N+1 Problem in JPA Queries

Bad Repository Code (N+1 Problem)

Swipe in →

```
List<Order> orders = orderRepository.findAll();
for (Order order : orders) {
  // Triggers extra query per order!
     System.out.println(order.getItems().size());
}
```

JPA Queries

What is the N+1 Problem?

First, you ask: “Give me the list of all customers” → ✅ 1 query

Then for each customer, you ask: “What did this customer buy?” → ❌ 1 query per customer

👉 **So if you have 10 customers, you end up doing:**

1 (for customers) + 10 (for their products) = 11 queries total

💣 That’s the N+1 Problem!

N = Number of Parent Records (Like Customers)

You do 1 query to get parents, & N Extra queries to get their children

Performance gets Worse as Data Grows.

Entity Design

Let's Start with this Entity Design - To Understand

```
@Entity
public class Order {
    @Id
    private Long id;

    @OneToMany(mappedBy = "order", fetch = FetchType.LAZY)
    private List<OrderItem> items;
}
```

```
@Entity
public class OrderItem {
    @Id
    private Long id;

    @ManyToOne
    private Order order;
}
```

Bad Repository Code (N+1 Problem):

```
List<Order> orders = orderRepository.findAll();
for (Order order : orders) {
    System.out.println(order.getItems().size()); // Triggers extra query per order!
}
```

What Happens Behind the Scenes?

findAll() → 1 query

getItems() (lazy) → fires 1 query per Order

Total queries = 1 (orders) + N (items)

Real-World Impact:

Orders Fetched	Queries Executed
10 Orders	11 Queries
100 Orders	101 Queries
1000 Orders	1001 Queries

Solution #1: Use JOIN FETCH

```
@Query("SELECT o FROM Order o JOIN FETCH o.items")
List<Order> findAllWithItems();
```

Solves the N+1 by fetching all OrderItems in **1 JOIN query**

Eager loading with control — avoids overfetching

@EntityGraph

Solution #2: Use @EntityGraph

@EntityGraph tells JPA: "Please fetch these related entities eagerly, but smartly – without falling into the N+1 trap."

```
@EntityGraph(attributePaths = {"items"})
@Query("SELECT o FROM Order o")
List<Order> findAllWithItems();
```

1. Cleaner & reusable
2. Declarative approach with less boilerplate
3. Works well with Spring Data JPA repositories

Detect N+1

Keep Watching Your Mistakes

Mistake	Impact
Using fetch = **FetchType.EAGER** by default	**Overfetching**, poor performance
Ignoring Hibernate **SQL** logs	Misses **query** explosion issues
Not indexing **FK** columns	Slows down **JOIN** performance

How to Detect N+1 Issues ?

Enable SQL logging: application.yml

```
spring.jpa.properties.hibernate.show_sql: true
spring.jpa.properties.hibernate.format_sql: true
logging.level.org.hibernate.SQL: DEBUG
logging.level.org.hibernate.type.descriptor.sql: TRACE
```