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)

```java
// fetch strategy: LAZY and Subselect
@OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
@JsonManagedReference
@Fetch(FetchMode.SUBSELECT)
private List<Review> reviews;
```

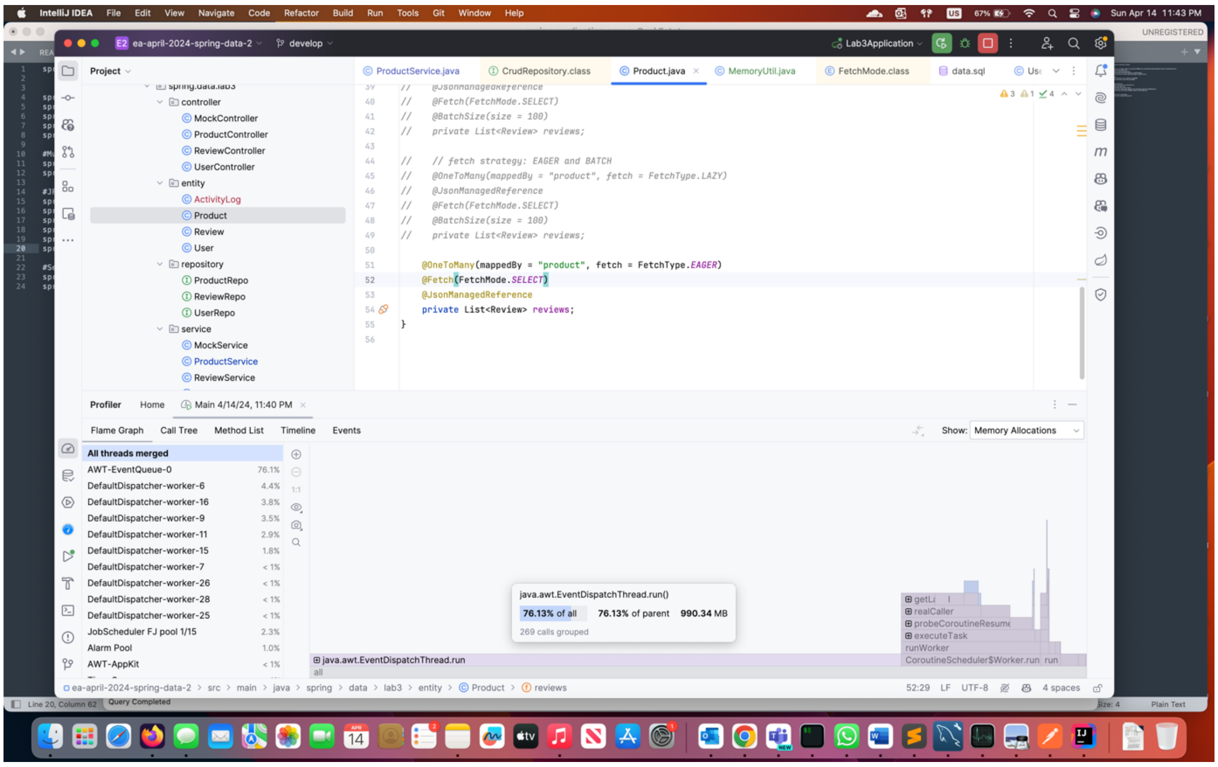
For instanse, with BATCH: @Fetch(value = FetchMode.BATCH)

```java
@OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
@JsonManagedReference
@Fetch(FetchMode.SELECT)
@BatchSize(size = 100)
private List<Review> reviews;
```

2. overview of the implemented fetching strategies.

