HQL overview in Hibernate

We have understood how to perform various operations like **Insert,Update,delete and Select** on single row of a table in Hibernate.

We have different methods in Hibernate like **save,get,delete and update** methods to perform the same

But what if we want to **Select or Update or Insert** all the records of a table **based on some condition** ?

Consider the below scenarios

What if we want to select all the records of a table ?

What if we want to join 2 tables and get the result ?

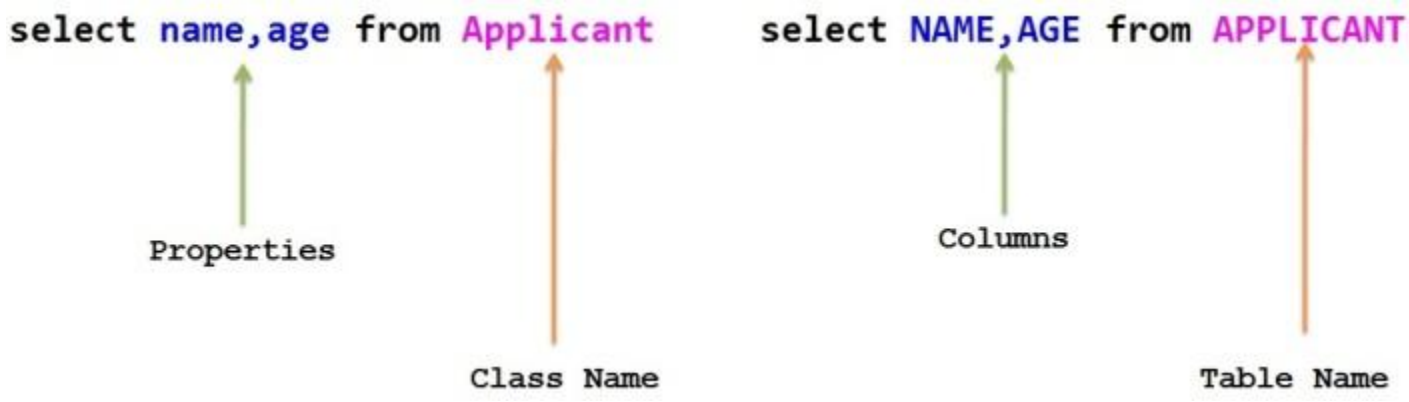
What if we want to perform aggregations on columns like avg(salary) of Employees.

Answer to all the above queries is "**HQL**".

Yes Hibernate Query Language (HQL) is an object oriented query language provided by **Hibernate**.

HQL is very similar to SQL except that we use **Class** names instead of **table** names and **attributes** of a class instead of **Columns** of a table that makes it more close to object oriented programming.
Same thing is shown in the below figure

D



HQL **SQL**

HQL is **case-insensitive** except for java class names and attribute names.

So **SeLeCT** is the **same as sELEct** which is **same as SELECT**, but '**com.kb.model.Applicant**' is **not same** as '**com.kb.model.APPLICANT**'.

HQL examples

1.Select Query examples

Select all the records of Applicant table.

Where **Applicant** is the **class name**.

Equivalent SQL query

```
Select  *  from APPLICANT
```

Where **APPLICANT** is the **table name**

Select all the records of Applicant table whose age >30

```
Query query = session.createQuery("from Applicant  where  age >:age ");

query.setParameter("age", 30);

List list = query.list();.
```

Where Applicant is the class name and **age** is the **attribute** name inside **Applicant class**.

Equivalent SQL query

```
Select * from APPLICANT where AGE >30
```

Where **APPLICANT** is the **table name** and **AGE** is the **column name.**

Select specific columns of Applicant table

```
Query query = session.createQuery("select a.name,a.age from Applicant a");
List<Object[]> applicants= (List<Object[]>)query.list();
```

Where Applicant is the class name , **age** and **name** are the **attribute** names inside **Applicant class**

Equivalent SQL query

```
select a.NAME,a.AGE from APPLICANT a
```

Where APPLICANT is the table name ,**AGE** and **NAME** are the **column names** in **APPLICANT table**.

2. Update Query example

```
Query query = session.createQuery("update Applicant  set  age =:age  where
id=:id");

query.setParameter("age", 30);

query.setParameter("id", 1);

int result = query.executeUpdate();
```

Where **Applicant** is the class name ,**id** and **age** are the **attribute** names inside **Applicant class**.

Equivalent SQL query

```
update APPLICANT  set  AGE =30  where  ID=1
```

Where **APPLICANT** is the **table name** and **AGE and ID** are the **column names**

3. Delete Query example

```
Query query = session.createQuery("delete from  Applicant  where  id=:id");

query.setParameter("id", 1);

int result = query.executeUpdate();
```

Where **Applicant** is the class name ,**id** is the **attribute** name inside **Applicant class**.

