**Advanced Java — Hibernate Query Language (HQL)**

**Hibernate Query Language**, or HQL, is similar to SQL (Structured Query Language) but is designed to work with Hibernate. Unlike SQL, which works directly with database tables, HQL works with Java objects and their properties. This makes HQL independent of the underlying database, so you can switch databases without changing your queries.

**Advantages of HQL**

* It is database-independent because it uses Java class names instead of table names.
* It supports polymorphic queries, which means you can query parent classes and their child classes in one go.
* It is easy for Java developers to learn because it uses object-oriented concepts.

**Query Interface in Hibernate**

In Hibernate, the **Query** interface allows you to create and execute HQL queries. You can get a **Query** object by calling the **createQuery()** method on a Hibernate **Session**.

**Common methods in the Query Interface:**

* **executeUpdate()**: Runs an update or delete query.
* **list()**: Returns the results as a list.
* **setFirstResult(int rowno)**: Sets the starting point of the results (useful for pagination).
* **setMaxResults(int rowno)**: Limits how many records to return.
* **setParameter(int position, Object value)**: Sets a value in the query using a position (like placeholders in SQL).
* **setParameter(String name, Object value)**: Sets a value using a named parameter.

**Simple HQL Examples**

**1. Get All Records**

This example fetches all records from the **Emp** class (which represents a table in the database):

Query query = session.createQuery("from Emp");  
List list = query.list();

**2. Pagination Example**

This retrieves records from the 5th to the 10th row, useful when you want to display data page by page:

Query query = session.createQuery("from Emp");  
query.setFirstResult(5);  
query.setMaxResults(10);  
List list = query.list();

**3. Update Record Example**

This updates the name of a user with a specific ID:

Transaction tx = session.beginTransaction();  
Query q = session.createQuery("update User set name=:n where id=:i");  
q.setParameter("n", "Udit Kumar");  
q.setParameter("i", 111);

int status = q.executeUpdate();  
System.out.println(status);  
tx.commit();

**4. Delete Record Example**

This deletes a record from the **Emp** class where the ID is 100:

Query query = session.createQuery("delete from Emp where id=100");  
query.executeUpdate();

**Using Aggregate Functions in HQL**

You can use common aggregate functions like **sum()**, **avg()**, **min()**, **max()**, and **count()** in HQL.

**Examples:**

* **Total Salary of All Employees:**

Query q = session.createQuery("select sum(salary) from Emp");   
List<Integer> list = q.list(); System.out.println(list.get(0));

* **Maximum Salary:**

Query q = session.createQuery("select max(salary) from Emp");

* **Minimum Salary:**

Query q = session.createQuery("select min(salary) from Emp");

* **Total Number of Employees:**

Query q = session.createQuery("select count(id) from Emp");

* **Average Salary of Employees:**

Query q = session.createQuery("select avg(salary) from Emp");

**Final Thoughts**

HQL makes it easier for Java developers to write database queries because it focuses on Java objects rather than database tables. With features like database independence, easy pagination, and support for aggregate functions, HQL simplifies complex database interactions while keeping your code clean and maintainable.