# FetchMode SUBSELECT

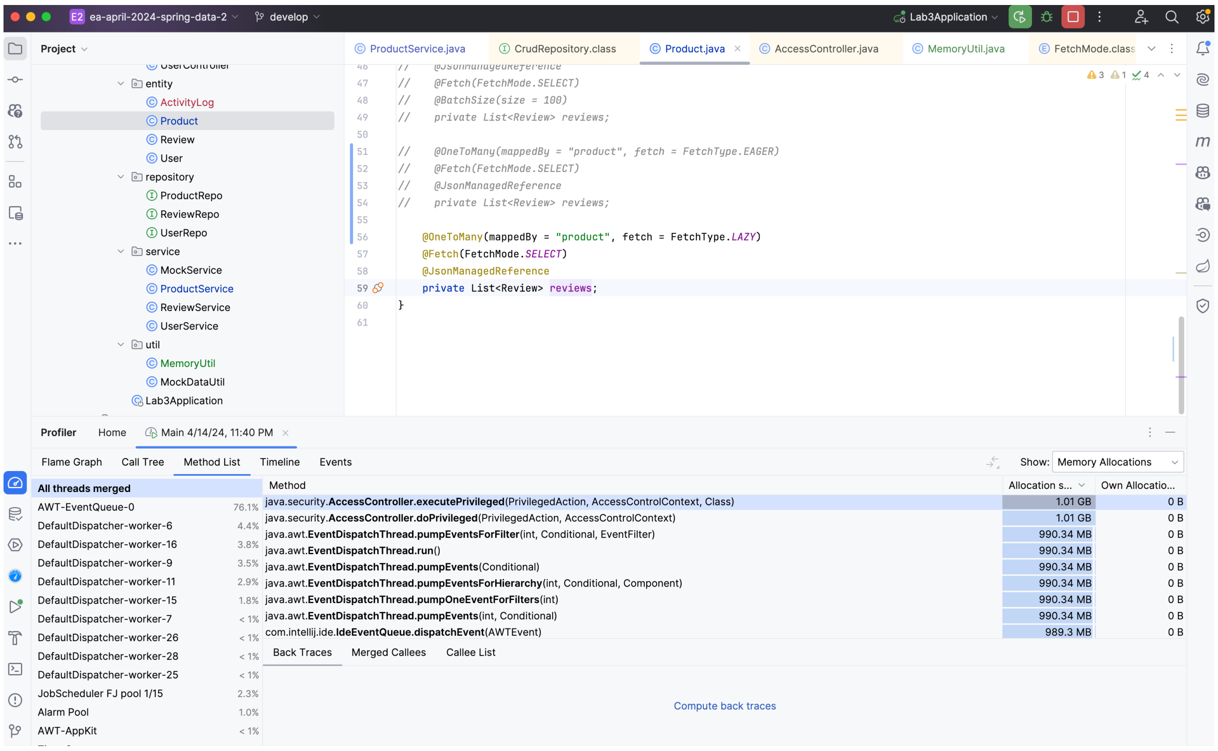
EAGER:



```java
//    @JsonManagedReference
//    @Fetch(FetchMode.SELECT)
//    @BatchSize(size = 100)
//    private List<Review> reviews;

//    // fetch strategy: EAGER and BATCH
//    @OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
//    @JsonManagedReference
//    @Fetch(FetchMode.SELECT)
//    @BatchSize(size = 100)
//    private List<Review> reviews;

    @OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
    @Fetch(FetchMode.SELECT)
    @JsonManagedReference
    private List<Review> reviews;
}
```

```
Profiler   Home   Main 4/14/24, 11:40 PM
Flame Graph   Call Tree   Method List   Timeline   Events        Show: Memory Allocations

All threads merged
AWT-EventQueue-0                  76.1%
DefaultDispatcher-worker-6         4.4%
DefaultDispatcher-worker-16        3.8%
DefaultDispatcher-worker-9         3.5%
DefaultDispatcher-worker-11        2.9%
DefaultDispatcher-worker-15        1.8%
DefaultDispatcher-worker-7         < 1%
DefaultDispatcher-worker-26        < 1%
DefaultDispatcher-worker-28        < 1%
DefaultDispatcher-worker-25        < 1%
JobScheduler FJ pool 1/15          2.3%
Alarm Pool                         1.0%
AWT-AppKit                         < 1%

java.awt.EventDispatchThread.run()
76.13% of all   76.13% of parent   990.34 MB
269 calls grouped
```

LAZY:

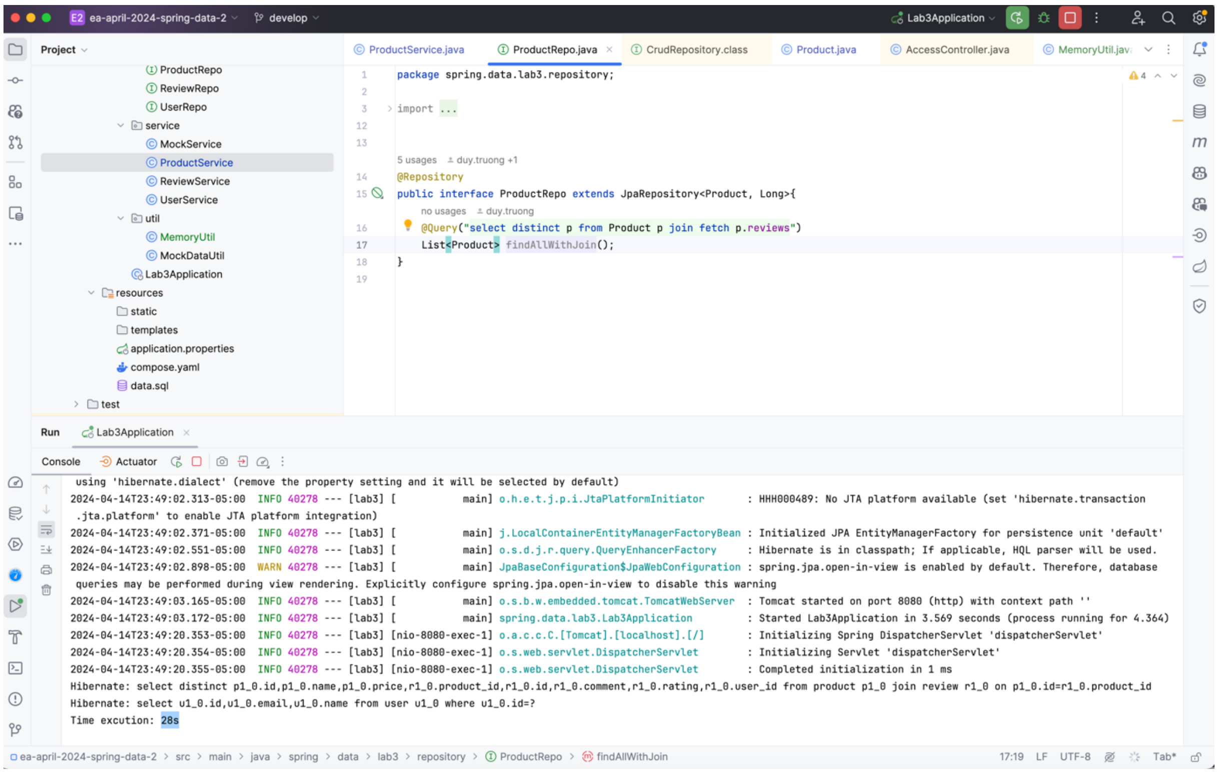


```java
//    @Fetch(FetchMode.SELECT)
//    @BatchSize(size = 100)
//    private List<Review> reviews;

//    @OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
//    @Fetch(FetchMode.SELECT)
//    @JsonManagedReference
//    private List<Review> reviews;

    @OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
    @Fetch(FetchMode.SELECT)
    @JsonManagedReference
    private List<Review> reviews;
}
```

