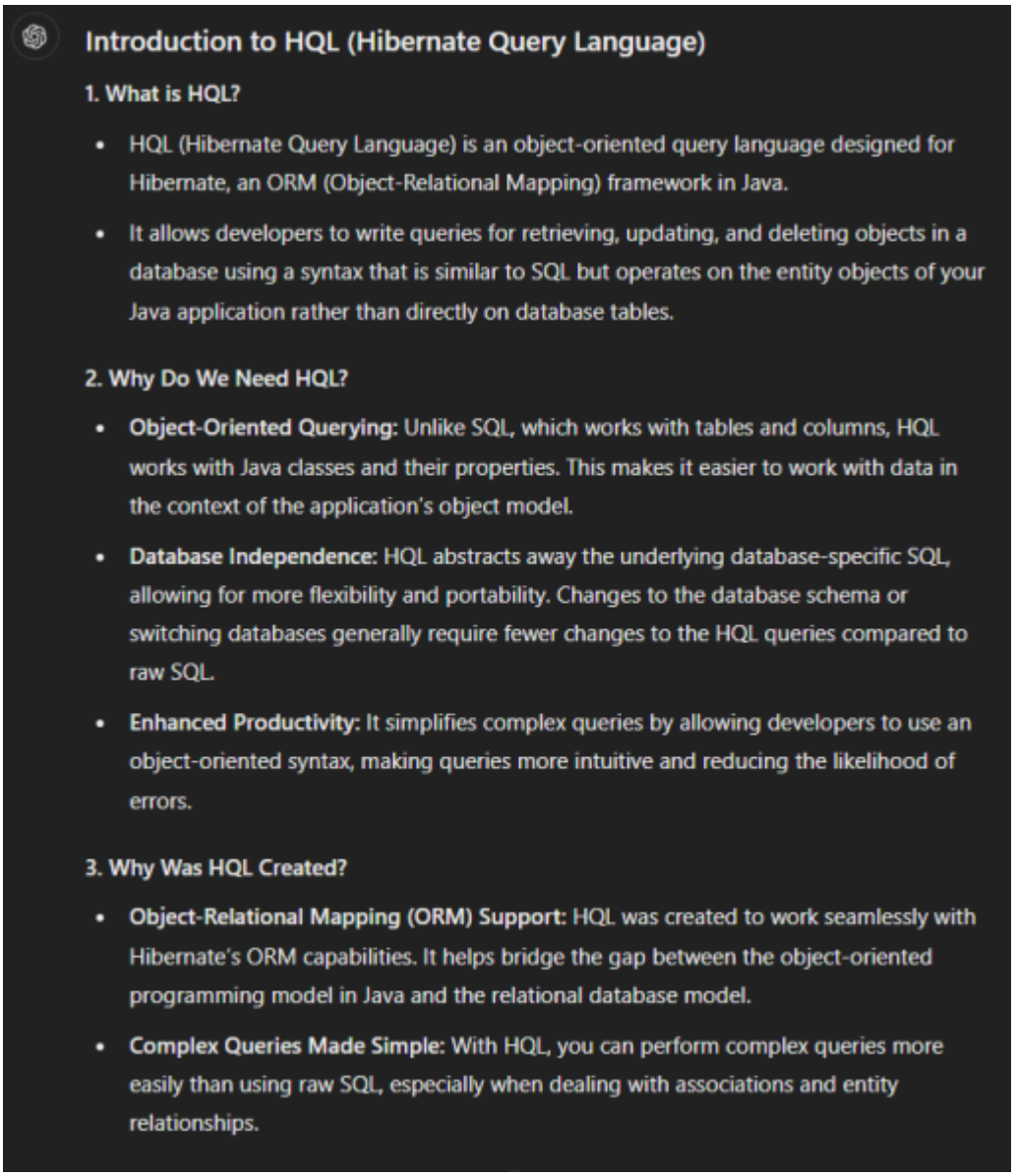
HQL- Hibernate Part 2



Introduction to HQL (Hibernate Query Language)

1. What is HQL?

- HQL (Hibernate Query Language) is an object-oriented query language designed for Hibernate, an ORM (Object-Relational Mapping) framework in Java.
- It allows developers to write queries for retrieving, updating, and deleting objects in a database using a syntax that is similar to SQL but operates on the entity objects of your Java application rather than directly on database tables.

