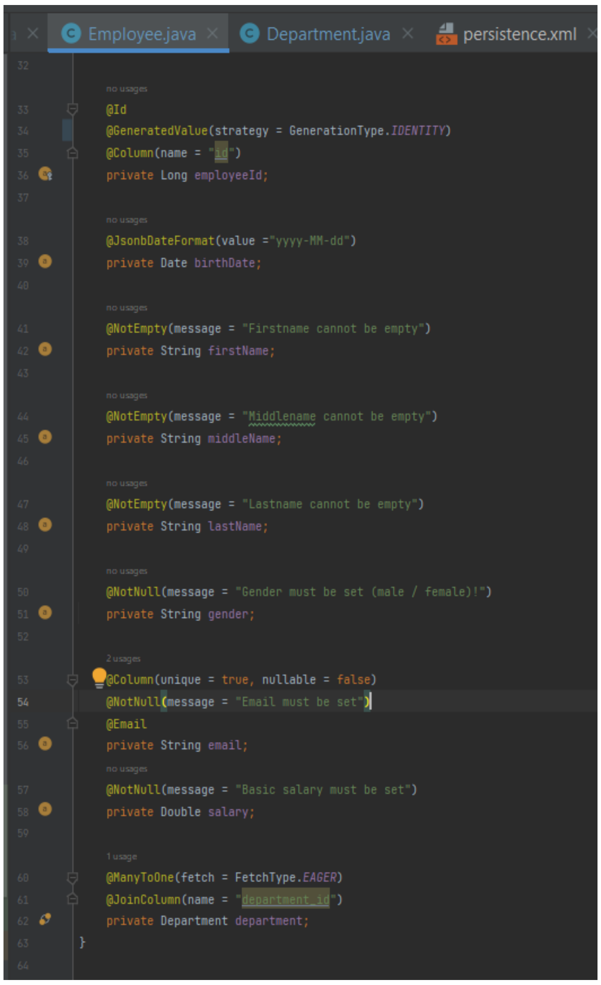
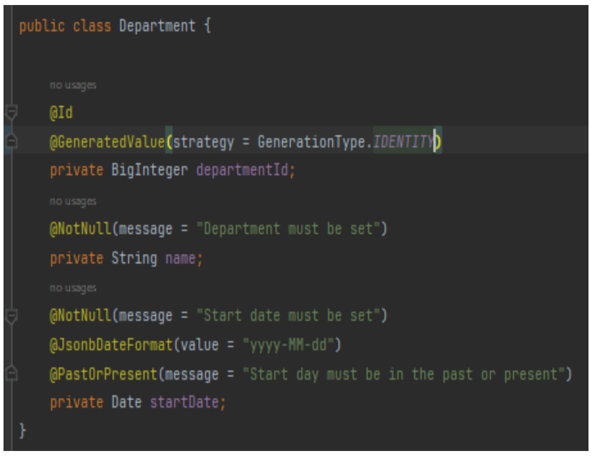
Lab: N + 1 Problem and Solving



Employee.java | Department.java | persistence.xml

```java
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long employeeId;

    no usages
    @JsonbDateFormat(value ="yyyy-MM-dd")
    private Date birthDate;

    no usages
    @NotEmpty(message = "Firstname cannot be empty")
    private String firstName;

    no usages
    @NotEmpty(message = "Middlename cannot be empty")
    private String middleName;

    no usages
    @NotEmpty(message = "Lastname cannot be empty")
    private String lastName;

    no usages
    @NotNull(message = "Gender must be set (male / female)!")
    private String gender;

    2 usages
    @Column(unique = true, nullable = false)
    @NotNull(message = "Email must be set")
    @Email
    private String email;
    no usages
    @NotNull(message = "Basic salary must be set")
    private Double salary;

    1 usage
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "department_id")
    private Department department;
}
```



```java
public class Department {

    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private BigInteger departmentId;
    no usages
    @NotNull(message = "Department must be set")
    private String name;
    no usages
    @NotNull(message = "Start date must be set")
    @JsonbDateFormat(value = "yyyy-MM-dd")
    @PastOrPresent(message = "Start day must be in the past or present")
    private Date startDate;
}
```

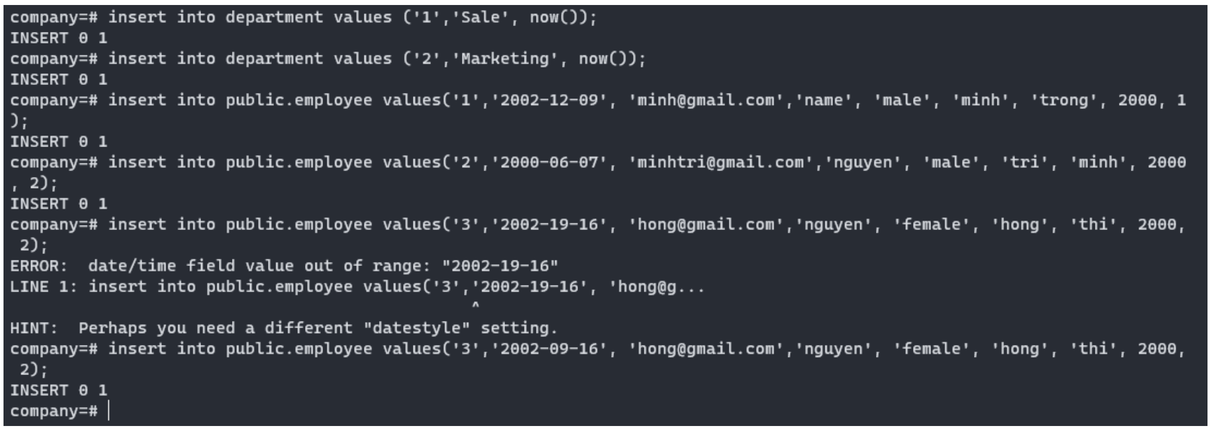
And the query:

```java
@NamedQuery(name = Employee.FIND_ALL, query = "SELECT e FROM Employee  e"),
```