All threads merged	
AWT-EventQueue-0	76.1%
DefaultDispatcher-worker-6	4.4%
DefaultDispatcher-worker-16	3.8%
DefaultDispatcher-worker-9	3.5%
DefaultDispatcher-worker-11	2.9%
DefaultDispatcher-worker-15	1.8%
DefaultDispatcher-worker-7	< 1%
DefaultDispatcher-worker-26	< 1%
DefaultDispatcher-worker-28	< 1%
DefaultDispatcher-worker-25	< 1%
JobScheduler FJ pool 1/15	2.3%
Alarm Pool	1.0%
AWT-AppKit	< 1%

Method	Allocation s...	Own Allocatio...
java.security.**AccessController.executePrivileged**(PrivilegedAction, AccessControlContext, Class)	1.01 GB	0 B
java.security.**AccessController.doPrivileged**(PrivilegedAction, AccessControlContext)	1.01 GB	0 B
java.awt.**EventDispatchThread.pumpEventsForFilter**(int, Conditional, EventFilter)	990.34 MB	0 B
java.awt.**EventDispatchThread.run**()	990.34 MB	0 B
java.awt.**EventDispatchThread.pumpEvents**(Conditional)	990.34 MB	0 B
java.awt.**EventDispatchThread.pumpEventsForHierarchy**(int, Conditional, Component)	990.34 MB	0 B
java.awt.**EventDispatchThread.pumpOneEventForFilters**(int)	990.34 MB	0 B
java.awt.**EventDispatchThread.pumpEvents**(int, Conditional)	990.34 MB	0 B
com.intellij.ide.**IdeEventQueue.dispatchEvent**(AWTEvent)	989.3 MB	0 B

#FetchMode JOIN:

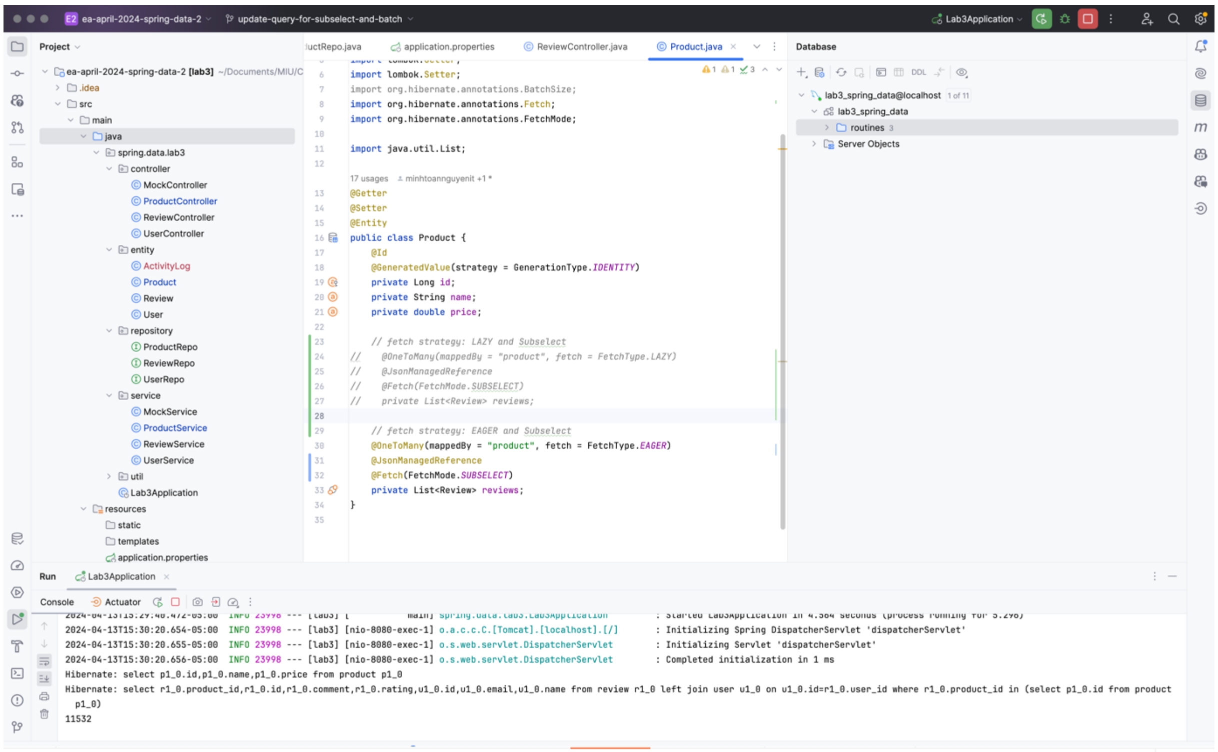


```java
package spring.data.lab3.repository;

import ...


5 usages  duy.truong +1
@Repository
public interface ProductRepo extends JpaRepository<Product, Long>{
    no usages  duy.truong
    @Query("select distinct p from Product p join fetch p.reviews")
    List<Product> findAllWithJoin();
}
```

```
 using 'hibernate.dialect' (remove the property setting and it will be selected by default)
2024-04-14T23:49:02.313-05:00  INFO 40278 --- [lab3] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction
 .jta.platform' to enable JTA platform integration)
2024-04-14T23:49:02.371-05:00  INFO 40278 --- [lab3] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-04-14T23:49:02.551-05:00  INFO 40278 --- [lab3] [           main] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-04-14T23:49:02.898-05:00  WARN 40278 --- [lab3] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database
 queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-04-14T23:49:03.165-05:00  INFO 40278 --- [lab3] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-14T23:49:03.172-05:00  INFO 40278 --- [lab3] [           main] spring.data.lab3.Lab3Application         : Started Lab3Application in 3.569 seconds (process running for 4.364)
2024-04-14T23:49:20.353-05:00  INFO 40278 --- [lab3] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-14T23:49:20.354-05:00  INFO 40278 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-14T23:49:20.355-05:00  INFO 40278 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
Hibernate: select distinct p1_0.id,p1_0.name,p1_0.price,r1_0.product_id,r1_0.id,r1_0.comment,r1_0.rating,r1_0.user_id from product p1_0 join review r1_0 on p1_0.id=r1_0.product_id
Hibernate: select u1_0.id,u1_0.email,u1_0.name from user u1_0 where u1_0.id=?
Time excution: 28s
```

