Spring Data II – Lab 3 – Report

Overview of Fetching Strategies

1. Select Fetching Strategy

Implementation: This strategy uses individual SQL queries for each association requiring a fetch. The generated query for fetching products and their reviews :

select p1_0.id,p1_0.description,p1_0.name from product p1_0 (1)

select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.age,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=?; (1000)

Observation: Resulted in a total of 1001 SQL queries for fetching 500 products and their 500 associated reviews.

Performance: Execution time averaged 566ms, with a memory usage of 13MB. The data transfer size was 126.58KB.

Practical Use: Best suited for scenarios where associated entities are accessed infrequently, minimizing unnecessary data fetching.

2. Join Fetching Strategy

Implementation: This strategy uses SQL JOINs to fetch associated entities. The generated query for fetching products and their reviews :

select p1_0.id,p1_0.description,p1_0.name from product p1_0 (1)

select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.age,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=? (1000)

Observation: Resulted in a total of 1001 SQL queries for fetching 500 products and their 500 associated reviews.

Performance: Execution time averaged 470ms, with a higher memory usage of 20MB. The data transfer size remained the same at 126.58KB.

Practical Use: Ideal for situations where associated entities are always accessed together and the number of associations is relatively small.

3. Subselect Fetching Strategy

Implementation: Executes a main query for the parent entities and a separate subselect query for each association, like :

select p1_0.id,p1_0.description,p1_0.name from product p1_0 (1) select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.age,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) (1)

Observation: Only 2 SQL queries were executed for fetching 500 products and their 500 reviews.

Performance: The fastest with an execution time of 35ms and minimal memory usage of 4MB. The data transfer size was consistent at 126.58KB.

Practical Use: Most efficient when dealing with a variable number of entities or when minimizing the number of SQL queries is a priority.

4. Batch Fetching Strategy (Batch Size 100)

Implementation: Fetches associations in batches of a specified size, reducing the number of queries. The generated query :

select p1_0.id,p1_0.description,p1_0.name from product p1_0 (1)

select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.age,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 (?,?) (10)

Observation: Resulted in 11 SQL queries for fetching 500 products and their 500 reviews.

Performance: Good balance with an execution time of 50ms and memory usage of 4MB. Data transfer size was 126.58KB.

Practical Use: Suitable for optimizing the number of SQL queries while keeping memory usage low, especially when dealing with moderate amounts of associated data.

5. Batch Fetching Strategy (Batch Size 200)

Implementation: Similar to Batch(100) but fetches associations in larger batches. The generated query :

select p1_0.id,p1_0.description,p1_0.name from product p1_0 (1)

select r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,u1_0.id,u1_0.age,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 (?,?) (5)

Observation: Reduced the number of SQL queries to 6 for fetching the same data.

Performance: Comparable to Batch(100) with a slightly faster execution time of 36ms and the same memory usage.

Practical Use: An improved version of Batch(100) for scenarios with larger datasets, providing a good trade-off between execution time and resource usage.

