Introduction to HQL and JPQL

1. **HQL (Hibernate Query Language)**

**Definition:**  
HQL is an object-oriented query language similar to SQL but specifically designed for Hibernate. It allows developers to query data using **Java class names and properties** rather than database tables and columns. HQL is aware of Hibernate's object-relational mapping (ORM), so it respects relationships, inheritance, and polymorphism defined in the entities.

**Key Features:**

* Works with Java objects and their properties.
* Supports SELECT, UPDATE, DELETE, and uniquely, INSERT.
* Supports joins, aggregates, subqueries, and more.
* Queries are written in terms of entity names.

**HQL SELECT Example:**

String hql = "FROM Employee E WHERE E.salary > 50000";

Query query = session.createQuery(hql);

List<Employee> results = query.list();

* This retrieves all employees whose salary is above 50,000.

**HQL INSERT Example:**

String hql = "UPDATE Employee E SET E.salary = E.salary + 2000 WHERE E.department = ‘IT’”;

int updated = session.createQuery(hql).executeUpdate();

System.out.println("Updated: " + updated);

1. **JPQL (Java Persistence Query Language)**

**Definition:**  
JPQL is the official query language of JPA (Java Persistence API). It is similar to HQL but more **standardized** across all JPA implementations (like Hibernate, EclipseLink, etc.). It operates on **entities and their relationships** rather than directly on database structures.

**Key Features:**

* Platform-independent (works with any JPA provider).
* Only supports SELECT, UPDATE, and DELETE.
* Does **not support INSERT** like HQL.
* Queries entities using class and field names, not tables and columns.

**JPQL SELECT Example:**

String jpql = "SELECT e FROM Employee e WHERE e.department = :dept";

TypedQuery<Employee> query = entityManager.createQuery(jpql, Employee.class);

query.setParameter("dept", "HR");

List<Employee> employees = query.getResultList();

**JPQL UPDATE Example:**

String jpql = "UPDATE Employee e SET e.salary = e.salary + 1500 WHERE e.department = 'Finance'";

int updated = entityManager.createQuery(jpql).executeUpdate();

System.out.println("Employees updated: " + updated);

**JPQL DELETE Example:**

String jpql = "DELETE FROM Employee e WHERE e.name = 'John'";

int deleted = entityManager.createQuery(jpql).executeUpdate();

System.out.println("Employees deleted: " + deleted);

1. Comparison Between HQL and JPQL

**1️. Definition and Origin**

**HQL (Hibernate Query Language)** is a query language developed specifically for Hibernate. It provides a way to query and manipulate data stored in Java objects that are mapped to database tables using Hibernate ORM.

**JPQL (Java Persistence Query Language)** is a part of the Java Persistence API (JPA) specification. It is designed to be a standard query language across different JPA providers, not just Hibernate.

**2. Scope and Compatibility**

HQL is tied to Hibernate and works only with Hibernate-based applications. In contrast, JPQL is **vendor-agnostic** and can be used with any ORM tool that implements the JPA specification, such as EclipseLink, OpenJPA, or Hibernate itself.

**3. uery Syntax and Structure**

Both HQL and JPQL use a SQL-like syntax, but instead of tables and columns, they operate on Java entity classes and their fields. The queries are written using **entity names and field names** defined in the object model, not database schema names.

1. **Supported Operations**

JPQL supports **SELECT**, **UPDATE**, and **DELETE** operations. However, it does **not support INSERT** queries. HQL, on the other hand, supports **all four**: SELECT, UPDATE, DELETE, and also **INSERT**, making it more versatile for data manipulation within Hibernate.

1. **Feature Coverage**

HQL offers more advanced features, such as **polymorphic queries**, **constructor expressions**, and **support for database-specific functions**. JPQL intentionally avoids some of these to maintain portability and simplicity across various JPA providers.

1. **Subset-Superset Relationship**

JPQL is considered a **subset of HQL**. This means that all JPQL queries are valid HQL queries. However, the reverse is **not true** — not all HQL queries are valid JPQL, especially those involving features like INSERT or Hibernate-specific functions.

1. **Use Case Preference**

Use **HQL** when working exclusively with Hibernate and when you need advanced ORM-specific functionality. Use **JPQL** when building applications intended to be portable across multiple JPA providers and you want to follow Java EE standards.