2. Setting up data & the problem

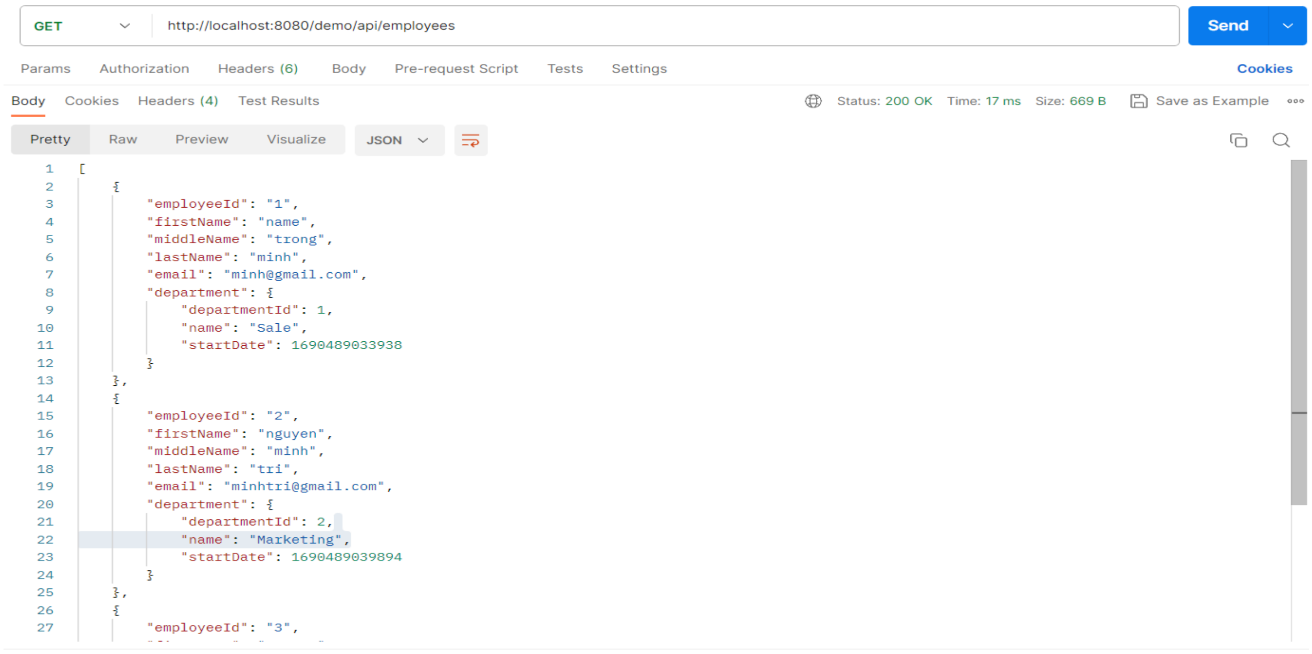
I insert into the postgres, 2 departments (using ternimal command):

- 2 Departments: Sale & Marketing.
- 3 Employees: The first employee is in Sale department and the rest belong to Marketing.



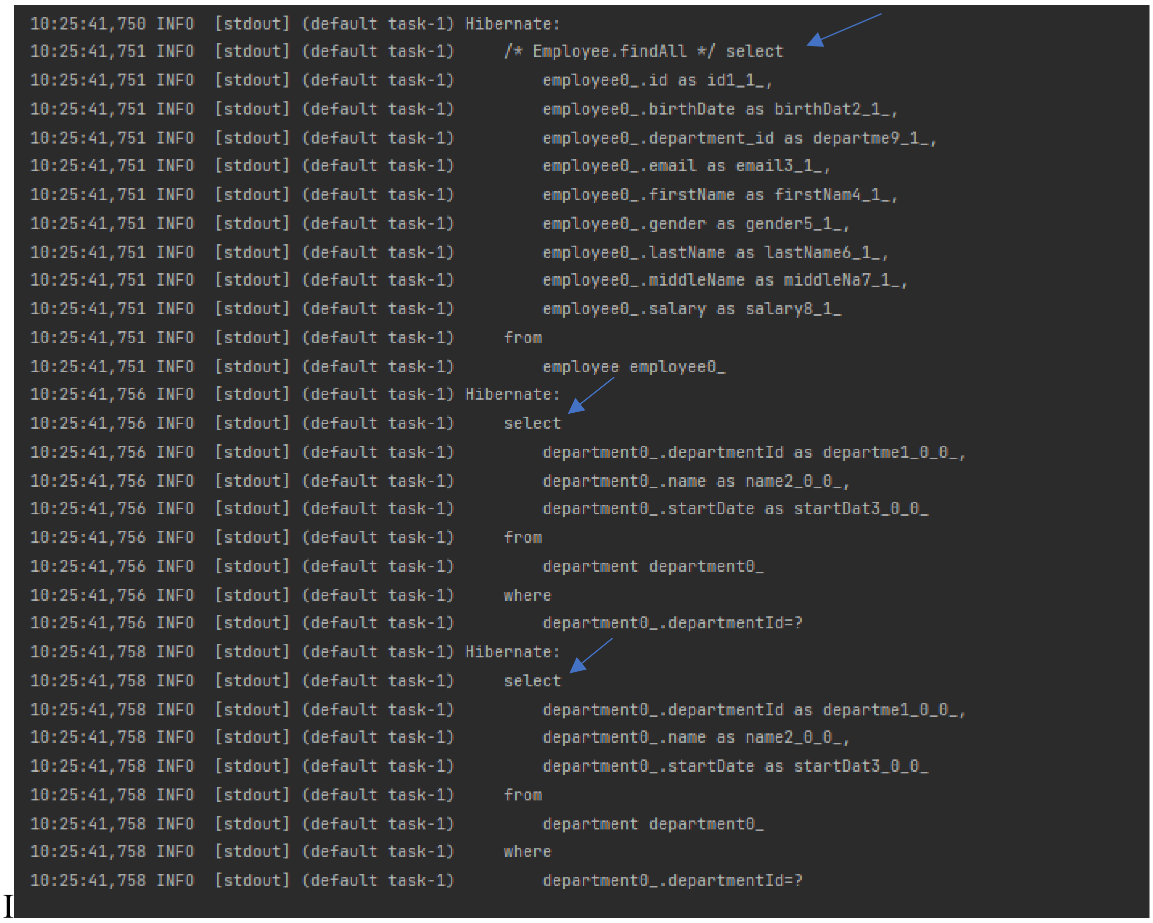
```
company=# insert into department values ('1','Sale', now());
INSERT 0 1
company=# insert into department values ('2','Marketing', now());
INSERT 0 1
company=# insert into public.employee values('1','2002-12-09', 'minh@gmail.com','name', 'male', 'minh', 'trong', 2000, 1
);
INSERT 0 1
company=# insert into public.employee values('2','2000-06-07', 'minhtri@gmail.com','nguyen', 'male', 'tri', 'minh', 2000
, 2);
INSERT 0 1
company=# insert into public.employee values('3','2002-19-16', 'hong@gmail.com','nguyen', 'female', 'hong', 'thi', 2000,
 2);
ERROR:  date/time field value out of range: "2002-19-16"
LINE 1: insert into public.employee values('3','2002-19-16', 'hong@g...
                                               ^
HINT:  Perhaps you need a different "datestyle" setting.
company=# insert into public.employee values('3','2002-09-16', 'hong@gmail.com','nguyen', 'female', 'hong', 'thi', 2000,
 2);
INSERT 0 1
company=# 
```