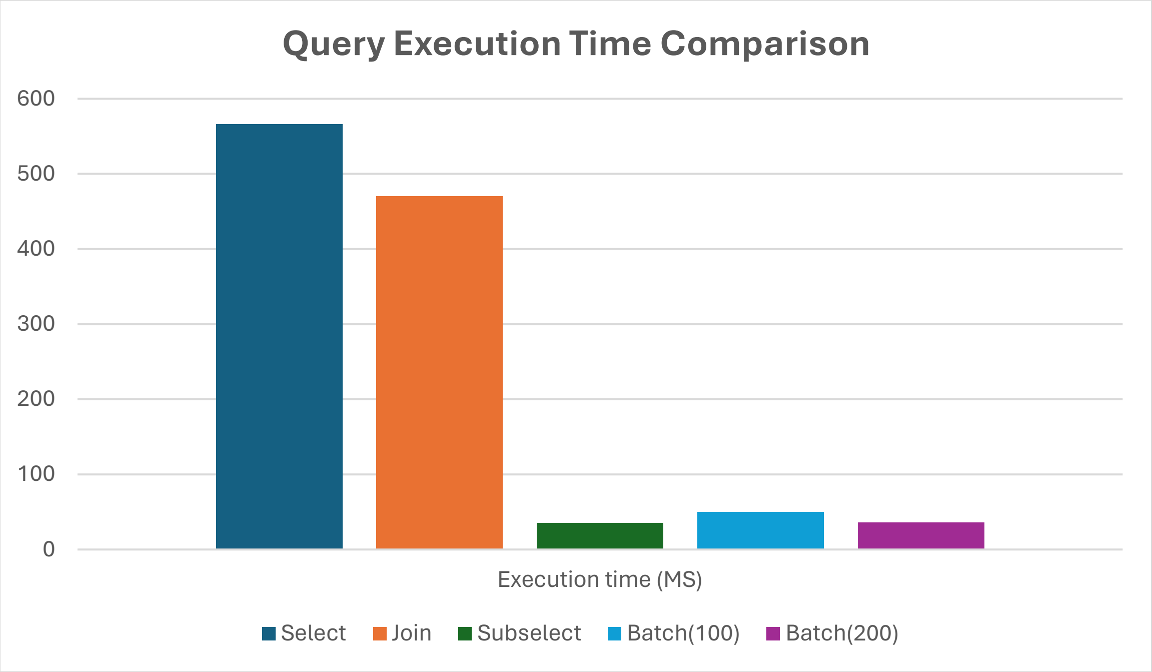
Detailed Analysis and Charts

For the detailed analysis, we will create charts to visually compare the execution time, number of SQL queries, memory usage, and data transfer size across different fetching strategies. This will help illustrate the trade-offs between them.

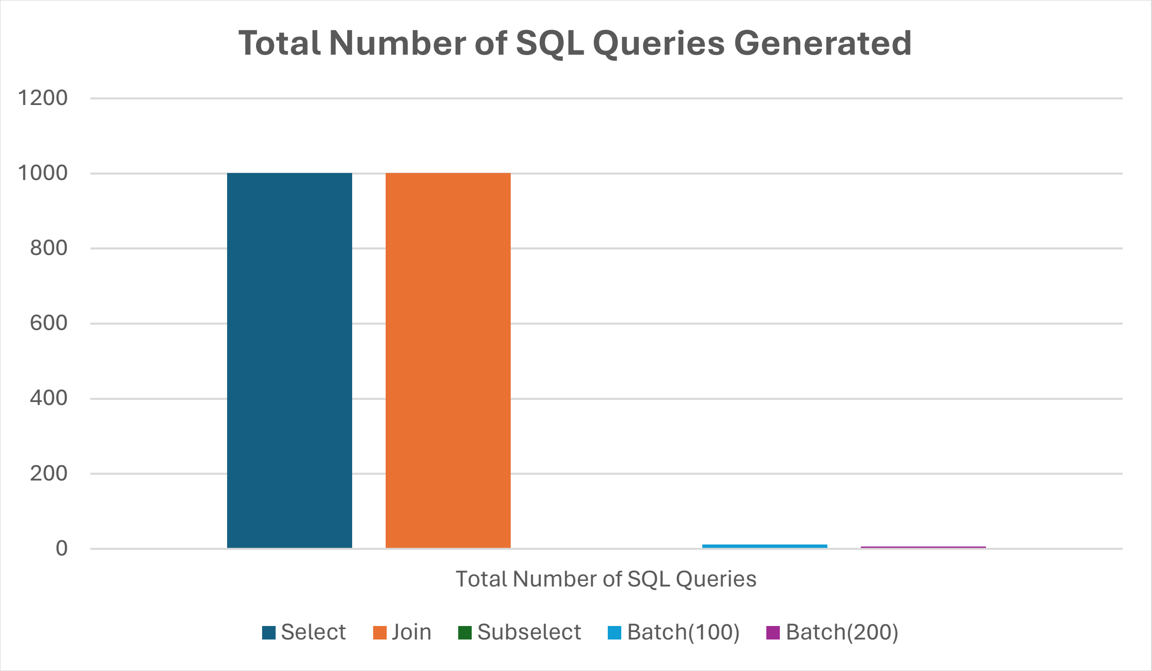
Here are the results we got form the experiment:

	Select	Join	Subselect	Batch (100)	Batch (200)
Memory usage (MB)	13	20	4	4	4
Execution time (MS)	566	470	35	50	36
Data transfer size (KB)	126.58	126.58	126.58	126.58	126.58
Total Number of SQL Queries	1001	1001	2	11	6