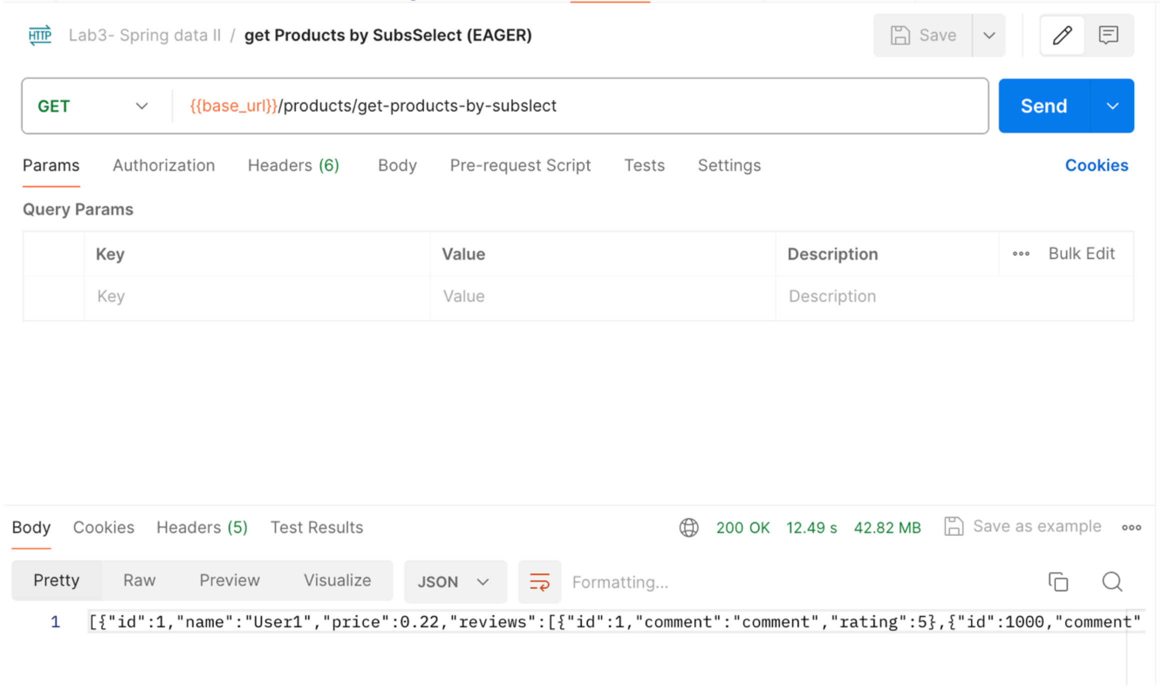
#FetchMode SUBSELECT

EAGER:



```java
import lombok.Setter;
import org.hibernate.annotations.BatchSize;
import org.hibernate.annotations.Fetch;
import org.hibernate.annotations.FetchMode;

import java.util.List;

@Getter
@Setter
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;

    // fetch strategy: LAZY and Subselect
//    @OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
//    @JsonManagedReference
//    @Fetch(FetchMode.SUBSELECT)
//    private List<Review> reviews;

    // fetch strategy: EAGER and Subselect
    @OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
    @JsonManagedReference
    @Fetch(FetchMode.SUBSELECT)
    private List<Review> reviews;
}
```

```
2024-04-13T15:29:40.472-05:00  INFO 23998 --- [lab3] [           main] spring.data.lab3.Lab3Application          : Started Lab3Application in 4.564 seconds (process running for 5.298)
2024-04-13T15:30:20.654-05:00  INFO 23998 --- [lab3] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-13T15:30:20.655-05:00  INFO 23998 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-13T15:30:20.656-05:00  INFO 23998 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 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 from product
  p1_0)
11532
```



