

Team members

- Minh Toan Nguyen (617556)
- Si Duy Nguyen (617588)
- Dinh Trung Tran (617578)

Requirements

- * Make domain models for `user`, `product`, `review`.
- * Populate the database with sample data including **at least** 100 users, 1000 products, and 1000 reviews for each product.
- * Implement methods in your service layer, each demonstrating one of the fetching strategies (`select`, `join`, `subselect`, `batch`).
- * Measure the number of SQL queries generated and the execution time for each fetching strategy.
- * Analyze the differences in performance among the fetching strategies, focusing on the trade-offs between the number of queries generated and the overall execution time.
- * Discuss the scenarios in which each fetching strategy would be most effective, considering factors such as data size, association complexity, and application requirements.

Submission

- * **Report**: Your report should include:
 - * An overview of the implemented fetching strategies. For each strategy, have the followings:
 - * `Implementation`: Explain how you implement it and write down the query generated by hibernate.
 - * `Observation`: Explain the output. For example, Resulted in a total of x SQL queries for fetching y users and their z products.
 - * `Performance`: For example, Execution time averaged at 50ms, with a moderate memory usage of 120MB. Data transfer size was relatively low at 300KB, reflecting the efficiency of targeted queries.
 - * `Practical Use`: For example, Best suited for scenarios where associated entities are rarely accessed or when the number of associated entities is unknown or highly variable.
 - * A detailed analysis of the performance evaluation results. Create charts to show the differences.
 - * `Query Execution Time Comparison`: A bar chart displaying the average execution time for each fetching strategy.
 - * `Total Number of SQL Queries Generated`: Another bar chart that shows the total number of SQL queries executed for each strategy.
 - * `Memory Usage`: A line chart showing the peak memory usage of each fetching strategy during execution.
 - * `Data Transfer Size`: A bar chart comparing the amount of data transferred from the database to the application for each fetching strategy.
 - * A discussion on the suitability of each fetching strategy for different use cases.

- * Your conclusion from this experiment.
- * Fork the repository and push your changes.
- * Once you finished your project, send a Pull Request. (Send only one Pull Request once you finish the assignment.)

