LAB7 REPORT

Hibernate Optimization

ABSTRACT

Hibernate Optimization for given data set. Runtime and generated queries are included for better understand Hibernate query optimization. Source code:

Davaabayar Battogtokh

CS544 – Enterprise Applications

Contents

Task description

In this exercise we will use System.nanotime() to check how long it takes for MySQL and Hibernate to retrieve the same dataset with different fetching strategies.

The Application:

The application has a Populate.java file that will insert 100,000 owner objects, each with 10 associated pet objects into the database. Run it once (will take a while).

Then change line 24 of the persistence.xml file to have the value of “none” instead of “drop-and-create”. This will stop the tables from being re-created every time and keeping you from having to recreate all the data.

Then run App.java, which will create an N+1 and tell you how long it took.

The Exercise:

Consider what the application does, and write down which strategy you think will perform best under these circumstances. To get a more accurate time you should probably run each test 3 times and take the average, but once is okay to get an idea.

a) Add the @LazyCollection with option EXTRA to the association and run App again.

b) Remove the @LazyCollection, and modify the mapping for Owner.java to use batch fetching, batch size 10. Also check the time when using sizes 5 and 50.

c) Modify the mapping to use the sub-select strategy instead of batch fetching.

d) Remove the sub-select strategy and use a join fetch query in App.java to retrieve everything. Also check the difference between using a named query, or just a query directly in code.

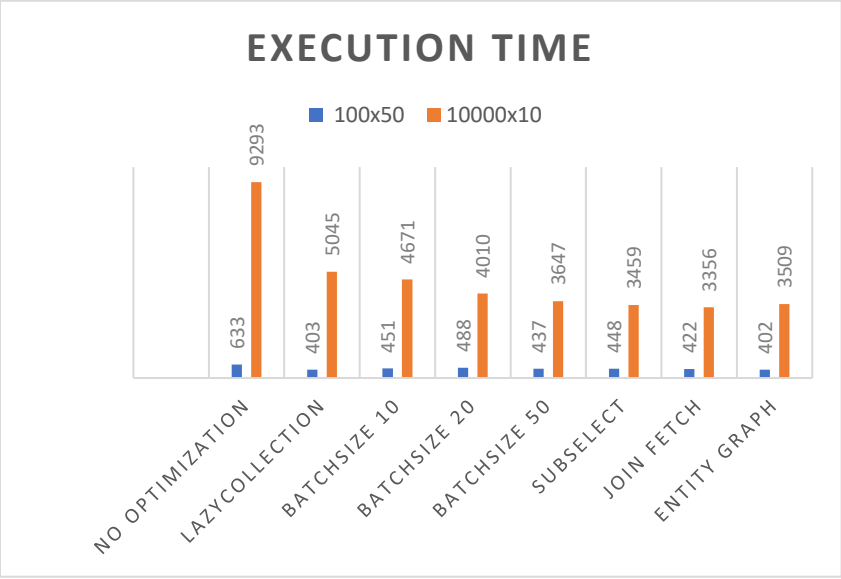
e) Lastly modify the application to use an Entity Graph instead of a join fetch.

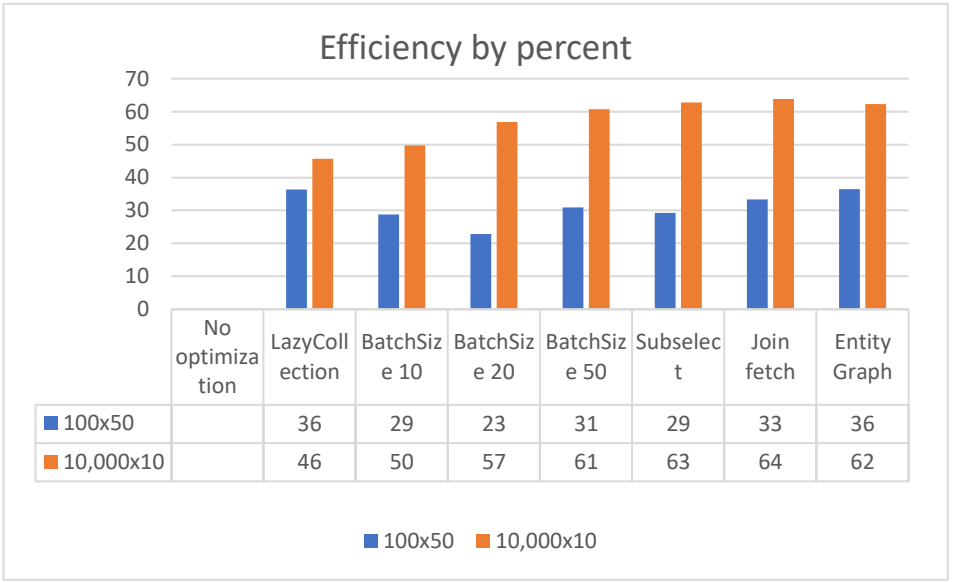
Check to see if the strategy you thought would perform best was indeed the best for this situation. Remember, just because a strategy performed well under these circumstances does not necessarily mean it will perform well under other circumstances.