Then, I use postman to send a request to get AllEmployees in company:



```
[
    {
        "employeeId": "1",
        "firstName": "name",
        "middleName": "trong",
        "lastName": "minh",
        "email": "minh@gmail.com",
        "department": {
            "departmentId": 1,
            "name": "Sale",
            "startDate": 1690489033938
        }
    },
    {
        "employeeId": "2",
        "firstName": "nguyen",
        "middleName": "minh",
        "lastName": "tri",
        "email": "minhtri@gmail.com",
        "department": {
            "departmentId": 2,
            "name": "Marketing",
            "startDate": 1690489039894
        }
    },
    {
        "employeeId": "3",
```

Now, I’m going to check the log file after send this request in the Wildfly server:



```
10:25:41,750 INFO  [stdout] (default task-1) Hibernate:
10:25:41,751 INFO  [stdout] (default task-1)     /* Employee.findAll */ select
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.id as id1_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.birthDate as birthDat2_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.department_id as departme9_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.email as email3_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.firstName as firstNam4_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.gender as gender5_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.lastName as lastName6_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.middleName as middleNa7_1_,
10:25:41,751 INFO  [stdout] (default task-1)         employee0_.salary as salary8_1_
10:25:41,751 INFO  [stdout] (default task-1)     from
10:25:41,751 INFO  [stdout] (default task-1)         employee employee0_
10:25:41,756 INFO  [stdout] (default task-1) Hibernate:
10:25:41,756 INFO  [stdout] (default task-1)     select
10:25:41,756 INFO  [stdout] (default task-1)         department0_.departmentId as departme1_0_0_,
10:25:41,756 INFO  [stdout] (default task-1)         department0_.name as name2_0_0_,
10:25:41,756 INFO  [stdout] (default task-1)         department0_.startDate as startDat3_0_0_
10:25:41,756 INFO  [stdout] (default task-1)     from
10:25:41,756 INFO  [stdout] (default task-1)         department department0_
10:25:41,756 INFO  [stdout] (default task-1)     where
10:25:41,756 INFO  [stdout] (default task-1)         department0_.departmentId=?
10:25:41,758 INFO  [stdout] (default task-1) Hibernate:
10:25:41,758 INFO  [stdout] (default task-1)     select
10:25:41,758 INFO  [stdout] (default task-1)         department0_.departmentId as departme1_0_0_,
10:25:41,758 INFO  [stdout] (default task-1)         department0_.name as name2_0_0_,
10:25:41,758 INFO  [stdout] (default task-1)         department0_.startDate as startDat3_0_0_
10:25:41,758 INFO  [stdout] (default task-1)     from
10:25:41,758 INFO  [stdout] (default task-1)         department department0_
10:25:41,758 INFO  [stdout] (default task-1)     where
10:25:41,758 INFO  [stdout] (default task-1)         department0_.departmentId=?
```

I

- You can see that I just want to get all the employees but I got 3 select queries: 1 select in employee table **(need)** and **2 selects in department table**(*) **that don't need.**
- Notice that we have 2 departments (Sale and Marketing) in DB -> so the (*) will be 2. If you have 3 departments → the (*) will be 3 → So this is N
- From all the evidences above, we have the formular N + 1 queries (with N = number of relatived entites or number of departments, in this case).

⇨ This is the N + 1 Problem

3. Why this is SERIOUS:

The "N + 1 problem" is considered serious in applications or websites because it can lead to significant performance issues and inefficiencies in database queries. When this problem occurs, it results in a large number of additional queries being executed against the database, which can negatively impact the application's performance, especially in scenarios with a large dataset.