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 = FetchType.JOIN)
    private List<Review> reviews;
}
```

Do the same with JOIN, SUBSELECT, BATCH.

For instance, with JOIN: @Fetch(value = FetchType.JOIN)

For instance, with SUBSELECT: @Fetch(value = FetchType.SUBSECTET)

For instance, with BATCH: @Fetch(value = FetchType.BATCH)

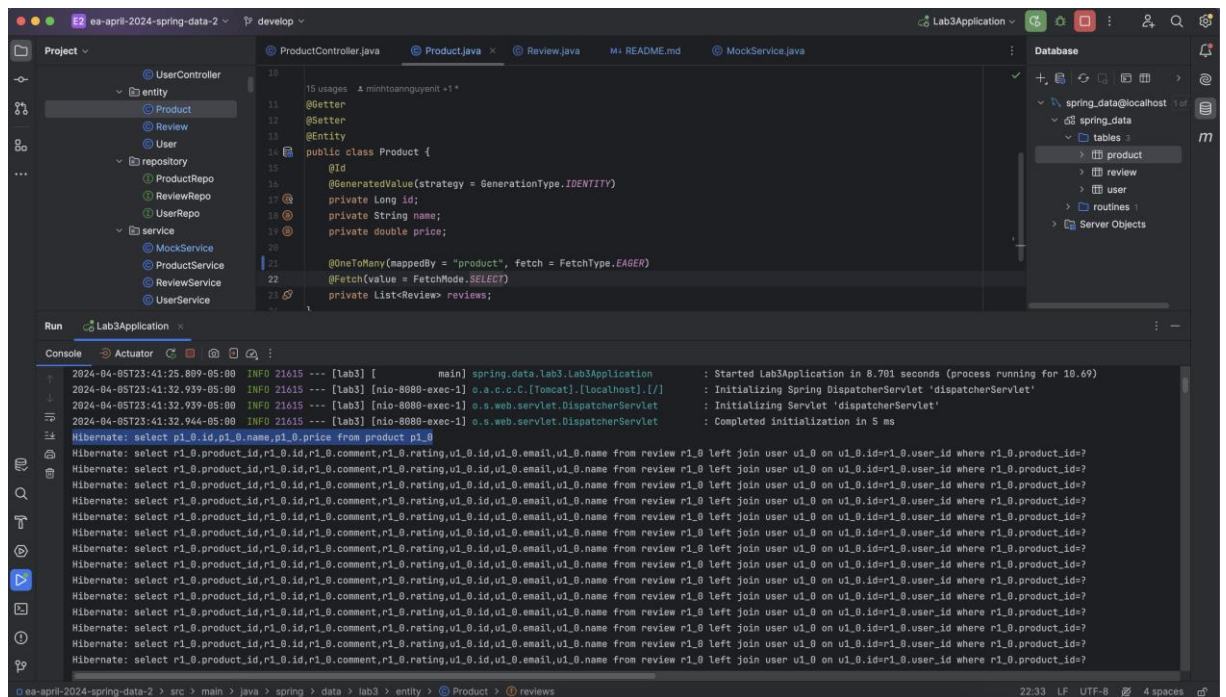
2. overview of the implemented fetching strategies.

- FetchType is EAGER:

FetchMode SELECT

Api get all products

- 1 query to get all products: select p1_0.id,p1_0.name,p1_0.price from product p1_0
 - N queries to get reviews of N products: select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.email,u1_0.name from review r1_0 left join user u1_0 on u1_0.id=r1_0.user_id where r1_0.product_id=?
- ⇒ Total SQL queries: N + 1



- Performance: When we try to get 1000 products and each product has 1000 reviews in it, we got a error that maximum response size reached, we decrease to 500 products the still get error, then we try with 300 products

EA Course / product / Get all