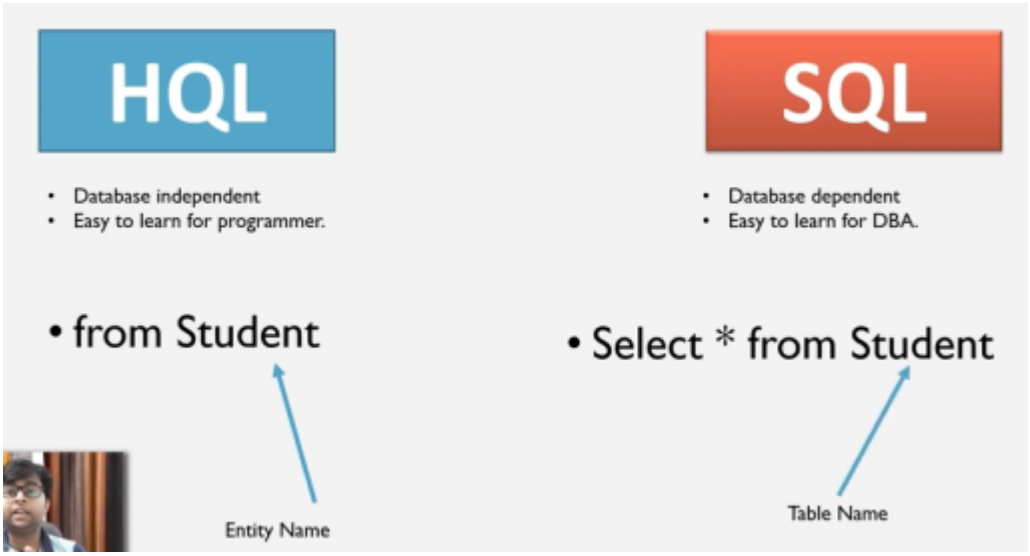
2. Why Do We Need HQL?

- **Object-Oriented Querying:** Unlike SQL, which works with tables and columns, HQL works with Java classes and their properties. This makes it easier to work with data in the context of the application's object model.
- **Database Independence:** HQL abstracts away the underlying database-specific SQL, allowing for more flexibility and portability. Changes to the database schema or switching databases generally require fewer changes to the HQL queries compared to raw SQL.
- **Enhanced Productivity:** It simplifies complex queries by allowing developers to use an object-oriented syntax, making queries more intuitive and reducing the likelihood of errors.

3. Why Was HQL Created?

- **Object-Relational Mapping (ORM) Support:** HQL was created to work seamlessly with Hibernate's ORM capabilities. It helps bridge the gap between the object-oriented programming model in Java and the relational database model.
- **Complex Queries Made Simple:** With HQL, you can perform complex queries more easily than using raw SQL, especially when dealing with associations and entity relationships.

HQL vs SQL



HQL is **database Independent** and it is used to perform complex quries on DB with hibernate Abstraction.

Steps for Creating HQL Queries :

1) Session Object for Interaction with DB 2) String hql = "HQL Query"; 3) Query Interface obje to creat and run it . Query q = session.createQuery(hql); 4) There are two ways to run and store it a) Singluar Results

Object result = q.getSingleResult();

b) Multiple Results

List<Object> = q.getResultList();

You can also set **Dynamic Values** using :

```
Select * from Table where id = :value;
Query q = session.createQuery(hql);
q.setParameter("value", values);
```

Different Commands Using HQL :

1) **Select :**

Select from TableName where condition;
Query q = session.createQuery(hql);
a) Singluar Results Object result = q.**getSingleResult**(); b) Multiple

Results List<Object> = q.**getResultList**();