SOLUTIONS

*The goal is to change from N + 1 down to only 1 query.

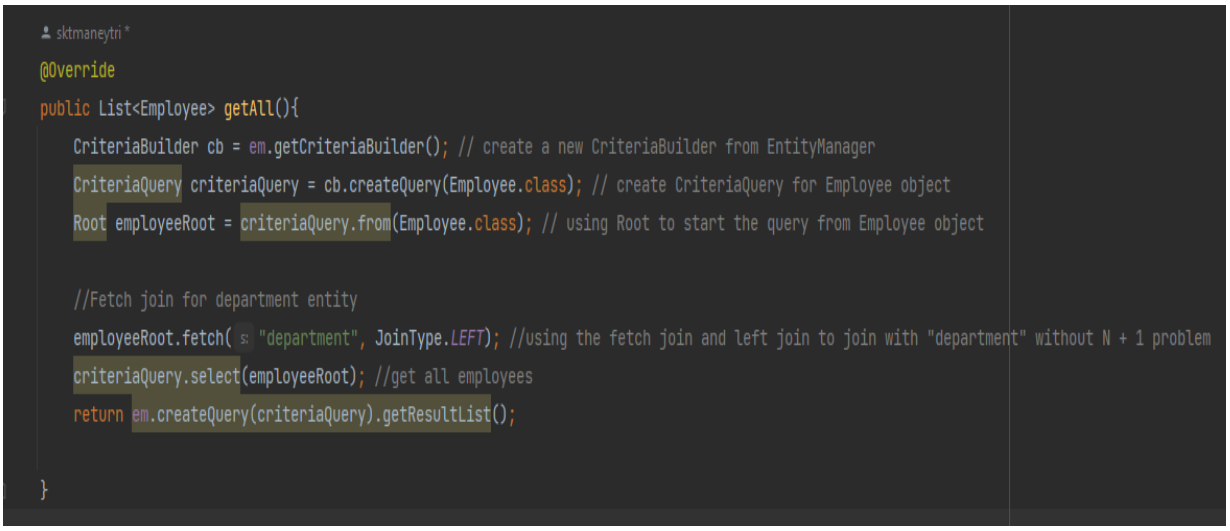
So now, I have 3 main ways that can help u for this problem:

4.1. Fetch Join in Criteria API

The Criteria API is part of the Java Persistence API (JPA) in Java, which provides a type-safe and robust way to query the database without writing SQL queries directly. You can read more in there.

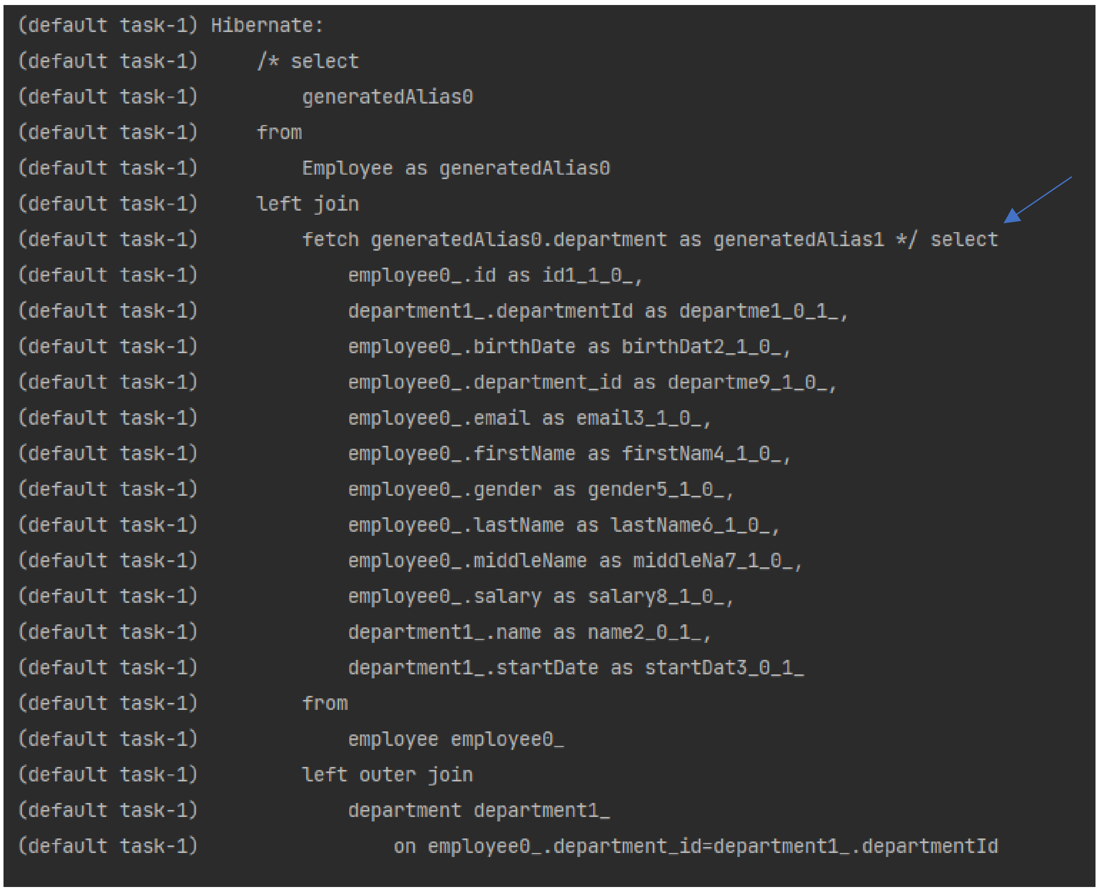
A FETCH JOIN is a way to load one or more related relationships in the same query when executing a query into the database. It helps to solve the "N + 1 problem," where many separate queries are made to load related relationship information when using lazy loading.