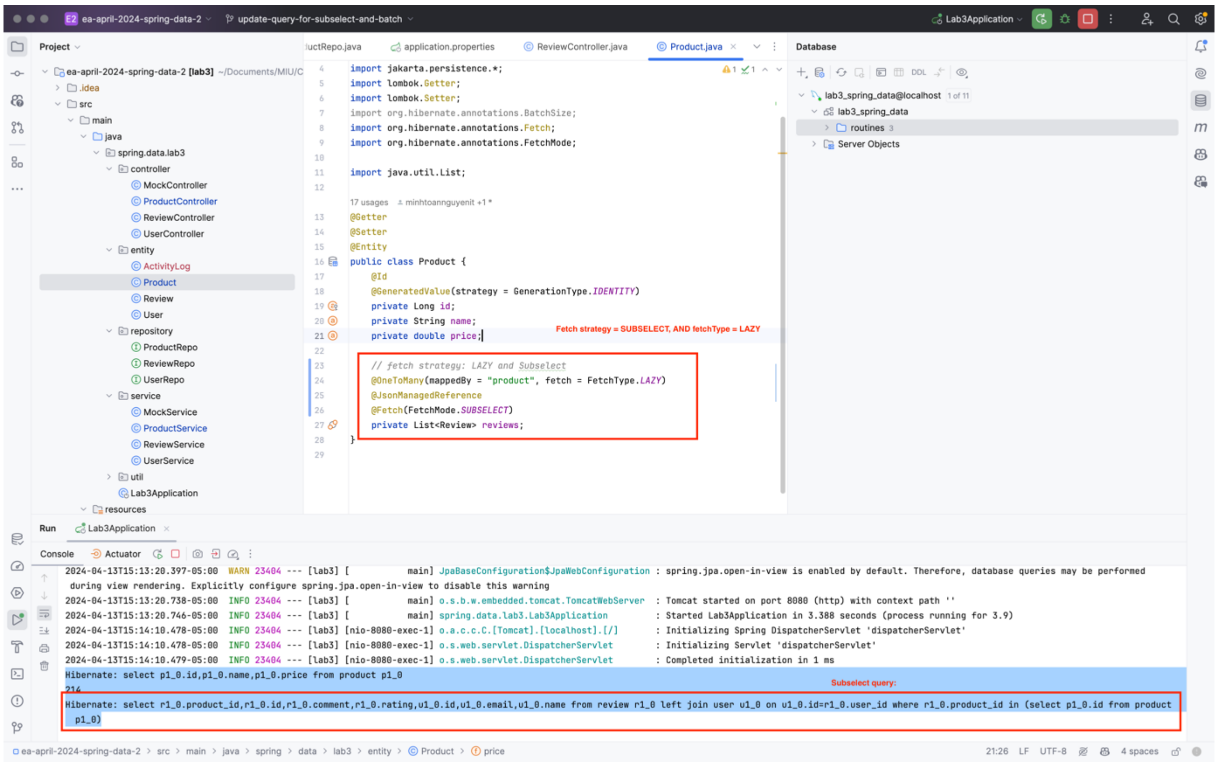
Lab3- Spring data II / get Products by SubsSelect (EAGER)

GET {{base_url}}/products/get-products-by-subslect

200 OK 12.49 s 42.82 MB

```
1  [{"id":1,"name":"User1","price":0.22,"reviews":[{"id":1,"comment":"comment","rating":5},{"id":1000,"comment"
```

LAZY:

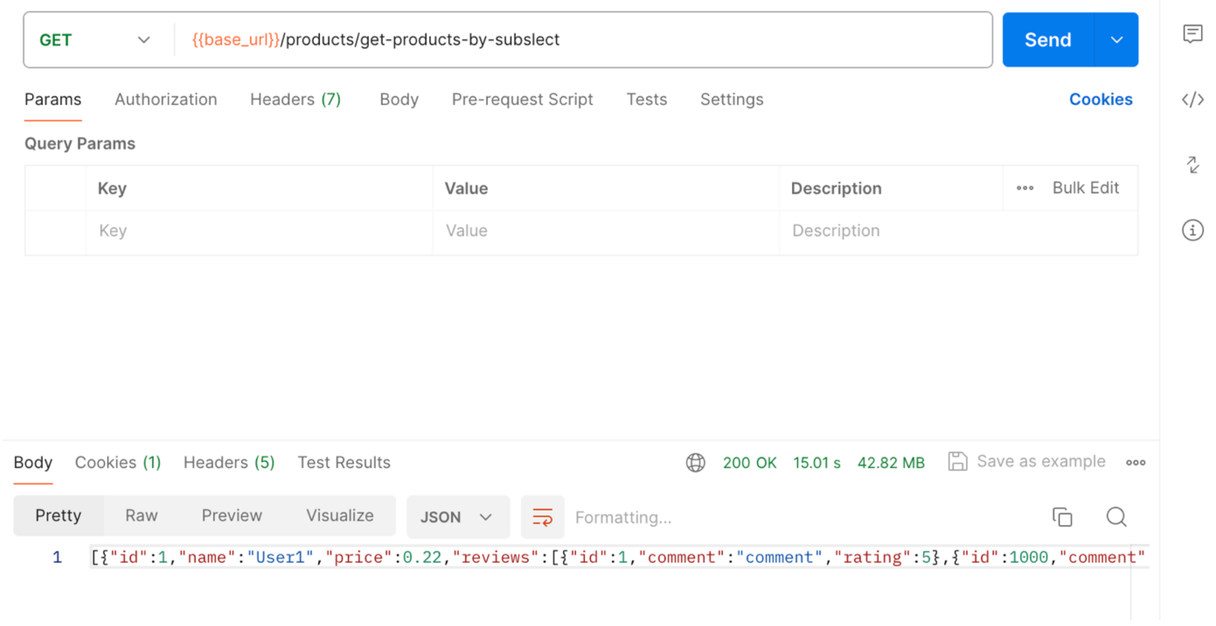


```java
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.BatchSize;
import org.hibernate.annotations.Fetch;
import org.hibernate.annotations.FetchMode;

import java.util.List;

@Getter
@Setter
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;

    // fetch strategy: LAZY and Subselect
    @OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
    @JsonManagedReference
    @Fetch(FetchMode.SUBSELECT)
    private List<Review> reviews;
}
```