GET `{{base_url}}/products/get-products-by-select` Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response

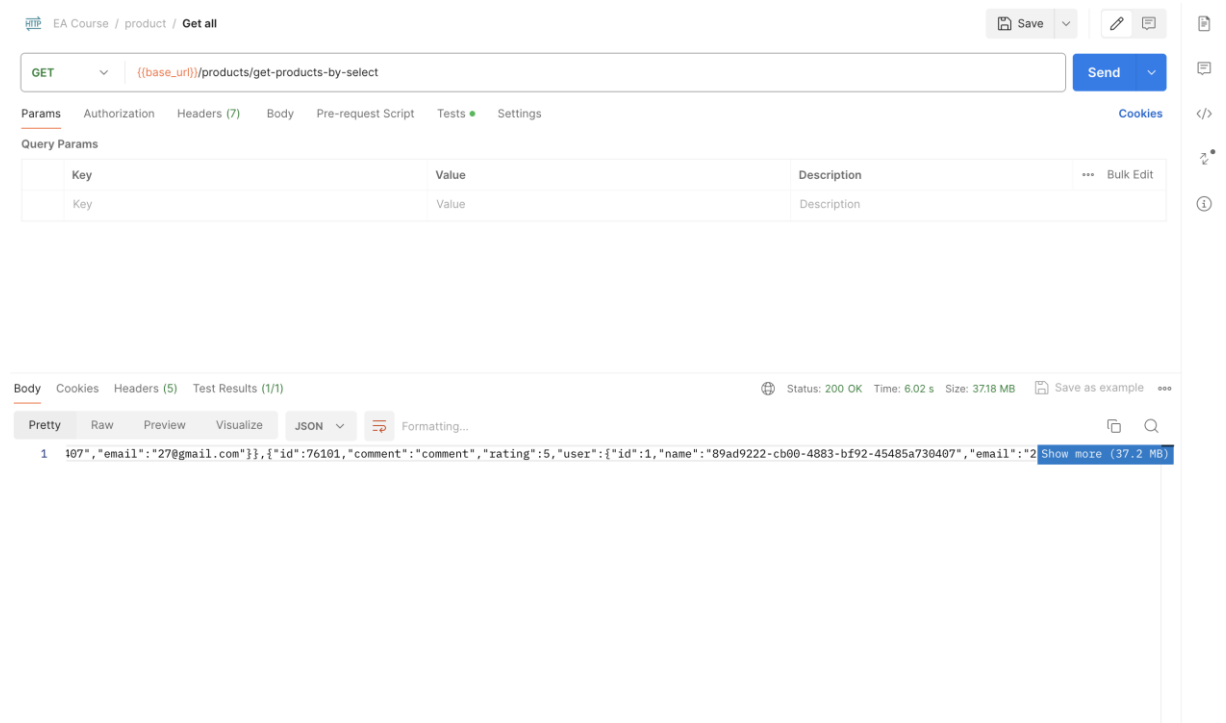
Could not get response

Error: Maximum response size reached [View in Console](#)

[Learn more about troubleshooting API requests](#)

- Performance with 300 products and each product has 1000 reviews in it: Total execution time is 11.98s with memory size is 37.2MB

- Performance: Total execution time is 6.02s with memory size is 37.2MB



EA Course / product / Get all

GET `{{base_url}}/products/get-products-by-select` Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results (1/1) Status: 200 OK Time: 6.02 s Size: 37.18 MB Save as example

Pretty Raw Preview Visualize JSON Formatting...

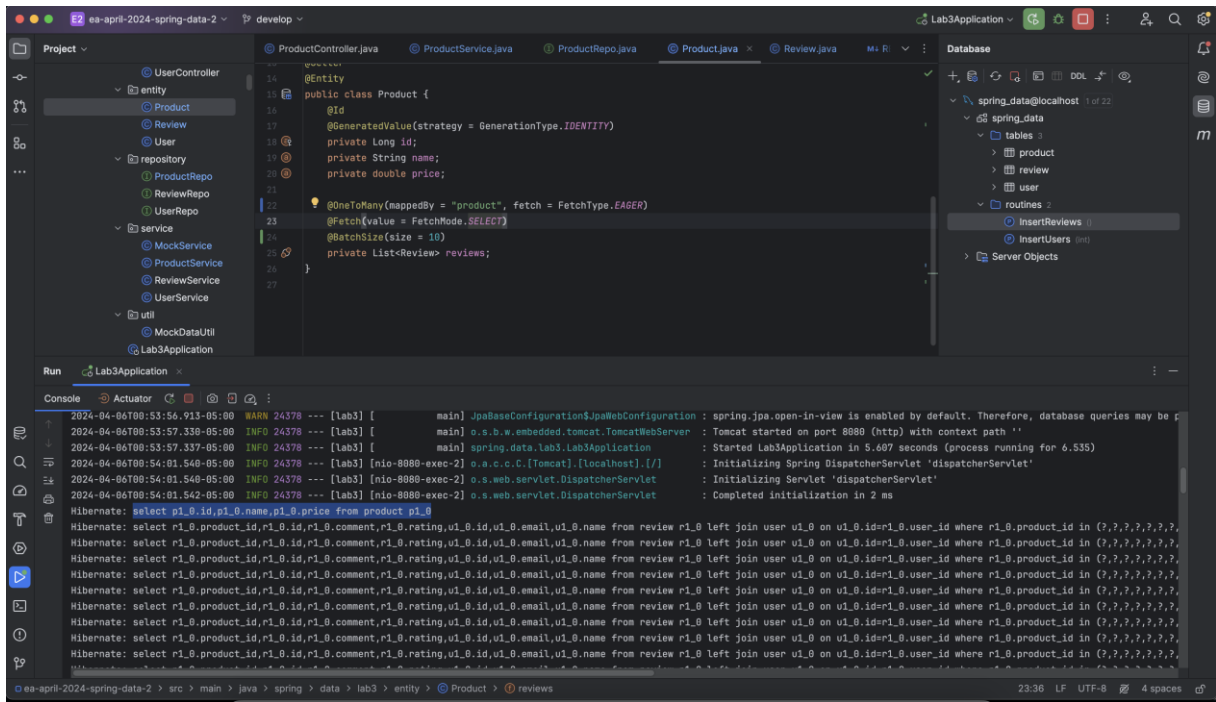
```
1 187", "email": "27@gmail.com"}}, {"id": 76181, "comment": "comment", "rating": 5, "user": {"id": 1, "name": "89ad9222-cb00-4883-bf92-45485a736407", "email": "2
```

Show more (37.2 MB)

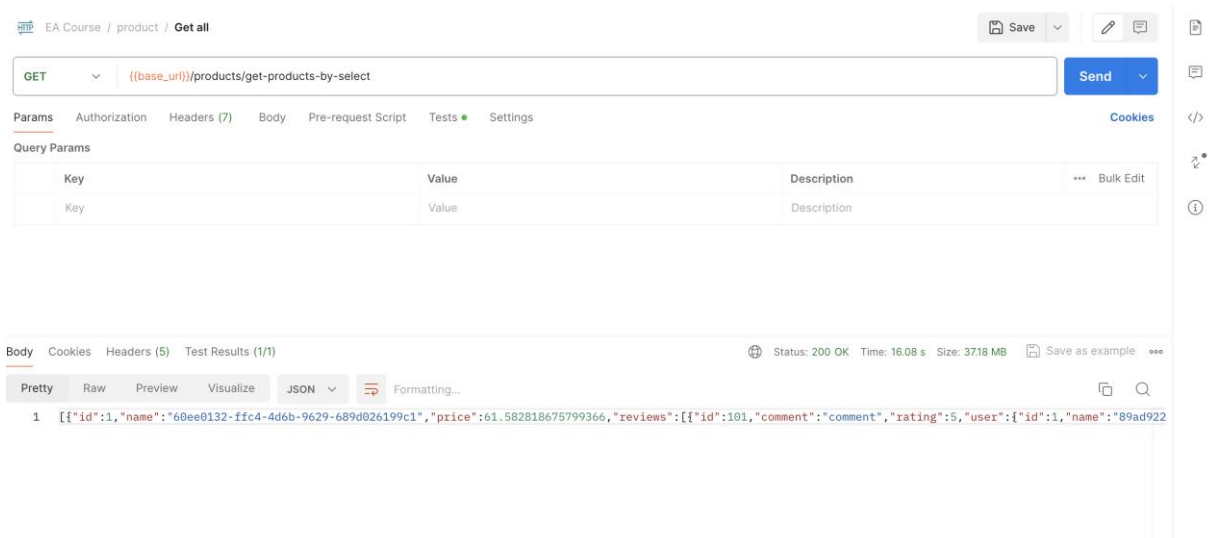
FetchMode BATCH

Api get all products

- Batch size is 10.
 - 1 query to get all products: `select p1_0.id,p1_0.name,p1_0.price from product p1_0`
 - Batch size is 10 -> $(\text{total products} / \text{batch size}) = 1000 / 10 = 100$ queries to get reviews of N products: `select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.email,u1_0.name from review r1_0 left join user u1_0 on u1_0.id=r1_0.user_id where r1_0.product_id in (?, ?, ?, ?, ?, ?, ?, ?, ?)`
- ⇒ Total SQL queries: $N/10 + 1$



- Performance (Batch size is 10): Total execution time is 16.08s with memory size is 37.2MB



- Performance (Batch size is 100): Total execution time is 10.90s with memory size is 37.2MB

- Performance: Total execution time is 20.83s with memory size is 37.2MB

Overview GET Get all POST Create PUT Update DEL Delete GET mock data GET Get all EA Course

EA Course / product / Get all

GET `{{(base_url)}}/products/get-products-by-select` Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results (1/1) Status: 200 OK Time: 20.83 s Size: 3718 MB Save as example

Pretty Raw Preview Visualize JSON Formatting...