Equivalent SQL query

```
Delete from  APPLICANT  where  ID=1
```

Where **APPLICANT** is the **table name** and **ID** is the **column name**

4.Insert Query example

HQL **INSERT can not** be used to insert the records **directly** but it can be used to **insert the records** retrieved from another entity using **Select** statement.

So "***Insert into … values(…..)***" is not supported in HQL as it supports in SQL.

We need to use Insert but with Select.

```
Query query = session.createQuery("insert into  ApplicantBackup(id,name,age)
select id,name,age from  Applicant");

int result = query.executeUpdate();
```

Where **Applicant** and **ApplicantBackup** are the class names

id,name and age are the **attribute** names inside Applicant and ApplicantBackup classes.

Equivalent SQL query

```
Insert into  APPLICANT_BACKUP(ID,NAME,AGE) values(1,"john",30);
```

In SQL we can **directly** do insert records with values without using Select, Where **APPLICANT_BACKUP** is the **table name** and **ID,NAME and AGE** are the **column names**

Note: The **query.executeUpdate()** method returns number of records inserted, updated or deleted.

5.Aggregate Query examples

select count(*) from Applicant

select avg(age) from Applicant

select max(age) from Applicant

select min(age) from Applicant

Where Applicant is the **class name** and **age** is the **attribute** name inside **Applicant class**

Advantages of HQL

We know the benefit of Java being platform independent , similary **HQL is database independent**.

It means queries written using **HQL syntax** will be able to execute in **all the Databases** without any **modification** in it.

HQL supports object oriented features like **Inheritance, polymorphism, Associations**(Relation ships) using which we can perform **complex queries** using Join

HQL also supports DDL operations like **Insert,update** along with **select** operation.

HQL syntax is **very similar to SQL** and hence learning HQL does not require huge effort if you know SQL already.

HQL can be used to perform **bulk operations** like selecting all the records of a Table or updating all records of a table etc.

HQL also supports **query parameters** to set the values **dynamically**.

Note:
Apart from HQL, Hibernate also supports native SQL queries using org.hibernate.SQLQuery interface

Let us understand Hibernate Criteria Query Language (HCQL)

What is HCQL

It's a Criteria based query language mainly used to **fetch the records** based on specific search **criteria**.

It supports complete object oriented approach for querying and retrieving the result from database based on search criteria.

HCQL **can not** be used to perform **DML operations** like Insert,Update and Delete.

It can be used **only for retrieving the records** based on search conditions.

Criteria queries should be **preferred** when we have **many optional** search condtions.

org.hibernate.Criteria interface has provided several methods to add search conditions.

Most commonly used methods in Criteria are

public Criteria add(Criterion c):
This method is used to add restrictions on the search results.

public Criteria addOrder(Order o)
This method is used to define the ordering of result like ascending or descending.

public Criteria setFirstResult(int firstResult)

public Criteria setMaxResult(int totalResult)

These 2 methods are used to achieve pagination by specifying first and maximum records to be retrieved.

Example: If there are 50 records in database and if we are defining the pagination with 25 records per page
FirstResult – 1 and MaxResult – 25 and then FirstResult-26 and MaxResult -25

public List list()
This method returns the list of object on which we are searching.

public Criteria setProjection(Projection projection)
This method is used to set the projection to retrieve only specific columns in the result.

Possible restriction used in HCQL are

lt(*less than*)
le(*less than or equal*)
gt(*greater than*)
ge(*greater than or equal*)
eq(*equal*)
ne(*not equal*)
between
like

Criteria with Projection

Projections will become **handy** when we want to load the **partial object**.

Partial object means **only few attributes** will be loaded rather than **all the attributes**.

In some cases, it is unnecessary to load all the attributes of an object.

Main points to remember about Projections

Projection is an Interface defined in “org.hibernate.criterion” package

Projections is a class and it is a factory for producing the projection objects.

Projection is mainly used to **retrieve partial object**.

To add a Projection object to Criteria , we need to call a **setProjection()** method on Criteria.

We can **add as many projection objects** as we want but **only latest object** will be **considered**, so no use in adding multiple projection objects to criteria.

We need to create one projection object for specifying one property.

If we want **more than one property** to be included in the result, then we need to **create multiple Projection objects** one for each property and add all of them to **ProjectionList** and then we need to set **ProjectionList object** to Criteria.

Example:
Below example illustrates retrieving only one column of a table using Projection.

```
1.  Criteria criteria = session.createCriteria(Employee.class);
```

```
2.  Projection projection = Projections.property("firstName");
3.  criteria.setProjection(projection);
4.  List list = criteria.list();
```

In the above example, we applied Projection on only one property "**FirstName**".

The equivalent SQL query is "**select firstName from Employee**".

