your case, FETCH will only have it's effect if you have department as a set inside Employee, something like this in the entity:

```
@OneToMany(fetch = FetchType.LAZY)
private Set<Department> department;
```

when you use

```
FROM Employee emp
JOIN FETCH emp.department dep
```

you will get `emp` and `emp.dep` . when you didnt use fetch you can still get `emp.dep` but hibernate will processing another select to the database to get that set of department.

so its just a matter of performance tuning, about you want to get all result(you need it or not) in a single query(eager fetching), or you want to query it latter when you need it(lazy fetching).

Use eager fetching when you need to get small data with one select(one big query). Or use lazy fetching to query what you need latter(many smaller query).

use fetch when :

- no large **unneeded** collection/set inside that entity you about to get
- communication from application server to **database server too far** and need long time
- you may need that collection latter when you don't have the access to it(**outside** of the **transactional** method/class)

edited Feb 16 at 2:17

answered Jul 3 '13 at 4:27

Angga
1,499 7 15

Could you explain it for the queries that I just wrote in the updated question. – abbas Jul 3 '13 at 13:44

1 done, i hope this help you. – Angga Jul 4 '13 at 3:15

Dherik : I'm not sure about what you say, when you don't use fetch the result will be of type : `List<Object[ ]>` which means a list of Object tables and not a list of Employee.

```
Object[0] refers an Employee entity
Object[1] refers a Departement entity
```

When you use fetch, there is just one select and the result is the list of Employee `List<Employee>` containing the list of departements. It overrides the lazy declaration of the entity.

edited Feb 21 '15 at 18:52

Konrad Krakowiak
8,685 9 37 40

answered Feb 21 '15 at 18:02

Bilal BOUTAYA
775 6 17

I don't know if I understand your concern. If you don't use `fetch` , your query will return only the Employees. If the Departments, even in this case, continue to be returned, is because your mapping between Employee

and Department (a @OneToMany) are setted with FetchType.EAGER. In this case, any HQL (with `fetch` or not) query with `FROM Employee` will bring all Departments. – Dherik May 9 '15 at 15:05

Without using fetch (join term only), the result would be an array of collections, two rows, the first is a collection of Employees and the second is a collection of Departments. Using eager fetch or lazy fetch, departments will be fetched. – Bilal BOUTAYA May 9 '15 at 15:28

Without fetch on HQL, this will happen only if your mapping between Employee and Department are EAGER (`@OneToMany(fetch = FetchType.EAGER`). If is not the case, the Departments will not be returned. – Dherik May 9 '15 at 15:50

@Dherik try it yourself, You'll get a ClassCastException. – Bilal BOUTAYA May 9 '15 at 16:14

I figure out the problem. Is not a fetch problem, but how the `select` was made in the HQL. Try `SELECT emp FROM Employee emp JOIN FETCH emp.department dep` . JPA/Hibernate have this behaviour of return a `List` of `Object[]` when you ommit the `SELECT` part. – Dherik Nov 24 '15 at 23:46

If you have @oneToOne mapping set to FetchType.LAZY and you use second query(because you need Department objects to be loaded as part of Employee objects) what Hibernate will do is, it will issue queries to fetch Department objects for every individual Employee object it fetches from DB. Later in the code you might access Department objects via Employee to Department single-valued association and Hibernate will not issue any query to fetch Department object for the given Employee. Remember Hibernate still issues queries equal to the number of Employees it has fetched. Hibernate will issue same number of queries in both above queries if you wish to access Department objects of all Employee objects

answered Dec 14 '15 at 22:51

Bunti
960 7 14