- Step 1: Rewrite the getAll function in EmployeeDAOImpl:



```
sktmaneytri *
@Override
public List<Employee> getAll(){
    CriteriaBuilder cb = em.getCriteriaBuilder(); // create a new CriteriaBuilder from EntityManager
    CriteriaQuery criteriaQuery = cb.createQuery(Employee.class); // create CriteriaQuery for Employee object
    Root employeeRoot = criteriaQuery.from(Employee.class); // using Root to start the query from Employee object

    //Fetch join for department entity
    employeeRoot.fetch( s: "department", JoinType.LEFT); //using the fetch join and left join to join with "department" without N + 1 problem
    criteriaQuery.select(employeeRoot); //get all employees
    return em.createQuery(criteriaQuery).getResultList();

}
```

- Step 2: Rebuild the project and send request:



```
(default task-1) Hibernate:
(default task-1)     /* select
(default task-1)         generatedAlias0
(default task-1)     from
(default task-1)         Employee as generatedAlias0
(default task-1)     left join
(default task-1)         fetch generatedAlias0.department as generatedAlias1 */ select
(default task-1)             employee0_.id as id1_1_0_,
(default task-1)             department1_.departmentId as departme1_0_1_,
(default task-1)             employee0_.birthDate as birthDat2_1_0_,
(default task-1)             employee0_.department_id as departme9_1_0_,
(default task-1)             employee0_.email as email3_1_0_,
(default task-1)             employee0_.firstName as firstNam4_1_0_,
(default task-1)             employee0_.gender as gender5_1_0_,
(default task-1)             employee0_.lastName as lastName6_1_0_,
(default task-1)             employee0_.middleName as middleNa7_1_0_,
(default task-1)             employee0_.salary as salary8_1_0_,
(default task-1)             department1_.name as name2_0_1_,
(default task-1)             department1_.startDate as startDat3_0_1_
(default task-1)         from
(default task-1)             employee employee0_
(default task-1)         left outer join
(default task-1)             department department1_
(default task-1)                 on employee0_.department_id=department1_.departmentId
```

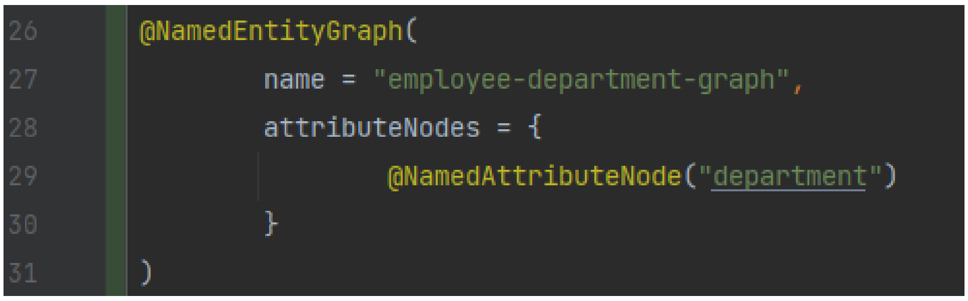
So I got 1 query just like this.

Notice the first select query in / */ just a note when u use Criteria API, it is not the output of the SQL query, it just displays the syntax of the query to clearly indicate to you that a Fetch Join was used.

4.2. Using "Named Entity Graph"

Named entity graph is a feature in JPA that help u to pre-define the relationships that u want to load (fetch) from the database in a specific way. This optimizes performance and reduces unnessary queries, especially in cases where there are complex relationships between entities. Read more at here.

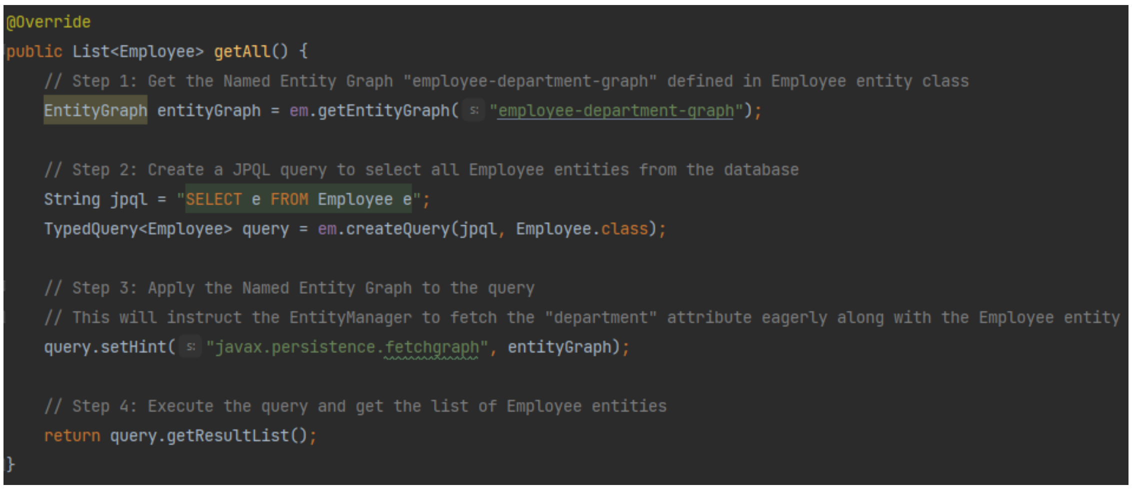
- Step 1: Define and configure like this



```
@NamedEntityGraph(
        name = "employee-department-graph",
        attributeNodes = {
                @NamedAttributeNode("department")
        }
)
```

