|  |  |
| --- | --- |
| A picture of a winding road and trees  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 2](#_Toc26288669)

[Runtime Report 3](#_Toc26288670)

[Solution summary 3](#_Toc26288671)

[0. Initial query 4](https://mum0-my.sharepoint.com/personal/dbattogtokh_mum_edu/Documents/Lab%207.docx#_Toc26288672)

[Generated queries for each method 4](#_Toc26288673)

[1. @LazyCollection 4](https://mum0-my.sharepoint.com/personal/dbattogtokh_mum_edu/Documents/Lab%207.docx#_Toc26288674)

[2. Batch fetching / @BatchSize(size=<n>) 4](https://mum0-my.sharepoint.com/personal/dbattogtokh_mum_edu/Documents/Lab%207.docx#_Toc26288675)

[3. FetchMode.SUBSELECT 4](https://mum0-my.sharepoint.com/personal/dbattogtokh_mum_edu/Documents/Lab%207.docx#_Toc26288676)

[4. Join fetch query 4](https://mum0-my.sharepoint.com/personal/dbattogtokh_mum_edu/Documents/Lab%207.docx#_Toc26288677)

[5. Entity Graph 4](https://mum0-my.sharepoint.com/personal/dbattogtokh_mum_edu/Documents/Lab%207.docx#_Toc26288678)

# 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.

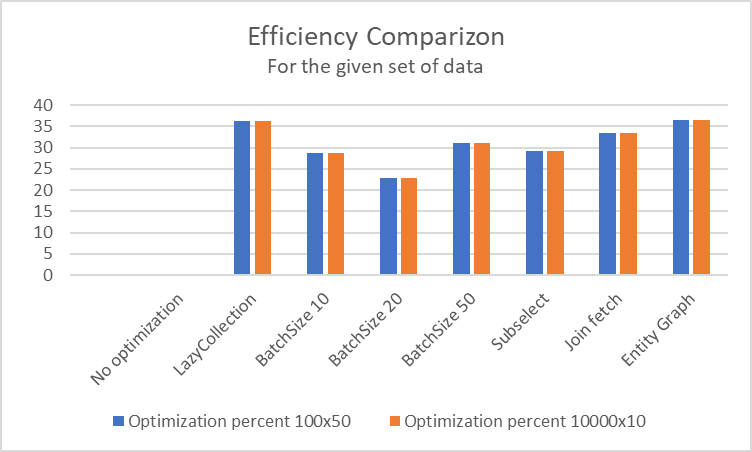
d) Remove the sub-select strategy and use a join fetch query in App.java to retrieve everything. Also check the difference between using a named query, or just a query directly in code.

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 Report

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **100x50** | | **10000x10** | |
| **Duration** | **%** | **Duration** | **%** |
| No optimization | 633 | 0 | 5064 | 0 |
| LazyCollection | 403 | 36 | 3224 | 36 |
| BatchSize 10 | 451 | 29 | 3608 | 29 |
| BatchSize 20 | 488 | 23 | 3904 | 23 |
| BatchSize 50 | 437 | 31 | 3496 | 31 |
| Subselect | 448 | 29 | 3584 | 29 |
| Join fetch | 422 | 33 | 3376 | 33 |
| Entity Graph | 402 | 36 | 3216 | 36 |

# 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.***

## @LazyCollection

select

count(id)

from

Pet

where

clientid =?

***N | 403 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 (

?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,?, ?, ?, ?, ?,

)

***1 / 437 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.***

## 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.***

## 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.***