|  |  |
| --- | --- |
| 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](#_Toc26291350)

[Runtime 3](#_Toc26291351)

[Comparison by graph 3](#_Toc26291352)

[Solution summary 4](#_Toc26291353)

[Initial query 5](file:///C:\Users\610567\Documents\Lab7Optimization\Lab7-Report.docx#_Toc26291354)

[Generated queries for each method 5](#_Toc26291355)

[@LazyCollection 5](file:///C:\Users\610567\Documents\Lab7Optimization\Lab7-Report.docx#_Toc26291356)

[Batch fetching / @BatchSize(size=<n>) 5](file:///C:\Users\610567\Documents\Lab7Optimization\Lab7-Report.docx#_Toc26291357)

[FetchMode.SUBSELECT 5](file:///C:\Users\610567\Documents\Lab7Optimization\Lab7-Report.docx#_Toc26291358)

[Join fetch query 5](file:///C:\Users\610567\Documents\Lab7Optimization\Lab7-Report.docx#_Toc26291359)

[Entity Graph 5](file:///C:\Users\610567\Documents\Lab7Optimization\Lab7-Report.docx#_Toc26291360)

# 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.***

## @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 (

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

)

***N/batchSize / 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.***