**@EntityGraph and @NamedEntityGraph in Spring JPA - 2024**

The Entity Graph specification was introduced in JPA 2.1, offering developers fine-grained control over how entities and their related data are fetched from the database. With entity graphs, developers can define explicit fetch plans, reducing the N+1 query problem and eliminating the need for manual JOIN clauses. This feature empowers developers to optimize data retrieval efficiently.

**What the Entity Graph Tries to Resolve**

Until JPA 2.0, to load an entity association, we usually used *FetchType.LAZY* and *FetchType.EAGER*as fetching strategies*.*This instructs the JPA provider to additionally fetch the related association or not. **Unfortunately, this meta configuration is static** and doesn’t allow switching between these two strategies at runtime.

**The main goal of the JPA Entity Graph is then to improve the runtime performance when loading the entity’s related associations and basic fields.**

Briefly put, the JPA provider loads all the graphs in one select query and then avoids fetching associations with more SELECT queries.

An example is given below.

We have Organisation, organization has branches, each branch has few employees, organization has also project and department.

@Entity(name="Organization") @Table(name = "org")

@NoArgsConstructor @Setter @Getter

@ToString(exclude = {"branchList", "departList", "projectList"})

**public** **class** Organization {

@Id @GeneratedValue

**private** Long id;

**private** String name;

**public** Organization(String name) {

**this**.name = name;

}

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** List<Branch> branchList = **new** ArrayList<>();

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** List<Department> departList = **new** ArrayList<>();

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** List<Project> projectList = **new** ArrayList<>();

}

@Entity(name="Branch") @Table(name = "branch")

@Getter @Setter @NoArgsConstructor

@ToString(exclude = "empList")

**public** **class** Branch {

@Id @GeneratedValue

**private** Long id;

**private** String name;

**public** Branch(String name) {

**this**.name = name;

}

@OneToMany(cascade=CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** List<Employee> empList = **new** ArrayList<>();

}

@Entity(name="Employee") @Table(name = "emp") @Getter @Setter @ToString @NoArgsConstructor

**public** **class** Employee {

@Id @GeneratedValue

**private** Long id;

**private** String name;

**public** Employee(String name) {

**this**.name = name;

}

}

@Entity(name="Department") @Table(name = "dept") @Getter @Setter @ToString @NoArgsConstructor

**public** **class** Department {

@Id @GeneratedValue

**private** Long id;

**private** String name;

**public** Department(String name) {

**this**.name = name;

}

}

@Entity(name="Project") @Table(name = "project") @Getter @Setter @ToString @NoArgsConstructor

**public** **class** Project {

@Id @GeneratedValue

**private** Long id;

**private** String name;

**public** Project(String name) {

**this**.name = name;

}

}

**Code to save the data**

**private** **void** saveOrg() {

Organization org = **new** Organization("Microsoft");

List<Employee> empList1 = List.*of*( **new** Employee("John"), **new** Employee("Vidya"), **new** Employee("Rambha"));

Branch branch1 = **new** Branch("Delhi");

branch1.setEmpList(empList1);

List<Employee> empList2 = List.*of*( **new** Employee("Urbasi"), **new** Employee("Meneka"), **new** Employee("Ramesh"));

Branch branch2 = **new** Branch("Bangalore");

branch2.setEmpList(empList2);

List<Branch> branchList = List.*of*(branch1, branch2);

List<Department> deptList = List.*of*(**new** Department("Finance"), **new** Department("IT"));

List<Project> projectList = List.*of*(**new** Project("Tombola"), **new** Project("DelTobosso"));

org.setBranchList(branchList);

org.setDepartList(deptList);

org.setProjectList(projectList);

orgRepo.save(org);

}

**How to use Adhoc @EntityGraph without using @NamedEntityGraph**

To get the Organisation details by Id, we have the following Repository class.

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

}

We have the following code to get the organization details.

**public** **void** showOrgDetails() {

Organization org = orgRepo.findById(1L).get();

System.***out***.println("Org: "+org);

**try** {

TimeUnit.***SECONDS***.sleep(5);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

org.getBranchList().forEach( branch -> System.***out***.println("Brnach: "+branch));

}

**OUTPUT**

**Initial generated query will be like this**

Hibernate: **select o1\_0.id, o1\_0.name from org o1\_0 where o1\_0.id=?**

Org: Organization(id=1, name=Microsoft)

After 5 seconds, the following query will be fired to get the branch details.

Hibernate: **select bl1\_0.organization\_id, bl1\_1.id, bl1\_1.name from org\_branch\_list bl1\_0**

**Join branch bl1\_1 on bl1\_1.id=bl1\_0.branch\_list\_id**

**Where bl1\_0.organization\_id=?**

Branch: Branch(id=1, name=Delhi)

Branch: Branch(id=2, name=Bangalore)

It happens because, we have declared branch as, fetch is lazy here.