Fetch strategy = SUBSELECT, AND fetchType = LAZY

```
2024-04-13T15:13:20.397-05:00  WARN 23404 --- [lab3] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed
 during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-04-13T15:13:20.738-05:00  INFO 23404 --- [lab3] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T15:13:20.746-05:00  INFO 23404 --- [lab3] [           main] spring.data.lab3.Lab3Application         : Started Lab3Application in 3.388 seconds (process running for 3.9)
2024-04-13T15:14:10.478-05:00  INFO 23404 --- [lab3] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-13T15:14:10.478-05:00  INFO 23404 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-13T15:14:10.479-05:00  INFO 23404 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
Hibernate: select p1_0.id,p1_0.name,p1_0.price from product p1_0
214
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 from product
 p1_0)
```

Subselect query:



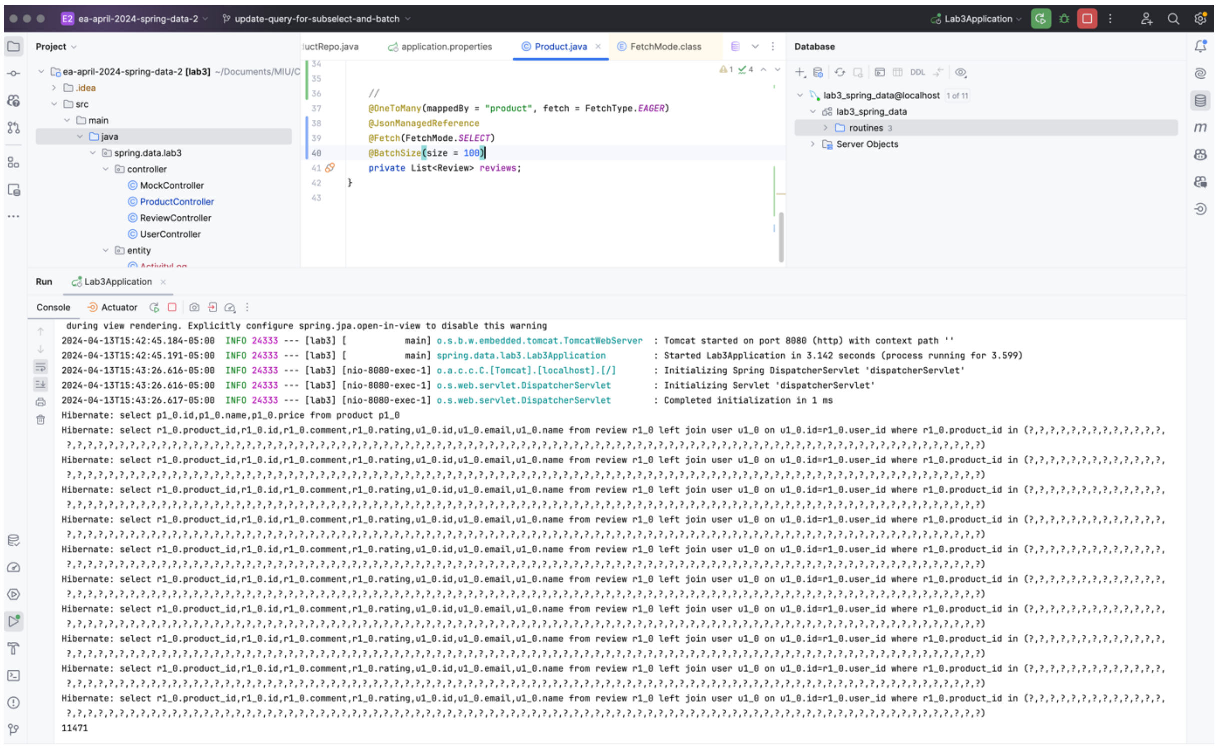
```
GET {{base_url}}/products/get-products-by-subslect
```

200 OK 15.01 s 42.82 MB

```
1  [{"id":1,"name":"User1","price":0.22,"reviews":[{"id":1,"comment":"comment","rating":5},{"id":1000,"comment"
```