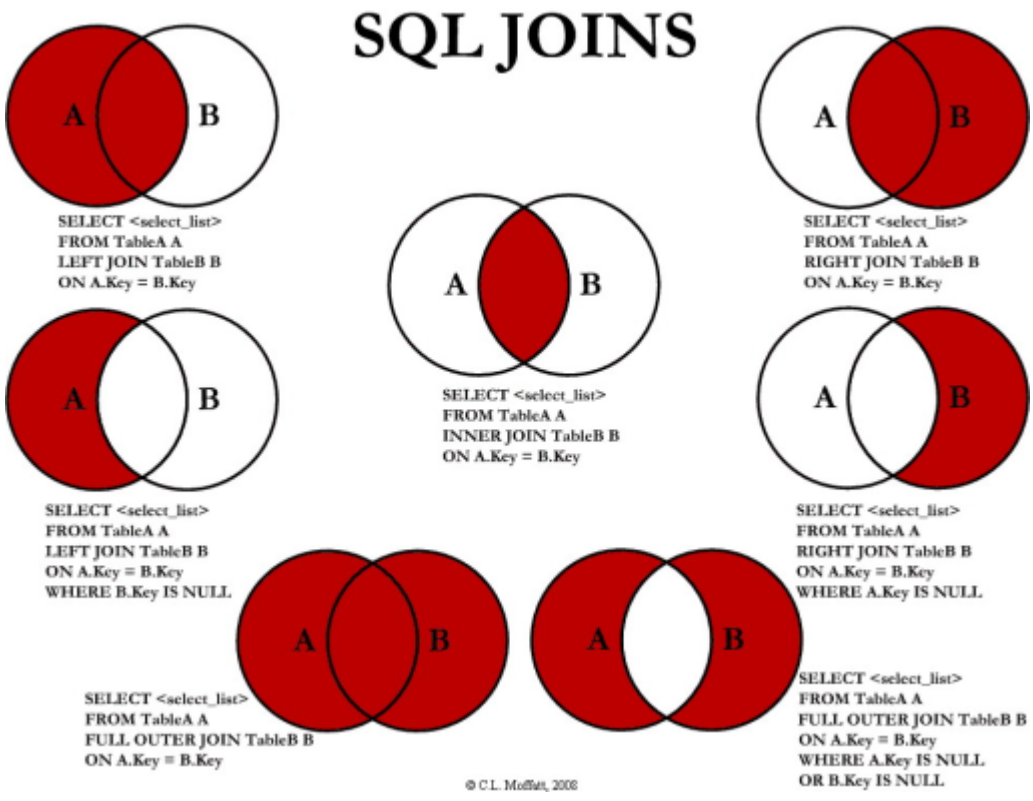
2) **Delete :**

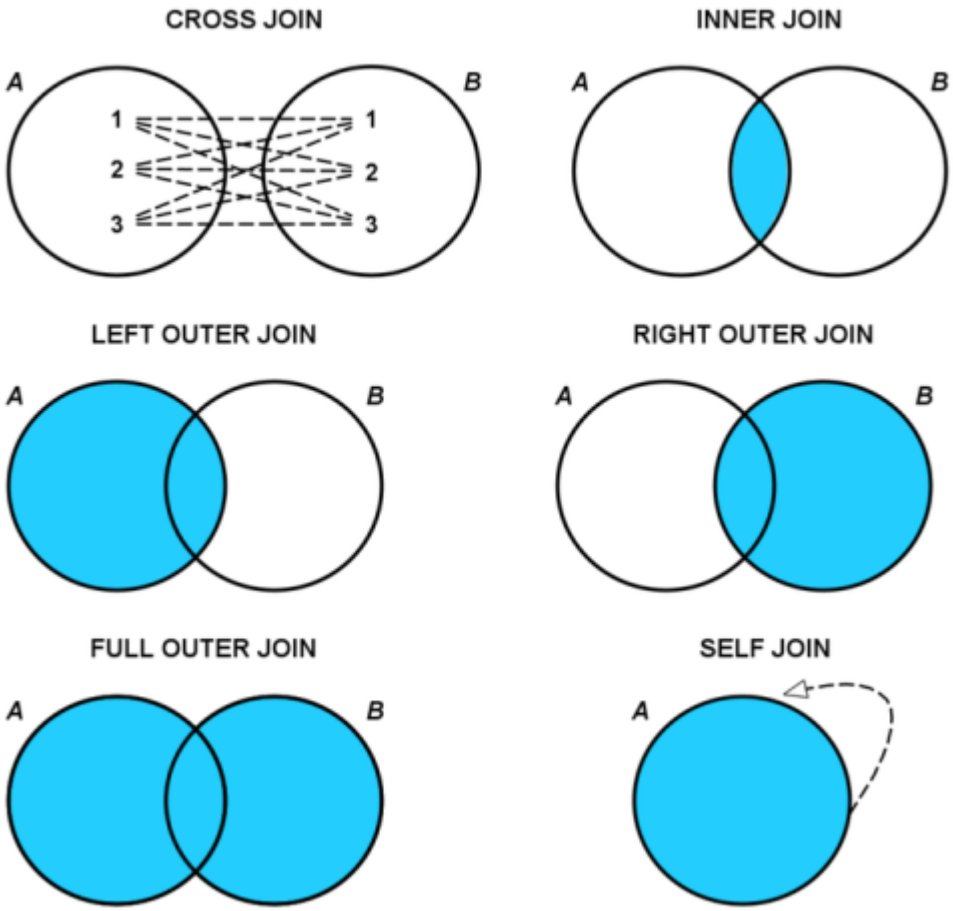
```
DELETE FROM EntityName WHERE condition;
Query q = session.createQuery(hql);
int res = q.executeUpdate();
```

3) **Update :**

```
UPDATE EntityName SET prop1 = value1 WHERE condition;
Query q = session.createQuery(hql);
int res = q.executeUpdate();
```

4) JOIN :





HQL have all joins , same as SQL The basic Syntax for them is :

FROM Entity_1 e (JOIN TYPE) e.relatedField Entity_2 Where Condition;

The return type is based on what u have selected like , if its **q , a** then return type would be a **list of objects array** ,if is just **q** or a then it would a **list of objects**

```
"SELECT q FROM Answer a INNER JOIN a.question q";
```