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** List<Branch> branchList = **new** ArrayList<>();

Now, we want to achieve the result with dynamic eager type using adhoc @EntityGraph

Now, the repository layer is modified as

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

**@EntityGraph(attributePaths = {"branchList"})**

Optional<Organization> findById(Long id);

}

If we re-execute the above code showOrgDetails(), the following output will be obtained.

Hibernate: **select o1\_0.id, o1\_0.name, bl1\_0.organization\_id, bl1\_1.id, bl1\_1.name**

**From org o1\_0**

**left join org\_branch\_list bl1\_0 on o1\_0.id=bl1\_0.organization\_id**

**left join branch bl1\_1 on bl1\_1.id=bl1\_0.branch\_list\_id**

**where o1\_0.id=?**

Org: Organization(id=1, name=Microsoft)

Branch: Branch(id=1, name=Delhi)

Branch: Branch(id=2, name=Bangalore)

Now we get the single query to get all the details.

If we execute the following statement,

org.getProjectList().forEach( project -> System.***out***.println("Project: "+project));

Since projectList is of type Lazy. Hibernate will again fetch lazily.

The output is given below.

Hibernate:

**Select o1\_0.id, o1\_0.name, bl1\_0.organization\_id, bl1\_1.id,**

**bl1\_1.name**

**from org o1\_0**

**left join org\_branch\_list bl1\_0 on o1\_0.id=bl1\_0.organization\_id**

**left join branch bl1\_1 on bl1\_1.id=bl1\_0.branch\_list\_id**

**where o1\_0.id=?**

Org: Organization(id=1, name=Microsoft)

Branch: Branch(id=1, name=Delhi)

Branch: Branch(id=2, name=Bangalore)

Hibernate:

**Select pl1\_0.organization\_id, pl1\_1.id, pl1\_1.name**

**From org\_project\_list pl1\_0**

**Join project pl1\_1 on pl1\_1.id=pl1\_0.project\_list\_id**

**Where pl1\_0.organization\_id=?**

Project: Project(id=1, name=Tombola)

Project: Project(id=2, name=DelTobosso)

To solve the above problem, again modify the repository query.

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

**@EntityGraph(attributePaths = {"branchList", "projectList"})**

**Optional<Organization> findById(Long id);**

}

**OUTPUT**

Hibernate:

Select o1\_0.id, o1\_0.name, bl1\_0.organization\_id, bl1\_1.id, bl1\_1.name

From org o1\_0

left join org\_branch\_list bl1\_0 on o1\_0.id=bl1\_0.organization\_id

left join branch bl1\_1 on bl1\_1.id=bl1\_0.branch\_list\_id

where o1\_0.id=?

Hibernate:

Select pl1\_0.organization\_id, pl1\_1.id, pl1\_1.name

From org\_project\_list pl1\_0

Join project pl1\_1 on pl1\_1.id=pl1\_0.project\_list\_id

Where pl1\_0.organization\_id=?

Org: Organization(id=1, name=Microsoft)

Branch: Branch(id=1, name=Delhi)

Branch: Branch(id=2, name=Bangalore)

Project: Project(id=1, name=Tombola)

Project: Project(id=2, name=DelTobosso)

There is another requirement to get the Organisation details along with branch and employees of each branch.

If the repository class is given like this.

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

**@EntityGraph(attributePaths = {"branchList"})**

**Optional<Organization> findById(Long id);**

}

We have the following code.

**public** **void** sleep(**long** time) {

**try** {

TimeUnit.***SECONDS***.sleep(time);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**public** **void** showOrgDetails() {

Organization org = orgRepo.findById(1L).get();

System.***out***.println("Org: " + org);

System.***out***.println("------------ GET Branch Details --------------");

sleep(5);

org.getBranchList().forEach(branch -> {

System.***out***.println("Branch: " + branch);

System.***out***.println("------------ GET Employee Details --------------");

sleep(5);

branch.getEmpList().forEach(emp -> System.***out***.println("Emp: " + emp));

});

}

The initial output will be like this.

Hibernate:

**Select o1\_0.id, o1\_0.name, bl1\_0.organization\_id, bl1\_1.id, bl1\_1.name From org o1\_0**

**left join org\_branch\_list bl1\_0 on o1\_0.id=bl1\_0.organization\_id**

**left join branch bl1\_1 on bl1\_1.id=bl1\_0.branch\_list\_id**

**where o1\_0.id=?**

Org: Organization(id=1, name=Microsoft)

------------ GET Branch Details --------------

Branch: Branch(id=1, name=Delhi)

------------ GET Employee Details --------------

Hibernate:

**Select el1\_0.branch\_id, el1\_1.id, el1\_1.name from branch\_emp\_list el1\_0**

**Join emp el1\_1 on el1\_1.id=el1\_0.emp\_list\_id**

**Where el1\_0.branch\_id=?**

Emp: Employee(id=1, name=John)

Emp: Employee(id=2, name=Vidya)

Emp: Employee(id=3, name=Rambha)

Branch: Branch(id=2, name=Bangalore)

------------ GET Employee Details --------------

Hibernate:

Select el1\_0.branch\_id, el1\_1.id, el1\_1.name from branch\_emp\_list el1\_0

Join emp el1\_1 on el1\_1.id=el1\_0.emp\_list\_id

Where el1\_0.branch\_id=?

