**Introduction to HQL and JPQL**

Hibernate Query Language (HQL) and Java Persistence Query Language (JPQL) are object-oriented query languages used for querying data from databases using entity objects rather than database tables.

1. What is HQL?

\* HQL stands for Hibernate Query Language.

\* It is specific to Hibernate ORM.

\* It works with entity objects and properties instead of tables and columns.

\* Syntax is similar to SQL but operates on Java objects.

Example:

Query query = session.createQuery("FROM Employee WHERE salary > 50000");

2. What is JPQL?

\* JPQL stands for Java Persistence Query Language.

\* It is defined as part of the JPA (Java Persistence API) specification.

\* It provides a database-independent way of querying entities managed by JPA.

\* Very similar to HQL but not tied to Hibernate.

Example:

TypedQuery<Employee> query = em.createQuery("SELECT e FROM Employee e WHERE e.salary > :amount", Employee.class);

query.setParameter("amount", 50000);

3. Differences Between HQL and JPQL

| Feature       | HQL                           | JPQL                             |

| -------------------- | ---------------------------- ----------------| ------------------------------- ----------|

| Defined by     | Hibernate                     | JPA                             |

| Usage Scope   | Hibernate only               | Any JPA provider                 |

| Named Queries | Supported                     | Supported                       |

| Native SQL     | Allowed via `createSQLQuery` | Allowed via `createNativeQuery` |

4. DML (Data Manipulation Language) Operations

INSERT (JPQL does not support insert directly):

Use entity manager's `persist()` method instead.

Employee e = new Employee("John", 60000);

em.persist(e);

UPDATE:

Query query = em.createQuery("UPDATE Employee e SET e.salary = :newSalary WHERE e.name = :name");

query.setParameter("newSalary", 70000);

query.setParameter("name", "John");

int rowsUpdated = query.executeUpdate();

DELETE:

Query query = em.createQuery("DELETE FROM Employee e WHERE e.name = :name");

query.setParameter("name", "John");

int rowsDeleted = query.executeUpdate();

5. Named Queries

\* Can be declared in entity classes using `@NamedQuery`.

Example:

@NamedQuery(name = "Employee.findBySalary", query = "SELECT e FROM Employee e WHERE e.salary > :salary")

Used like:

TypedQuery<Employee> query = em.createNamedQuery("Employee.findBySalary", Employee.class);

query.setParameter("salary", 50000);

6. Advantages

\* Database independent

\* Easy to read and write

\* Supports object-oriented features like inheritance, polymorphism

7. Limitations

\* No support for insert DML via queries

\* Slight performance overhead compared to native SQL