5 Ways To Initialize Lazy Relationships

1. Call a method on the mapped relation

Hibernate performs an additional query to initialize the relationship.

```
Order order = this.em.find(Order.class, orderId);
order.getItems().size();
```

2. Fetch Join in JPQL

Hibernate fetches the selected entity and the relationship within the same query.

```
Query q = this.em.createQuery(
      "SELECT o FROM Order o JOIN FETCH o.items i “
      + “WHERE o.id = :id");
q.setParameter("id", orderId);
Order order = (Order) q.getSingleResult();
```

3. Fetch Join in Criteria API

Hibernate fetches the selected entity and the relationship within the same query.

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery q = cb.createQuery(Order.class);
Root o = q.from(Order.class);
o.fetch("items", JoinType.INNER);
q.select(o);
q.where(cb.equal(o.get("id"), orderId));
Order o = (Order) this.em.createQuery(q).getSingleResult();
```

4. Named Entity Graph

The named entity graph is a new feature of JPA 2.1. It can be used to annotate a graph of entities which shall be fetched from the database.

```
@Entity
@NamedEntityGraph(name = "graph.Order.items",
    attributeNodes = @NamedAttributeNode("items"))
public class Order implements Serializable {
....
}
```

You can then use the named entity graph with the find method of the EntityManager.

```
EntityGraph graph =
this.em.getEntityGraph("graph.Order.items");

Map hints = new HashMap();
hints.put("javax.persistence.fetchgraph", graph);

Order order = this.em.find(Order.class, orderId, hints);
```

5. Dynamic Entity Graph

The dynamic entity graph is a new feature of JPA 2.1 and very similar to the named entity graph. It uses a Java API to define a graph of entities which shall be fetched from the database.

```
EntityGraph graph = this.em.createEntityGraph(Order.class);
Subgraph itemGraph = graph.addSubgraph("items");

Map hints = new HashMap();
hints.put("javax.persistence.loadgraph", graph);

Order order = this.em.find(Order.class, orderId, hints);
```