

In Hibernate and JPA, there are three primary ways to query the database:

1. **HQL (Hibernate Query Language)**
2. **Named Queries**
3. **Native SQL Queries**

Let's explore each of them with examples.

### 1. HQL (Hibernate Query Language):

- HQL is an **object-oriented query language** similar to SQL but operates on **entity objects** rather than directly **on database tables**.
- It uses **entity names and fields instead of table names and columns**.

#### Example: Fetching Users with HQL

```
String hql = "FROM User u WHERE u.firstName = :firstName";
```

```
Query<User> query = session.createQuery(hql, User.class);  
query.setParameter("firstName", "Gopal");
```

```
List<User> users = query.getResultList();
```

```
for (User user : users) {  
    System.out.println(user.getFullName());  
}
```

#### Explanation:

- "FROM User u" refers to the User entity, not the database table.
- :firstName is a named parameter used for setting query values.

#### Advantages of HQL:

- **Object-Oriented:** Works with entity objects, leveraging Hibernate's mapping capabilities.
- **Portable:** Independent of the underlying SQL dialect, making it easier to switch databases.

## 2. Named Query:

- A **named query** is a predefined HQL or JPQL (Java Persistence Query Language) query that is defined using the **@NamedQuery** annotation or XML configuration.
- It allows you to define queries once and use them multiple times.

### Example Using @NamedQuery Annotation:

#### Entity Definition:

##### @Entity

##### @NamedQuery(

**name = "User.findByLastName",**

**query = "FROM User u WHERE u.lastName = :lastName"**

)

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String fullName;

// Getters and Setters

}

#### Using the Named Query:

**Query<User> query =**

**session.createNamedQuery("User.findByLastName", User.class);**

**query.setParameter("lastName", "Kumar");**

**List<User> users = query.getResultList();**

```
for (User user : users) {  
    System.out.println(user.getFullName());  
}
```

#### Explanation:

- @NamedQuery defines a query named "User.findByLastName".
- The query is fetched using session.createNamedQuery().

#### Advantages of Named Queries:

- **Centralized Definition:** Queries are defined in one place (either annotations or XML), making them easier to manage.
- **Reusability:** Can be reused across the application without rewriting the query.

### 3. Native SQL Query:

- Native queries are **plain SQL queries** executed directly against the database.
- These queries **do not** involve the **HQL or JPQL** syntax and are specific to the database dialect.

#### Example of a Native SQL Query:

```
String sql = "SELECT * FROM User WHERE first_name = :firstName";  
Query<User> query = session.createNativeQuery(sql, User.class);  
query.setParameter("firstName", "Gopal");  
List<User> users = query.getResultList();
```

```
for (User user : users) {  
    System.out.println(user.getFullName());  
}
```

#### Explanation:

- "SELECT \* FROM User" is standard SQL and operates directly on the database table.
- createNativeQuery() is used to execute this SQL query.

## Advantages of Native Queries:

- **Full SQL Power:** You can use database-specific features like functions, joins, and subqueries.
- **Flexibility:** Ideal for complex queries that cannot be easily expressed in HQL or JPQL.

## Comparison:

Feature	HQL	Named Query	Native Query
<b>Syntax</b>	Object-oriented HQL/JPQL	Object-oriented HQL/JPQL	Plain SQL
<b>Performance</b>	Optimized by Hibernate	Optimized and precompiled	Directly executed by DB
<b>Database Independence</b>	Yes	Yes	No (DB-specific features)
<b>Ease of Use</b>	Easy for entity queries	Reusable, easy maintenance	More control, but complex
<b>Flexibility</b>	Limited to HQL features	Same as HQL	Full SQL capabilities

## Use Cases:

- **HQL:** When working with entity objects and you want to leverage Hibernate's mapping features.
- **Named Query:** When you have frequently used queries that can be predefined for reuse.
- **Native Query:** When you need to use complex SQL features or database-specific optimizations.