If we want to apply projections to **retrieve multiple properties,** we can use **ProjectionList** as shown below

```
1.  Criteria criteria = session.createCriteria(Employee.class);
2.  Projection projection1 = Projections.property("firstName");
3.  Projection projection2 = Projections.property("salary");
4.  Projection projection3 = Projections.property("age");
5.
6.  ProjectionList projectionList = Projections.projectionList();
7.  projectionList.add(projection1);
8.  projectionList.add(projection2);
9.  projectionList.add(projection3);
10. criteria.setProjection(projectionList);
11. List list = criteria.list();
```

Named Queries in Hibernate

If we want to use **same queries in multiple places** of an application, then instead of writing same query in multiple places,

we can **define the query in one place** with the name assigned to it and use that name in all the places wherever required.

The concept of defining the query with name in one place and accessing that query by using its query name wherever required is called Named Query.

Named query will be defined in the **hibernate mapping file**.

Named query can be used for both **HQL queries** and **Native SQL queries**.

Hibernate Named query example with HQL

```
< query name="hql_select_employee" >from Employee  e  where e.id=:empId< /query >
```

For HQL named query, **< query >** tag should be used.

we have given the **name** to it, so that we can access the query with that **name** from required class.

We can also pass the **dynamic parameters** as well to named query as shown above

Accessing the HQL named query in our class

```
Query query = session.getNamedQuery("hql_select_employee");

query.setParameter("empId",new Integer(1));
```

Hibernate Named query example with SQL

```
< sql-query name="sql_select_employee" >select * from Employee e where e.id=:empId</sql-query>
```

For SQL named query, **< sql-query >** tag should be used.

we have given the **name** to it, so that we can access the query with that **name** from required class.

We can also pass the **dynamic parameters** as well to named query as shown above.

Accessing the SQL named query in our class

```
Query query = session.getNamedQuery("sql_select_employee");

query.setParameter("empId",new Integer(1));
```

We can define **Named query** using either **XML** mapping or **Annotation** mapping

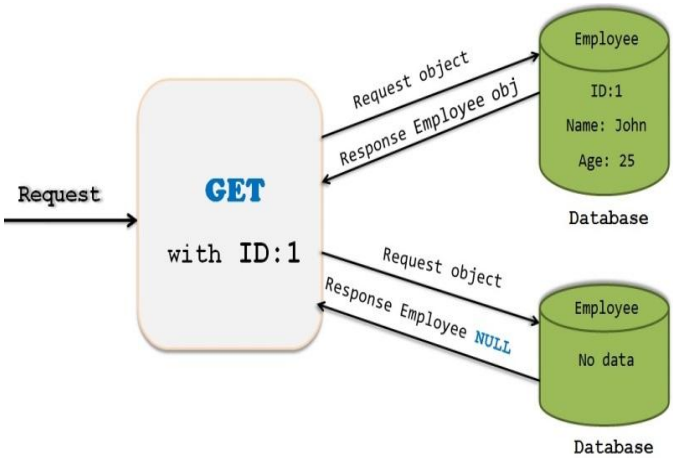
GET Vs LOAD: difference between Get and LOAD method:

Session.get() and Session.load() are decalred in session.

End use of both the methods is to retrieve the object(single row) from the DB.

GET: when we use session.get() method,it will hit the db immediately and returns the real object from DB.

If there is no row corresponding to requested indentifier exists, it will NULL.

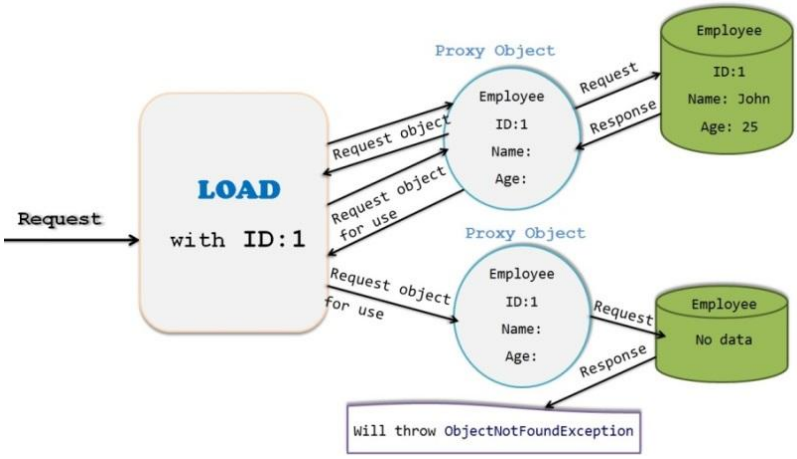


LOAD:

when we use session.load() method,it will always returns the proxy object without hitting the DB.

Proxy object is a temporary Object which does not have any values assigned to it except primary key.

If there is no row corresponding to requested indentifier exists, it will throw "Object Not Found Exception"


Request
LOAD
with ID:1
Request object
Request object for use
Proxy Object
Employee
ID:1
Name:
Age:
Request
Response
Employee
ID:1
Name: John
Age: 25
Request object for use
Proxy Object
Employee
ID:1
Name:
Age:
Request
Response
Employee
No data
Will throw ObjectNotFoundException