5 tips to write efficient queries with JPA and Hibernate

1. Choose a projection that fits your use case

You should decide for each use case, which information it needs and which operations it has to perform and choose accordingly.

Entities are a good fit if you have to update or remove a record. They might also be ok for use cases which need to read (almost) all entity attributes. But keep in mind that the persistence context has to manage the entities which creates an overhead compared to a DTO projection.

DTO's are a good fit for use cases that only need to read a record if they provide all required and no additional properties. That often requires you to create a new DTO when you implement a new use case. That is where most discussions start. You can't reuse the same DTO and data access services for all use cases if you want to optimize for efficiency.

But don't worry, this doesn't have to be a black and white decision. Most development teams decide to do a little bit of both. They accept minor inefficiencies in their database access and create DTOs that are quite a good but not an optimal fit for multiple use cases to improve reusability. That's totally fine. You just have to be aware of it so that you can change it if you into performance issues.

2. Avoid eager fetching in your mapping definition

From a performance point of view, choosing the right *FetchTypes* for your entity associations is one of the most important steps. The *FetchType* defines when Hibernate performs additional queries to initialize an association. It can either do that when it loads the entity (*FetchType.EAGER*) or when you use the association (*FetchType.LAZY*).

It doesn't make any sense to perform additional queries to load data before you know that you need it. You should use *FetchType.LAZY* by default and apply the next tip if a use case uses an entity association.

3. Initialize all required associations in your query

Hibernate has to perform additional queries to load uninitialized, lazy associations. When you do that for multiple entities, you create an n+1 select issue.

You can easily avoid that by initializing all required associations within the query that loads your entities. You can either do that with a query-independent *EntityGraph* or with a simple *JOIN FETCH* clause in your JPQL or Criteria Query.

4. Use pagination when you select a list of entities

Humans can't handle lists with hundreds of elements. Most UIs, therefore, split them into multiple chunks and present each of them on a separate page.

In these cases, it doesn't make any sense to fetch all entities or DTOs in one query. The UI doesn't need them, and it just slows down your application. It's much better to use the same pagination approach in your query and fetch only the records that are shown in the UI. You can do that by setting appropriate values for firstResult and maxResult on the Query interface.

5. Log SQL statements

You should always check the executed SQL statements when you apply any changes to your code. The easiest way to do that is to activate the logging of SQL statements in your development configuration. You can do that by setting the log level of *org.hibernate.SQL* to *DEBUG*.