Emp: Employee(id=4, name=Urbasi)

Emp: Employee(id=5, name=Meneka)

Emp: Employee(id=6, name=Ramesh)

Now let us modify the Repository class like this.

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

@EntityGraph(attributePaths = {"branchList", "branchList.empList"})

Optional<Organization> findById(Long id);

}

Output will be like this.

Hibernate:

Select o1\_0.id, o1\_0.name, bl1\_0.organization\_id, bl1\_1.id, bl1\_1.name from org o1\_0

left join org\_branch\_list bl1\_0 on o1\_0.id=bl1\_0.organization\_id

left join branch bl1\_1 on bl1\_1.id=bl1\_0.branch\_list\_id

where o1\_0.id=?

Hibernate:

select el1\_0.branch\_id, el1\_1.id, el1\_1.name from branch\_emp\_list el1\_0

join emp el1\_1 on el1\_1.id=el1\_0.emp\_list\_id

where el1\_0.branch\_id=?

Hibernate:

Select el1\_0.branch\_id, el1\_1.id, el1\_1.name from branch\_emp\_list el1\_0

Join emp el1\_1 on el1\_1.id=el1\_0.emp\_list\_id

Where el1\_0.branch\_id=?

Org: Organization(id=1, name=Microsoft)

------------ GET Branch Details --------------

Branch: Branch(id=1, name=Delhi)

------------ GET Employee Details --------------

Emp: Employee(id=1, name=John)

Emp: Employee(id=2, name=Vidya)

Emp: Employee(id=3, name=Rambha)

Branch: Branch(id=2, name=Bangalore)

------------ GET Employee Details --------------

Emp: Employee(id=4, name=Urbasi)

Emp: Employee(id=5, name=Meneka)

Emp: Employee(id=6, name=Ramesh)

You can also used **@NamedEntityGraph in class level** to make the eager fetching.

Example is given below.

@Entity(name="Organization") @Table(name = "org")

@NoArgsConstructor @Setter @Getter @ToString(exclude = {"branchList", "departList", "projectList"})

**@NamedEntityGraph(**

**name = "Org.branch-with-emp-and-project-eager-type",**

**attributeNodes =**

**{**

**@NamedAttributeNode(value = "branchList", subgraph = "empList" /\* For inner object \*/),**

**@NamedAttributeNode("projectList")**

**})**

**public** **class** Organization {

@Id @GeneratedValue

**private** Long id;

**private** String name;

**public** Organization(String name) {

**this**.name = name;

}

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** Set<Branch> branchList = **new** HashSet<>();

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** Set<Department> departList = **new** HashSet<>();

@OneToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***LAZY***)

**private** Set<Project> projectList = **new** HashSet<>();

}

Define the name in repository class like this.

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

@EntityGraph(value = "Org.branch-with-emp-and-project-eager-type")

Organization getOrganizationByName(String name);

}

You can also use @EntityGraph along with @Query

@Repository

**public** **interface** OrgRepository **extends** CrudRepository<Organization, Long> {

@EntityGraph(attributePaths = {"branches", "branches.employees"})

Optional<Organization> findById(Long id);

@EntityGraph(value = "Org.branch-with-emp-and-project-eager-type")

Organization getOrganizationByName(String name);

// EntityGraph with @Query

@EntityGraph(attributePaths = {"projects"}) **🡸 @EntityGraph + @Query**

@Query("select org from Organization org where org.name=:name")

Organization getOrgAlongWithProject(@Param("name") String name);

}

**As per the above example, the following database objects were created.**

|  |  |
| --- | --- |
| **Sequence Name** | **Table Name** |
| branch\_seq | branch |
| dept\_seq | branch\_emp\_list |
| emp\_seq | dept |
| org\_seq | emp |
| project\_seq | org |
|  | org\_branch\_list |
|  | org\_depart\_list |
|  | org\_project\_list |
|  | project |

References

https://jeddict.github.io/page.html?l=tutorial/NamedEntityGraph

https://stackoverflow.com/questions/31943989/spring-data-jpa-and-namedentitygraphs

https://blog.ippon.tech/boost-the-performance-of-your-spring-data-jpa-application

https://turkogluc.com/understanding-the-effective-data-fetching-with-jpa-entity-graphs-part-2/

https://refactorizando.com/en/entity-graph-with-jpa-in-spring-boot/

https://blog.adamgamboa.dev/effective-data-load-in-jpa-with-entity-graph/

https://foojay.io/today/a-walk-to-lazy-fetching-with-hibernate-and-spring-data-jpa/