Pagination

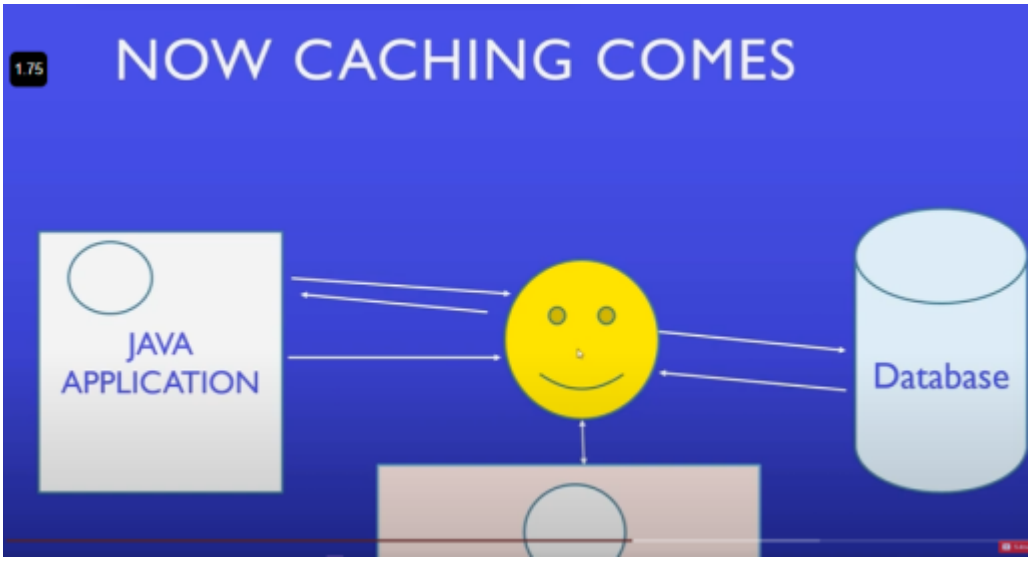
Dividing the fetched result into small small pages (fixed size chunks) to increase performane , and decrease memory load on the program.

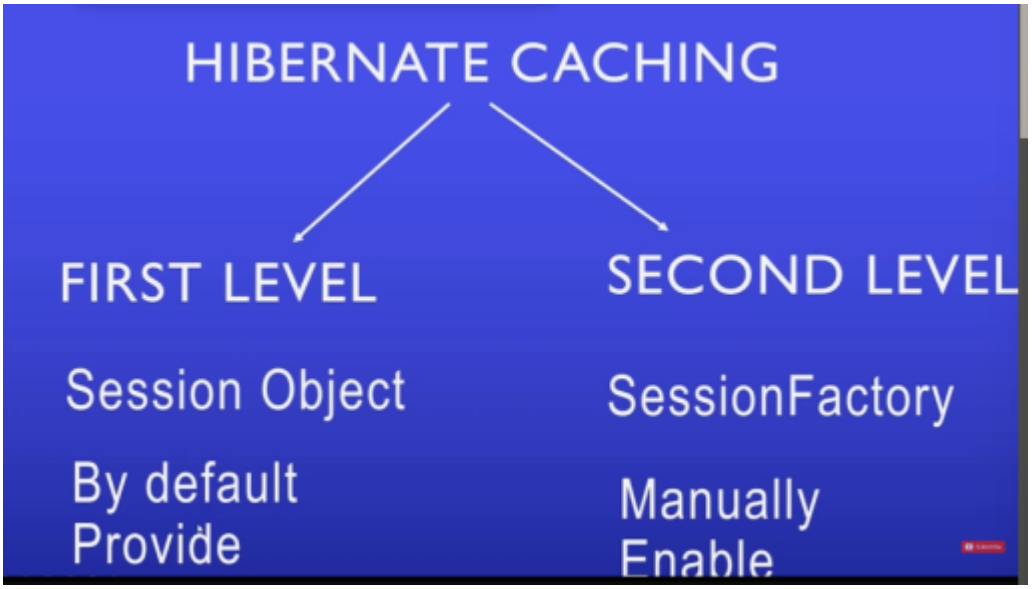
```
.setFirstResult(0);  //starting index for DB
.setMaxResults(5); // Fetch only 5 records
```

This means if there are 10 records being fetched, then only 5 records will be fetched starting from the 0th i.e. the first record.

Caching in Hibernate

Temp store freq accessed data in cache for fast retrieval.
This decreases the number of DB Hits, thus increasing Efficiency.





- Similar to L1 caching, but instead of being tied to a single session, L2 cache is shared across sessions.
- If an entity is not found in L1 cache, Hibernate checks the L2 cache before hitting the database.
- If found in L2 cache, the entity is stored in L1 cache for the current session and returned.