```
1 [{"id":1,"name":"60ee0132-ffc4-4d6b-9629-689d026199c1","price":61.582818675799366,"reviews":[{"id":101,"comment":"comment","rating":5,"user":{"id":1,"name":"89ad922
```

FetchMode SUBSELECT

Api get all products

- 1 query to get all products: `select p1_0.id,p1_0.name,p1_0.price from product p1_0`
- ⇒ 1 queries to get reviews of N products: `select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.email,u1_0.name from review r1_0 left join user u1_0 on u1_0.id=r1_0.user_id where r1_0.product_id in (select p1_0.id from product p1_0)`
- ⇒ Total SQL queries: 2

Project ProductController.java ProductService.java ProductRepo.java Product.java Review.java Database

```
14 @Entity
15 public class Product {
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Long id;
19     private String name;
20     private double price;
21
22     @OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
23     @Fetch(value = FetchMode.SUBSELECT)
24     private List<Review> reviews;
25 }
26
```

Run Lab3Application

Console

```
2024-04-06T01:02:34.124-05:00 INFO 24725 --- [Lab3] [main] org.hibernate.Version : HHH0000412: Hibernate ORM core version 6.4.4.Final
2024-04-06T01:02:34.191-05:00 INFO 24725 --- [Lab3] [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
2024-04-06T01:02:34.580-05:00 INFO 24725 --- [Lab3] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-04-06T01:02:34.676-05:00 WARN 24725 --- [Lab3] [main] org.hibernate.orm.deprecation : HHH9000025: MySQLDialect does not need to be specified explicitly using 'hibernate
2024-04-06T01:02:35.724-05:00 INFO 24725 --- [Lab3] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to e
2024-04-06T01:02:35.845-05:00 INFO 24725 --- [Lab3] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-04-06T01:02:36.243-05:00 WARN 24725 --- [Lab3] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be p
2024-04-06T01:02:36.897-05:00 INFO 24725 --- [Lab3] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-06T01:02:36.905-05:00 INFO 24725 --- [Lab3] [main] spring.data.lab3.Lab3Application : Started Lab3Application in 6.965 seconds (process running for 8.004)
2024-04-06T01:02:42.076-05:00 INFO 24725 --- [Lab3] [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-06T01:02:42.076-05:00 INFO 24725 --- [Lab3] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-04-06T01:02:42.080-05:00 INFO 24725 --- [Lab3] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 4 ms
Hibernate: select p1_0.id,p1_0.name,p1_0.price from product p1_0
Hibernate: select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.email,u1_0.name from review r1_0 left join user u1_0 on u1_0.id=r1_0.user_id where r1_0.product_id in (select p1_0.id
```

- Performance: Total execution time is 17.70s with memory size is 37.2MB

The screenshot shows a REST client interface with the following details:

- Request:** GET `{{base_url}}/products/get-products-by-select`
- Response:** Status: 200 OK, Time: 17.70 s, Size: 37.18 MB
- Response Body (JSON):**