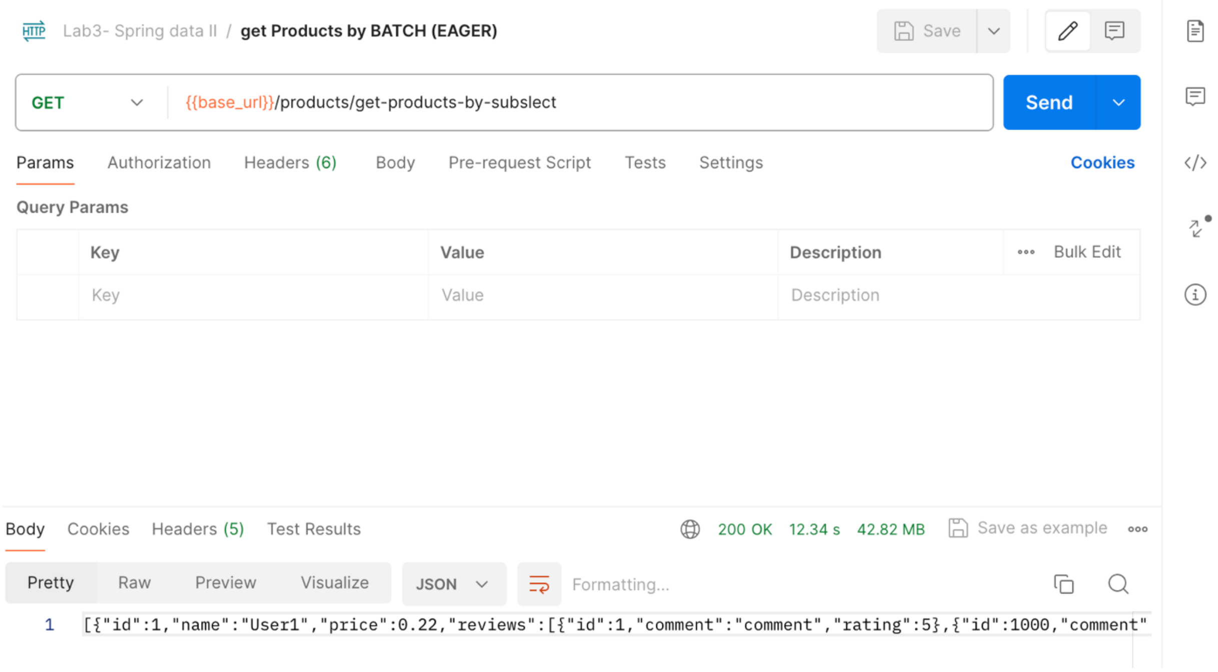
#FetchMode: BATCH

EAGER:



```java
    //
    @OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
    @JsonManagedReference
    @Fetch(FetchMode.SELECT)
    @BatchSize(size = 100)
    private List<Review> reviews;
}
```

```
 during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-04-13T15:42:45.184-05:00  INFO 24333 --- [lab3] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T15:42:45.191-05:00  INFO 24333 --- [lab3] [           main] spring.data.lab3.Lab3Application         : Started Lab3Application in 3.142 seconds (process running for 3.599)
2024-04-13T15:43:26.616-05:00  INFO 24333 --- [lab3] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-13T15:43:26.616-05:00  INFO 24333 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-13T15:43:26.617-05:00  INFO 24333 --- [lab3] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
11471
```