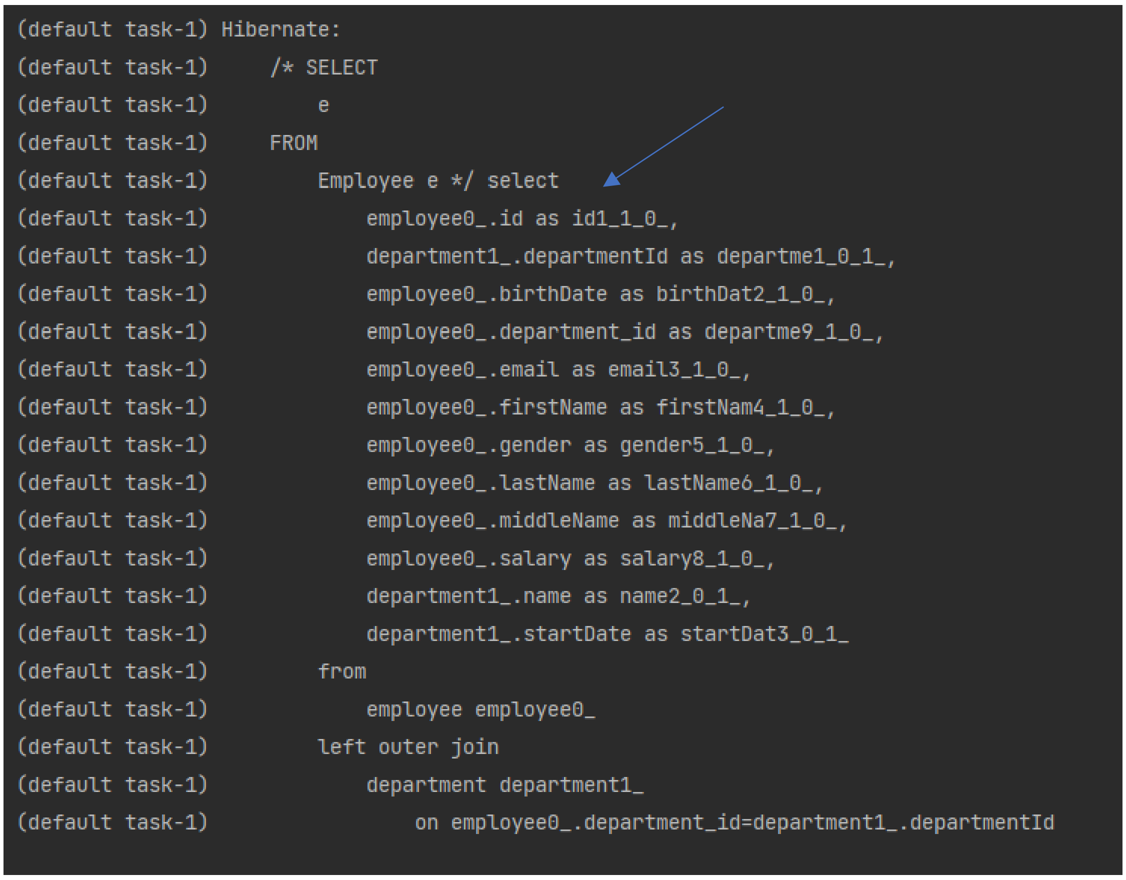
In the above example, I have defined a Named Entity Graph named "employee-department-graph" for the Employee entity, and we want to load the information of the department relationship.

Rewrite the function getAll() like this.



```java
@Override
public List<Employee> getAll() {
    // Step 1: Get the Named Entity Graph "employee-department-graph" defined in Employee entity class
    EntityGraph entityGraph = em.getEntityGraph( s: "employee-department-graph");

    // Step 2: Create a JPQL query to select all Employee entities from the database
    String jpql = "SELECT e FROM Employee e";
    TypedQuery<Employee> query = em.createQuery(jpql, Employee.class);

    // Step 3: Apply the Named Entity Graph to the query
    // This will instruct the EntityManager to fetch the "department" attribute eagerly along with the Employee entity
    query.setHint( s: "javax.persistence.fetchgraph", entityGraph);

    // Step 4: Execute the query and get the list of Employee entities
    return query.getResultList();
}
```

- Step 2: Rebuild the project and send request



```
(default task-1) Hibernate:
(default task-1)     /* SELECT
(default task-1)         e
(default task-1)     FROM
(default task-1)         Employee e */ select
(default task-1)             employee0_.id as id1_1_0_,
(default task-1)             department1_.departmentId as departme1_0_1_,
(default task-1)             employee0_.birthDate as birthDat2_1_0_,
(default task-1)             employee0_.department_id as departme9_1_0_,
(default task-1)             employee0_.email as email3_1_0_,
(default task-1)             employee0_.firstName as firstNam4_1_0_,
(default task-1)             employee0_.gender as gender5_1_0_,
(default task-1)             employee0_.lastName as lastName6_1_0_,
(default task-1)             employee0_.middleName as middleNa7_1_0_,
(default task-1)             employee0_.salary as salary8_1_0_,
(default task-1)             department1_.name as name2_0_1_,
(default task-1)             department1_.startDate as startDat3_0_1_
(default task-1)         from
(default task-1)             employee employee0_
(default task-1)         left outer join
(default task-1)             department department1_
(default task-1)                 on employee0_.department_id=department1_.departmentId
```

I got only one query select.