```
1 [{"id":1,"name":"60ee0132-ffc4-4d6b-9629-689d026199c1","price":61.582818675799366,"reviews":{"id":101,"comment":"comment","rating":5,"user":{"id":1,"name":"89ad922"}}

```

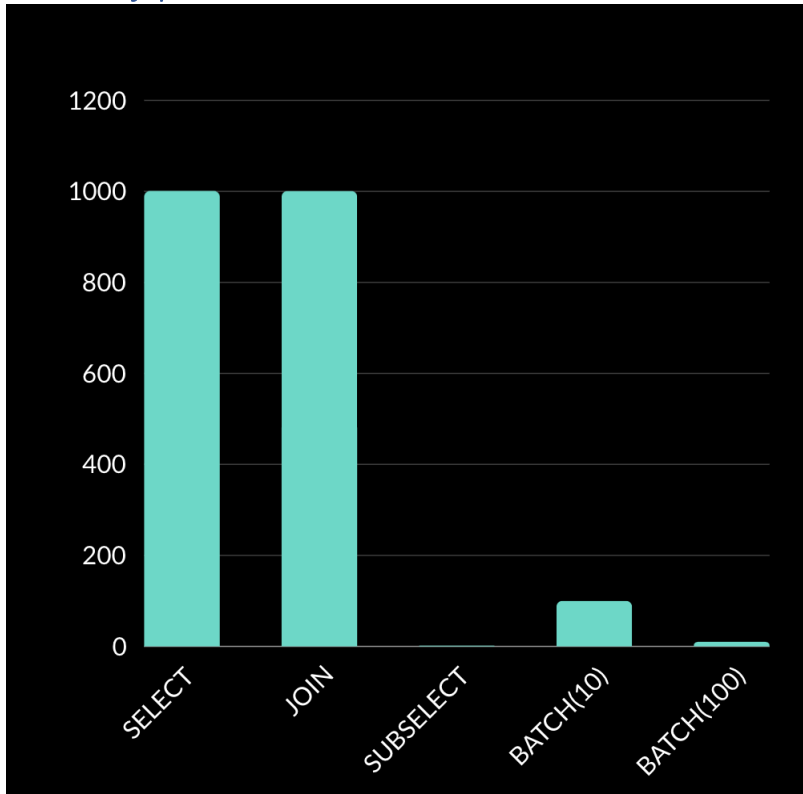
3. analysis

To analysis the detail, we will compare the total number of query, execution time, memory usage of 4 fetching strategies and compare with FetchType is EAGER.

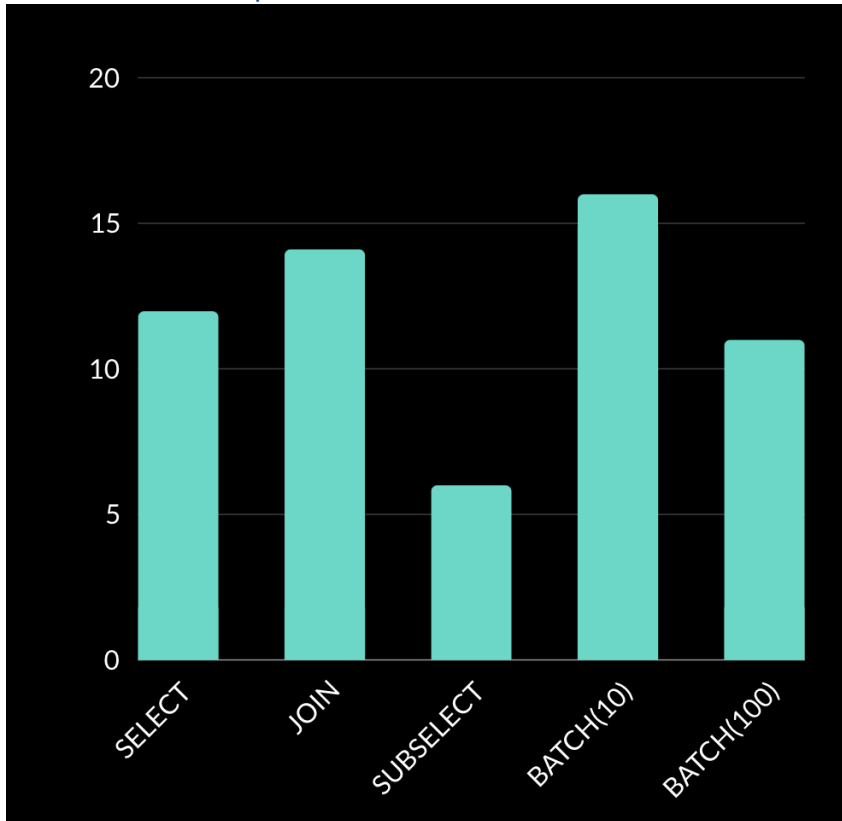
Table compare 4 fetching strategies:

	SELECT	JOIN	SUBSELECT	BATCH (size 10)	BATCH (size 100)
Number of queries	1001	1001	2	1000/10 = 100	1000/100 = 10
Execution time (s)	11.98	14.11s	6.02s	16.08s	10.90s
Memory usage (MB)	37.2MB	37.2MB	37.2MB	37.2MB	37.2MB

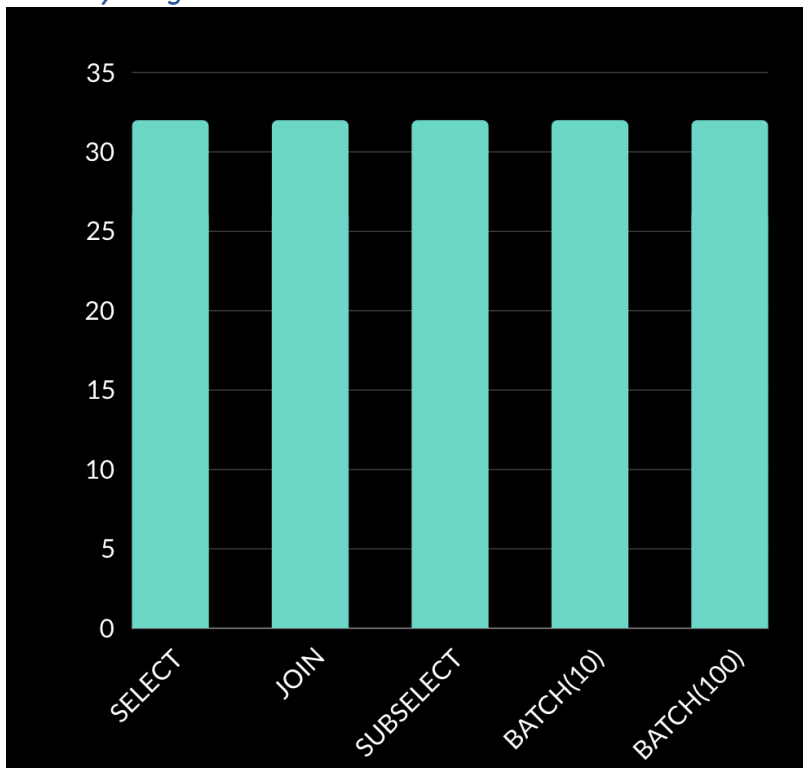
Number of queries chart



Execution time comparison chart



Memory usage chart



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, avoiding the N+1 query problem and results in better performance. Cannot fetch data lazily, fetches all associated entities eagerly, maybe performance issues if not need.
- **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.