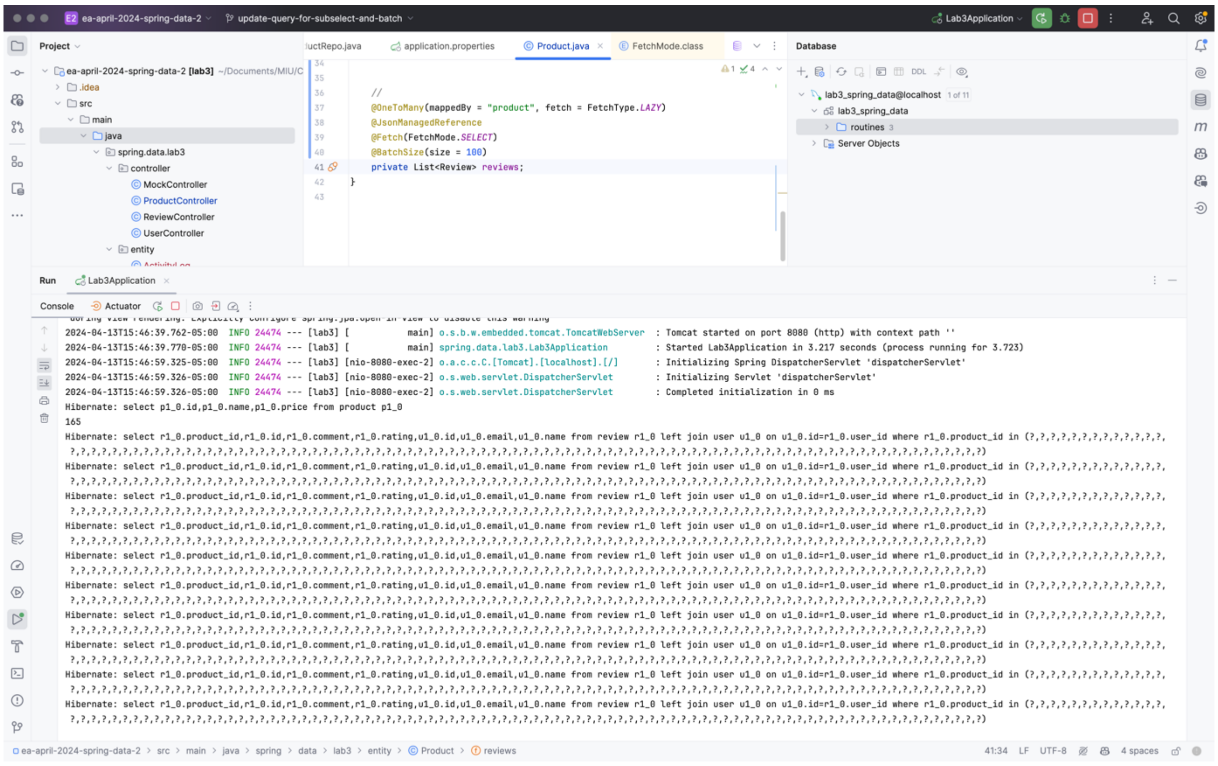


GET {{base_url}}/products/get-products-by-subslect

200 OK 12.34 s 42.82 MB

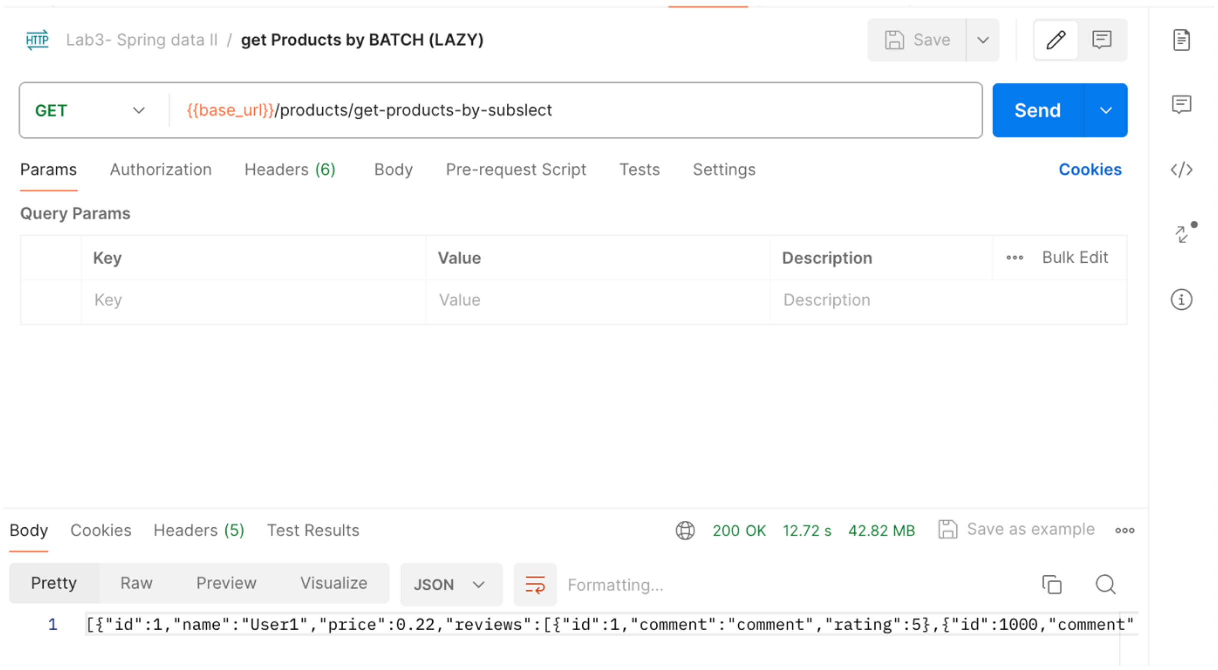
```
1  [{"id":1,"name":"User1","price":0.22,"reviews":[{"id":1,"comment":"comment","rating":5},{"id":1000,"comment"
```

LAZY:



```java
    //
    @OneToMany(mappedBy = "product", fetch = FetchType.LAZY)
    @JsonManagedReference
    @Fetch(FetchMode.SELECT)
    @BatchSize(size = 100)
    private List<Review> reviews;
}
```

```
2024-04-13T15:46:39.762-05:00  INFO 24474 --- [lab3] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T15:46:39.770-05:00  INFO 24474 --- [lab3] [           main] spring.data.lab3.Lab3Application         : Started Lab3Application in 3.217 seconds (process running for 3.723)
2024-04-13T15:46:59.325-05:00  INFO 24474 --- [lab3] [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-13T15:46:59.326-05:00  INFO 24474 --- [lab3] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-13T15:46:59.326-05:00  INFO 24474 --- [lab3] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
Hibernate: select p1_0.id,p1_0.name,p1_0.price from product p1_0
165
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
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 (?,?,?,?,?,?,?,?,?,?,?,?,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
```



Lab3- Spring data II / get Products by BATCH (LAZY)

GET {{base_url}}/products/get-products-by-subslect

Query Params

Key	Value	Description
Key	Value	Description

200 OK 12.72 s 42.82 MB

```
1   [{"id":1,"name":"User1","price":0.22,"reviews":[{"id":1,"comment":"comment","rating":5},{"id":1000,"comment"
```

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.