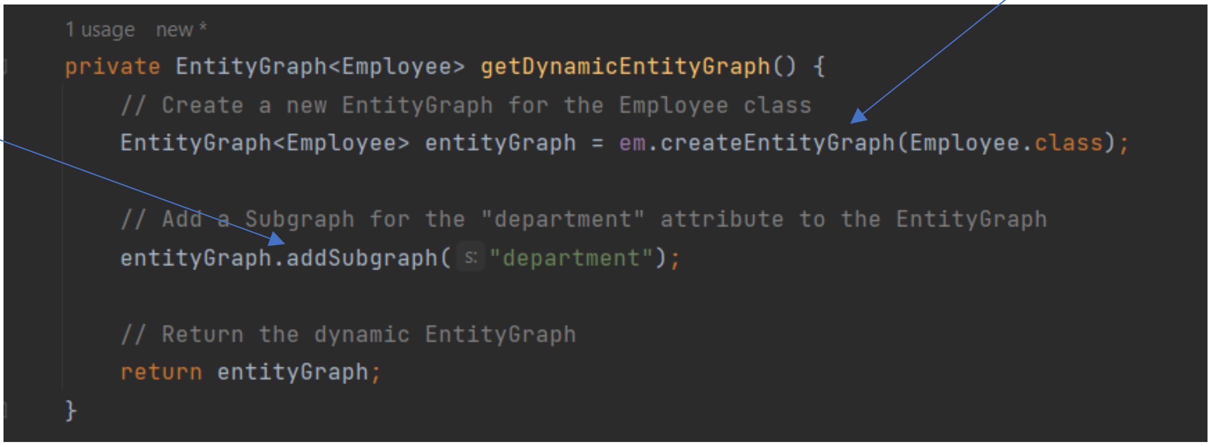
Notice the first select query in / */ just a note

4.2. Using “Dynamic Entity Graph”

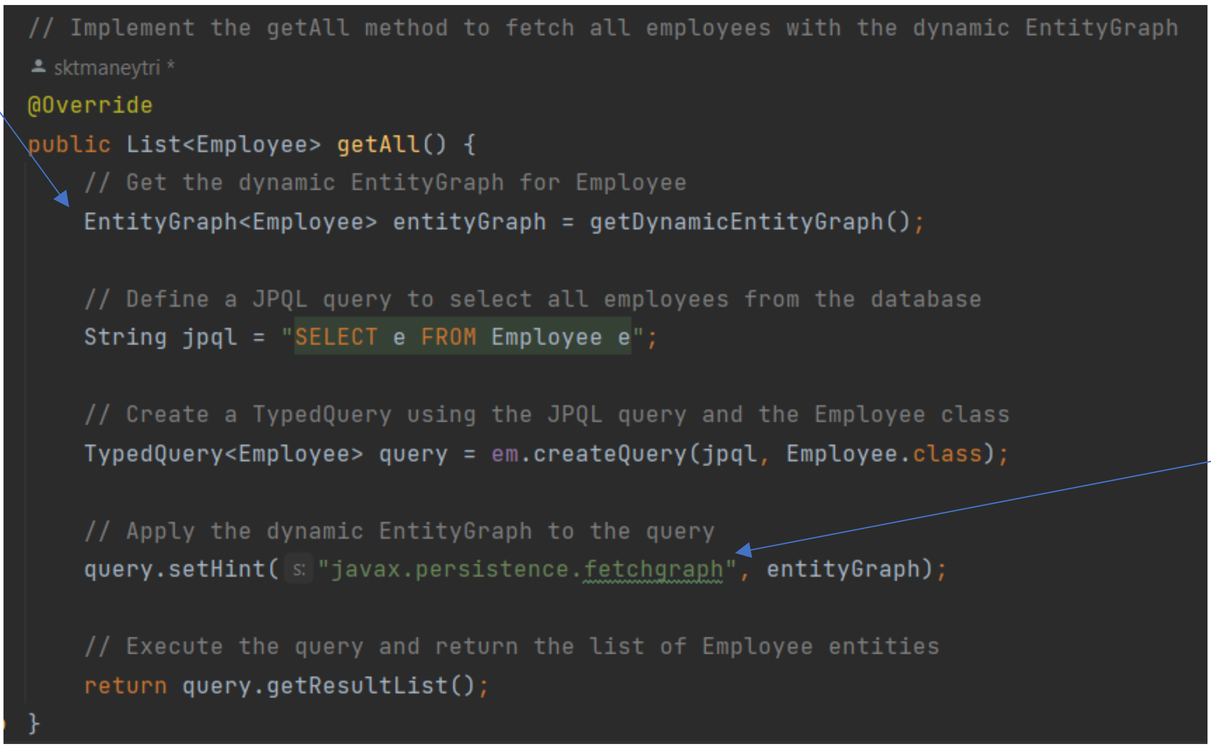
The dynamic entity graph is similar to the named entity graph the only difference is, that the entity graph is defined via a Java API.

The more convenient thing when you use dynamic entity graph is that you will not need to pre-define the annotations on the entity class side, which makes the code in your entity class more concise and clear.