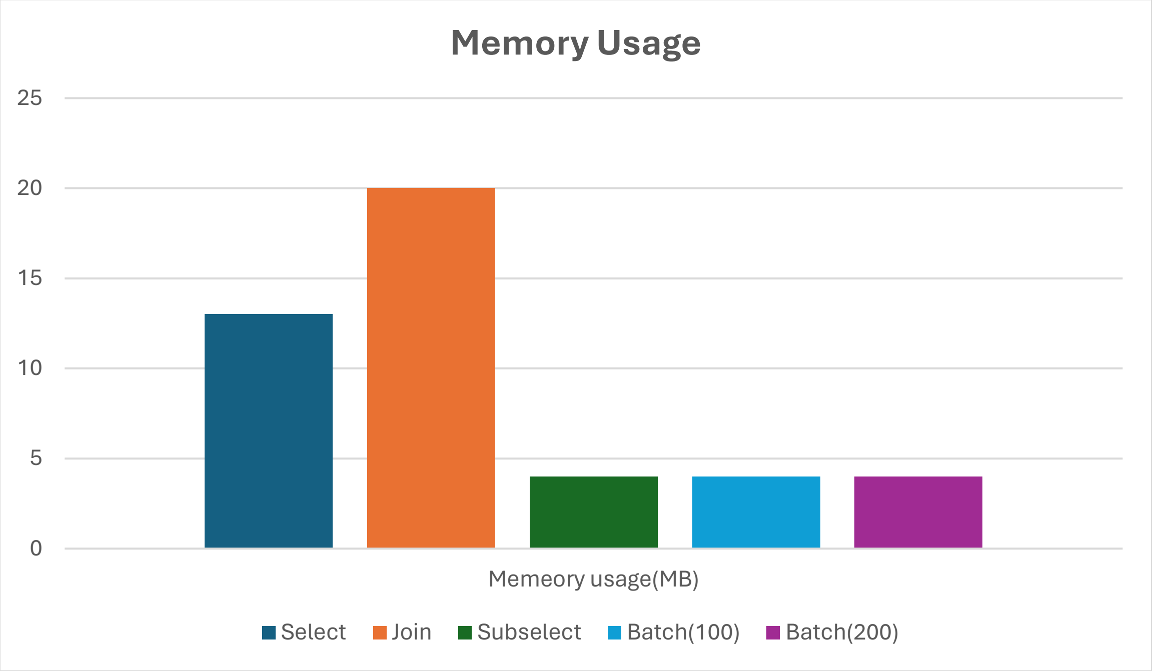
Query Execution Time Comparison: A bar chart displaying the average execution time for each fetching strategy.



Total Number of SQL Queries Generated: A bar chart showing the total number of SQL queries executed for each strategy.



Memory Usage: A bar chart showing the peak memory usage of each fetching strategy during execution.



Discussion on the Suitability of Each Fetching Strategy

- **Select Fetching Strategy** is suitable when individual associations are accessed infrequently, avoiding unnecessary data loading. However, it can lead to a high number of SQL queries.
- **Join Fetching Strategy** is ideal for cases where entities and their associations are always used together, although it can increase memory usage due to duplicate data in the result set.
- **Subselect Fetching Strategy** shines in scenarios where minimizing the number of SQL queries is crucial and the dataset size is manageable, offering a significant performance boost.
- **Batch Fetching Strategies** (both sizes) are versatile, balancing between reducing SQL queries and maintaining low memory usage. They are particularly effective when working with moderate to large datasets and a variable number of associations.

Conclusion

From this experiment, it's clear that no one strategy is superior in all scenarios. The choice of fetching strategy should be guided by the specific use case, including the size of the data, the nature of the associations between entities, and performance requirements.

Subselect and Batch fetching strategies generally offer the best balance between execution time, memory usage, and the number of SQL queries, making them suitable for a wide range of applications. However, the specific characteristics of your application data and access patterns should ultimately drive the decision on which strategy to implement.

By carefully selecting the appropriate fetching strategy, developers can significantly improve the performance and efficiency of their applications, enhancing both user experience and resource utilization.