Runtime

Data size		Initial	Relation			Query	
Owners	**Pets**	No optimization	LazyCollection	BatchSize	Subselect	Join fetch	Entity Graph
		N	N	N/batch size	1	1	1
100	**5**	633ms	403ms	10 = 451ms 20 = 488ms 50 = 437ms	448ms	422ms	402ms
10000	**10**	9293ms	5045ms	10 = 4671ms 20= 4010ms 50 = 3647ms	3459ms	3356ms	3509ms

Comparison by graph





Solution summary

Optimization Method	DB hit	Syntax

@LazyCollection	N	`@OneToMany (cascade={CascadeType.PERSIST})`
		`@JoinColumn (name="clientid")`
		`@LazyCollection(LazyCollectionOption.EXTRA)`
		`private List<Pet> pets;`

@Batchsize	N/batchSize	`@OneToMany (cascade={CascadeType.PERSIST})`
		`@JoinColumn (name="clientid")`
		`@BatchSize(size=10)`
		`private List<Pet> pets;`

@FetchMode. Subselect	1	`@OneToMany(cascade = {CascadeType.PERSIST})`
		`@JoinColumn(name="clientid")`
		`@Fetch(FetchMode.SUBSELECT)`
		`private List<Pet> pets;`

Join Fetch query	1	`@OneToMany (cascade={CascadeType.PERSIST})`
		`@JoinColumn (name="clientid")`
		`private List<Pet> pets;`
		
		`TypedQuery<Owner> query = em.createQuery("from Owner o JOIN FETCH o.pets",`
		`Owner.class);`

Entity Graph query	1	`EntityGraph<Owner> graph = em.createEntityGraph(Owner.class);`
		`graph.addAttributeNodes("pets");`
		`TypedQuery<Owner> query = em.createQuery("from Owner",Owner.class);`
		`query.setHint("javax.persistence.fetchgraph", graph);`
		`List<Owner> ownerlist = query.getResultList();`

Generated queries for each method

Initial query

```
select
    pets0_.clientid as clientid3_1_0_,
    pets0_.id as id1_1_0_,
    pets0_.id as id1_1_1_,
    pets0_.name as name2_1_1_
  from
    Pet pets0_
  where
    pets0_.clientid=?
```

N / 633 milliseconds.

Join fetch query

```
select
    owner0_.id as id1_0_0_,
    pets1_.id as id1_1_1_,
    owner0_.name as name2_0_0_,
    pets1_.name as name2_1_1_,
    pets1_.clientid as clientid3_1_0__,
    pets1_.id as id1_1_0__
  from
    Owner owner0_
  inner join
    Pet pets1_
      on owner0_.id=pets1_.clientid
```

1 / 422 milliseconds.

@LazyCollection

```
  select
    count(id)
  from
    Pet
  where
    clientid =?
```

N | 403 milliseconds.

FetchMode.SUBSELECT

```
  select
    pets0_.clientid as clientid3_1_1_,
    pets0_.id as id1_1_1_,
    pets0_.id as id1_1_0_,
    pets0_.name as name2_1_0_
  from
    Pet pets0_
  where
    pets0_.clientid in (
      select
        owner0_.id
      from
        Owner owner0_
    )
```

1 /448 milliseconds.

Batch fetching / @BatchSize(size=<n>)

```
select
    pets0_.clientid as clientid3_1_1_,
    pets0_.id as id1_1_1_,
    pets0_.id as id1_1_0_,
    pets0_.name as name2_1_0_
  from
    Pet pets0_
  where
    pets0_.clientid in (
      ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,?, ?, ?, ?, ?,
    )
```

N/batchSize / 437 milliseconds

Entity Graph

```
    select
        owner0_.id as id1_0_0_,
        pets1_.id as id1_1_1_,
        owner0_.name as name2_0_0_,
        pets1_.name as name2_1_1_,
        pets1_.clientid as clientid3_1_0__,
        pets1_.id as id1_1_0__
      from
        Owner owner0_
      left outer join
        Pet pets1_
          on owner0_.id=pets1_.clientid
```

1 / 402 milliseconds.