## Team members

- Minh Toan Nguyen

- Si Duy Nguyen

- Dinh Trung Tran

Sample data

|  |  |  |
| --- | --- | --- |
| User | Product | Review |
| 100 | 1,000 | 1,000 for each product |

**Performance of getting all products**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fetch mode | Fetch | No of SQL query | Execution time (s) | Memory usage |
| SELECT | EAGER | 1001 | 16 | 1.37 GB |
| SELECT | LAZY | 1001 | 6.16 | 1.01 GB |
| JOIN | EAGER | 1 | 28 | 2.63 GB |
| SUBSELECT | EAGER | 2 | 15.31 | 1.35 GB |
| SUBSELECT | LAZY | 2 | 13.40 | 1.16 GB |
| BATCH (Size = 100) | EAGER | 11 | 16.34 | 1.35 GB |
| BATCH (Size = 100) | LAZY | 11 | 16.06 | 1.35 GB |

# 1. How we Implementation:

We created the services class for each domain like user, product and review in each Entity of domain

In this test, we apply this test with relationship between product and review.

in the "Product" entity:

@Entity  
public class Product {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 private double price;  
  
 @OneToMany(mappedBy = "product", fetch = FetchType.*EAGER*)  
 @Fetch(value = FetchMode.*JOIN*)  
 private List<Review> reviews;  
}

For instansce, with JOIN: @Fetch(value = FetchMode.JOIN)

A screenshot of a computer code

Description automatically generated

For instansce, with BATCH: @Fetch(value = FetchMode.BATCH)

A screenshot of a computer program

Description automatically generated

# 2. overview of the implemented fetching strategies.

# FetchMode SUBSELECT

EAGER:

A screenshot of a computer

Description automatically generated

LAZY:

A screenshot of a computer

Description automatically generated

### **#FetchMode JOIN:**

A screenshot of a computer

Description automatically generated

### **#FetchMode SUBSELECT**

EAGER:

A screenshot of a computer

Description automatically generated

A screenshot of a web page

Description automatically generated

LAZY:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### **#FetchMode: BATCH**

**EAGER**:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**LAZY:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#### **Compare the fetching strategies:**

* Select Fetching Strategy: Retrieves associated entities in both eager and lazy, with lazy then retrieves when they are accessed. Generates separate SQL queries for each association, leading to N+1 query problem and can result in performance issues.
* Join Fetching Strategy: Uses SQL JOINs to retrieve associated entities in a query,
* Subselect Fetching Strategy: Retrieves data in both eager and lazy, executes a separate SQL query to retrieve associated entities. But requires an additional SQL query for each association, which can make performance issues and more complex SQL queries compared to join fetching.
* Batch Fetching Strategy: Reduces the number of SQL queries by batching multiple entity fetches into a single SQL query, the number of queries depend on the size of each batch. And may lead to increased memory usage if the batch size is too large but if batch size is too small, then cause a lot of SQL query.

In conclusion, the choice of fetch strategy depends on factors such as the size of the data, performance requirements, the number of associations to be fetch, and be careful of encountering N+1 query problems. It is essential to carefully analyze these factors and choose the most appropriate fetch strategy for each use case to achieve optimal performance and avoid common issues.