- Step 1: Define and configure like this

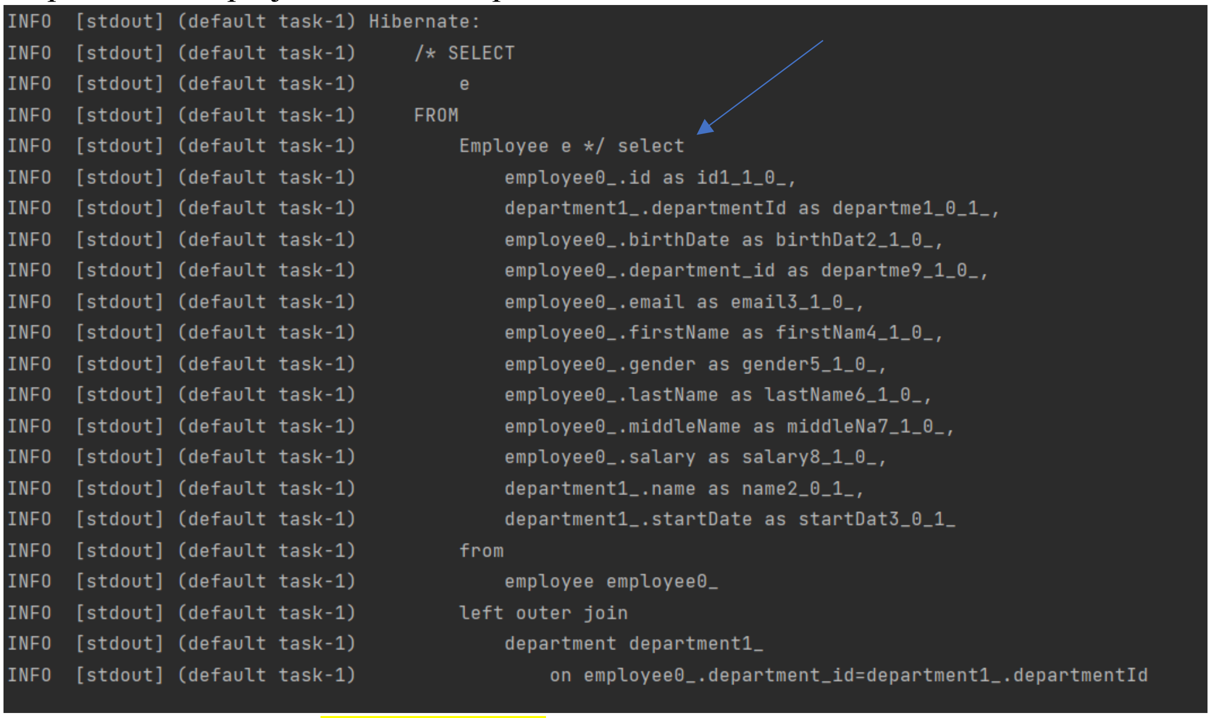


```
1 usage   new *
private EntityGraph<Employee> getDynamicEntityGraph() {
    // Create a new EntityGraph for the Employee class
    EntityGraph<Employee> entityGraph = em.createEntityGraph(Employee.class);

    // Add a Subgraph for the "department" attribute to the EntityGraph
    entityGraph.addSubgraph( s: "department");

    // Return the dynamic EntityGraph
    return entityGraph;
}
```



```
// Implement the getAll method to fetch all employees with the dynamic EntityGraph
sktmaneytri *
@Override
public List<Employee> getAll() {
    // Get the dynamic EntityGraph for Employee
    EntityGraph<Employee> entityGraph = getDynamicEntityGraph();

    // Define a JPQL query to select all employees from the database
    String jpql = "SELECT e FROM Employee e";

    // Create a TypedQuery using the JPQL query and the Employee class
    TypedQuery<Employee> query = em.createQuery(jpql, Employee.class);

    // Apply the dynamic EntityGraph to the query
    query.setHint( s: "javax.persistence.fetchgraph", entityGraph);

    // Execute the query and return the list of Employee entities
    return query.getResultList();
}
```

- Step 2: Rebuild project and sent request:



```
INFO  [stdout] (default task-1) Hibernate:
INFO  [stdout] (default task-1)     /* SELECT
INFO  [stdout] (default task-1)         e
INFO  [stdout] (default task-1)     FROM
INFO  [stdout] (default task-1)         Employee e */ select
INFO  [stdout] (default task-1)             employee0_.id as id1_1_0_,
INFO  [stdout] (default task-1)             department1_.departmentId as departme1_0_1_,
INFO  [stdout] (default task-1)             employee0_.birthDate as birthDat2_1_0_,
INFO  [stdout] (default task-1)             employee0_.department_id as departme9_1_0_,
INFO  [stdout] (default task-1)             employee0_.email as email3_1_0_,
INFO  [stdout] (default task-1)             employee0_.firstName as firstNam4_1_0_,
INFO  [stdout] (default task-1)             employee0_.gender as gender5_1_0_,
INFO  [stdout] (default task-1)             employee0_.lastName as lastName6_1_0_,
INFO  [stdout] (default task-1)             employee0_.middleName as middleNa7_1_0_,
INFO  [stdout] (default task-1)             employee0_.salary as salary8_1_0_,
INFO  [stdout] (default task-1)             department1_.name as name2_0_1_,
INFO  [stdout] (default task-1)             department1_.startDate as startDat3_0_1_
INFO  [stdout] (default task-1)         from
INFO  [stdout] (default task-1)             employee employee0_
INFO  [stdout] (default task-1)         left outer join
INFO  [stdout] (default task-1)             department department1_
INFO  [stdout] (default task-1)                 on employee0_.department_id=department1_.departmentId
```

⇨ You can see that, I got one select query to get all employees and their departments.

Therefore, if you need to design a use case-specific graph that you won't reuse, I advise using dynamic entity graphs.

It is simpler to annotate a named entity graph if you want to reuse the entity graph.

CONCLUSION

Method	**In case of used**
Criteria API	The Criteria API also supports fetch joins and you need specific code for each combination of associations that shall be initialized.
Named entity graphs	Named entity graphs are a good solution if you will reuse the defined graph in our code.
Dynamic entity graphs	Dynamic entity graphs can be the better solution if you need to define a use case specific graph.

This lab is presented by Minh Tri Nguyen and is referenced through the website: https://thorben-janssen.com/5-ways-to-initialize-lazy-relations-and-when-to-use-them/

THANKS FOR READING!