

N
+
1
P
r
o
b
l
e
m

by
Ale
xan
der
Ma
kee

Discover Anything 🔍

LoginReadWrite🔔☰

HACKERNOON

Stellar Launch your DeFi idea on Stellar

3 Ways to Deal With Hibernate N+1 Problem by @alexandermakeev

h,
202
2

TCJR

EN

1x

Read
by
Dr.
One

Au
dio
Pre
sen
ted
.. by

Hibernate N+1 problem occurs when you use FetchType . LAZY for your entity associations. If you perform a query to select n-entities and if you try to call any access method of your entity's lazy association, Hibernate will perform n-additional queries to load lazily fetched objects.

For example, we have the following Author entity with one-to-many books collection:

```
public class Author {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Integer id;  
    private String fullName;  
    @OneToMany(fetch = FetchType.LAZY)  
    private Set<Book> books;  
}
```

Let's try to load all authors and print each author's name with his books collection size:

```
entityManager.createQuery("select a from Author a", Author.class)  
    .getResultList()  
    .forEach(a -> System.out.printf("%s had written %d books\n",  
                                    a.getFullName(), a.getBooks().size()));
```

The first query Hibernate will generate is to select all authors:

```
SELECT author0_.id AS id1_0_,  
       author0_.fullName AS fullname2_0_  
FROM authors author0_;
```

After that, when we call `size()` method on the books collection, this association needs to be initialized, so Hibernate will perform an additional query:

```
SELECT books0_.author_id AS author_i4_1_0_,  
       books0_.id AS id1_1_0_,  
       books0_.id AS id1_1_1_,  
       books0_.author_id AS author_i4_1_1_,  
       books0_.title AS title2_1_1_,  
       books0_.year AS year3_1_1_  
FROM books books0_  
WHERE books0_.author_id=?;
```

This query will be called n-times for each author when we print the amount of books in addition to the first query. Thus the total number of queries will be equal to $N+1$.

Hibernate provides a couple of ways to eliminate this issue:

1. The first solution is to use join fetch:

```
entityManager.createQuery("select a from Author a left join fetch a.books",
                           Author.class);
```

```
SELECT author0_.id AS id1_0_0_,
       books1_.id AS id1_1_1_,
       author0_.fullName AS fullname2_0_0_,
       books1_.author_id AS author_i4_1_1_,
       books1_.title AS title2_1_1_,
       books1_.year AS year3_1_1_,
       books1_.author_id AS author_i4_1_0_,
       books1_.id AS id1_1_0_
FROM authors author0_
LEFT OUTER JOIN books books1_ ON author0_.id=books1_.author_id;
```

This query works fine, but it has one issue: it doesn't allow us to use pagination because the limit will not be applied to the authors. If you specify `query.setMaxResults(n)`, Hibernate will fetch all existing rows and do the pagination in the memory, significantly increasing memory consumption.

2. Another way is to use `@BatchSize` on the lazy association:

```
public class Author {
    ...
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "author")
    @BatchSize(size = 10)
    private Set<Book> books;
}
```

Hibernate will create the first query to retrieve all authors:

```
SELECT author0_.id AS id1_0_,
       author0_.fullName AS fullname2_0_
FROM authors author0_;
```

In this case, we can easily perform the pagination on the authors. Then, when we call `size()` method on the books collection, Hibernate will perform this query:

```
/* load one-to-many Author.books */
SELECT books0_.author_id AS author_i4_1_1_,
       books0_.id AS id1_1_1_,
       books0_.id AS id1_1_0_,
       books0_.author_id AS author_i4_1_0_,
       books0_.title AS title2_1_0_,
       books0_.year AS year3_1_0_
```

```
FROM books books0_
WHERE books0_.author_id in (?, ?, ?, ?, ?, ?, ?, ?, ? /*batch size*/);
```

This query will be called N/M times, where N is the amount of authors and M is the specified batch size. Totally we will call N/M+1 queries.

3. The third way is to use a sub query returning a list of author identifiers

Hibernate provides this opportunity by setting `@Fetch(FetchMode.SUBSELECT)` on the lazy association:

```
public class Author {
    ...
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "author")
    @Fetch(FetchMode.SUBSELECT)
    private Set<Book> books;
}
```

The first query will load all authors:

```
SELECT author0_.id AS id1_0_,
       author0_.fullName AS fullname2_0_
FROM authors author0_;
```

The second query will fetch books by using authors sub query:


```
SELECT books0_.author_id AS author_i4_1_1_,
       books0_.id AS id1_1_1_,
       books0_.id AS id1_1_0_,
       books0_.author_id AS author_i4_1_0_,
       books0_.title AS title2_1_0_,
       books0_.year AS year3_1_0_
FROM books books0_
WHERE books0_.author_id in
      (SELECT author0_.id
       FROM authors author0_);
```

If you look closely into the IN condition, you'll see that the code inside the sub query almost repeats the first query. It can slow the performance if we have to execute a very complex query twice. To speed up this case we can filter and page authors retrieving their ids by the 1st query. Then we can pass these identifiers directly to the 2nd query's sub query:


```
List<Integer> authorIds = em.createQuery("select a.id from Author a", Integer.class)
                          .setFirstResult(5)
                          .setMaxResults(10)
                          .getResultList();
```

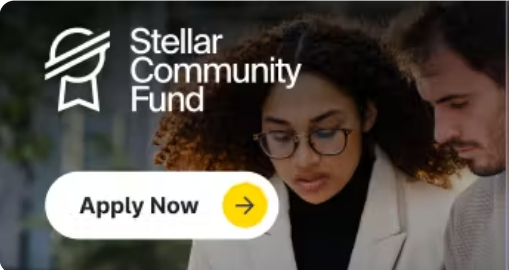
```
List<Author> resultList = entityManager.createQuery("select a from Author a"
    + " left join fetch a.books"
    + " where a.id in :authorIds", Author.class)
    .setParameter("authorIds", authorIds)
    .getResultList();
```





Stellar
Community
Fund

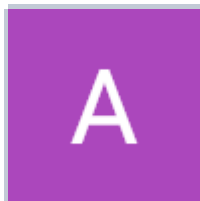
Apply Now 



FAST TRACK YOUR SMART CONTRACT DEVELOPMENT

Build with us at the virtual Stellar Startup Camp

About Author



Alexander Makeev @alexandermakeev

name@company.com

Subscribe

Senior SWE at Layermark

Read My Stories

Comments

Add Comment



@avinashkolap

6 months ago