Hibernate Query Language (HQL) Example Tutorial

March 15, 2015 by Mukesh Kumar at 3:47 pm

- Hibernate Framework comes with a powerful object-oriented query language – Hibernate Query Language (HQL). Hibernate Query Language is same as SQL (Structured Query Language) but instead of tables it deals with classes and instead of columns it deals with properties or class attributes.
- HQL queries are database independent queries because HQL queries internally converted into database specific SQL queries using Dialect class mentioned in hibernate-cfg.xml file.

Advantage of HQL

- Database independent
- Easy to learn for Java Programmer
- HQL fully supports polymorphic queries. That is, along with the object to be returned as a query result, all child objects (objects of subclasses) of the given object shall be returned.

HQL Syntax

Most of HQL's syntax and features are very similar to SQL. An HQL query may consist of following elements:

- Clauses
- Aggregate functions
- Subqueries

Clauses

Some of the commonly supported clauses in HQL are:

- from
- as
- select
- where
- order by
- group by
- update
- delete
- insert

from clause :

From clause is the most important hibernate query. This clause is used to load a complete persistent object into memory.

Syntax:-

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

as clause :

as clause will allow to assign aliases to the classes.

Syntax:-

```
String hql = "FROM Employee AS E";
Query query = session.createQuery(hql);
List results = query.list();
```

The "as" keyword is optional—you can also specify the alias directly after the class name, as mentioned below:

```
from Employee e
```

select clause :

The select clause is used to obtain few properties of objects instead of the complete object.

Syntax:-

```
String hql = "SELECT e.firstName FROM  Employee e";
Query query = session.createQuery(hql);
List results = query.list();
```

where clause :

The where clause is used to fetch the specific objects from the database.

Syntax:-

```
String hql = "FROM Employee e WHERE e.id = 100";
Query query = session.createQuery(hql);
List results = query.list();
```

Order by clause :

This clause is used to sort our HQL query's results. We can arrange the result by any property on the objects in the result set either ascending (ASC) or descending (DESC) order.

Syntax:-

```
String hql = "FROM Person P WHERE P.id > 10 ORDER BY P.salary DESC";
Query query = session.createQuery(hql);
```

group by clause :

Group by clause Takes information from database ant group it based on a value of an attribute and typically, use the result to include an aggregate value. It returns aggregate values which is grouped by any property of a returned class or component.

Syntax:-

```
String hql = "SELECT SUM(P.salary), P.firtName FROM Person P” +
"GROUP BY P.firstName";
Query query = session.createQuery(hql);
List results = query.list();
```

update clause :

The UPDATE clause can be used to update one or more properties of one or more objects.

Syntax:-

```
String hql = "UPDATE Employee set salary =: salary "+
"WHERE id =:empId";
Query query = session.createQuery(hql);
query.setParameter("salary", 10000);
query.setParameter("empId", 10);
int result = query.executeUpdate();
System.out.println("Rows Affected: " + result);
```

DELETE clause :

The DELETE clause can be used to delete one or more objects.

Syntax:-

```
String hql = "DELETE FROM Employee "+
"WHERE id = :empId";
Query query = session.createQuery(hql);
query.setParameter("empId", 10);
int result = query.executeUpdate();
System.out.println("Rows Affected: " + result);
```

Insert clause :

Insert into clause of hibernate query language support where records can be inserted from one object to another object.

Syntax:-

```
String hql = "INSERT INTO Person(firstName, lastName, salary)"  +
"SELECT firstName, lastName, salary FROM old_person";
```

```
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows Affected: " + result);
```

Aggregate Functions

- avg(...), sum(...), min(...), max(...)
- count(*) ,count(...), count(distinct ...), count(all...)

Subqueries

Subqueries are nothing but its a query within another query. Hibernate supports Subqueries if the underlying database supports it.

Steps to work with HQL

Step 1 :

Write the HQL query as per requirement.For Example:

```
from Employee
```

Step 2 :

Obtain an instance of org.hibernate.Query by passing HQL query.For Example:

```
Query q = session.createQuery("from Employee");
```

Step 3 :

If the query contains parameter set those values.For Example:

```
query.setParameter(index,value);
```

Step 4 : Execute the Query.

we need to call list() method for executing select query on database, it will return java.util.List.For update and delete queries we need to call executeUpdate() method.For Example:

```
query.list(); //for executing select queries
query.executeUpdate();//for executing update/delete queries
```

Step 5 : Iterate the List

we need to iterate the List collection.For Example:

```
Query q = session.createQuery("from Employee");
List<employee> empList= q.list();
for(Employee employee : empList) {
System.out.println(employee.getEmpno());
System.out.println(employee.getUserName());
</employee>
```

Query :

- ” Query ” is an interface given in org.hibernate package.If we want to execute execute an HQL query on a database, we need to create a query object.
- In order to get query object, we need to call createQuery() method in the session Interface.

```
Query q = session.createQuery("from Employee");
```

The Query interface provides many methods.Below are the list of commonly used methods

- **public int executeUpdate()** : is used to execute the update and delete queries.
- **public List list()** : returns the result of the select query as list.
- **public Query setFirstResult(int rowno)**: specifies the row no from where record will be retrieved.
- **public Query setMaxResult(int rowno)**: specifies the no of records to be retrieved from the table.
- **public Query setParameter(int position , Object value)**: It sets the value to the JDBC style query parameter.
- **public Query setParameter(String name , Object value)**: It sets the value to a named query parameter.

Different Ways to write HQL Select query

The from Clause and Aliases

The most important feature in HQL is the alias. Hibernate allows you to assign aliases to the classes in your query with the as clause. Consider the example given below.

```
from Employee as e
```

or

```
from Employee as employee
```

The "as" keyword is optional—you can also specify the alias directly after the class name, as mentioned below:

```
from Employee e
```

The above query returns all the Employee objects in the database along with all associated object and non-lazy collections.

Fetching Complete Object :

If we want to select a Complete Object from the database, we need to use POJO class reference in place of * while writing the query.Example:

In SQL

```
select * from Employee
```

In HQL

```
select e from Employee e
```

or

```
from Employee e
```

Fetching Partial Object :

If we want to load the Partial Object from the database then we need to replace column names with POJO class variable names.Example:

In SQL

```
select empid,name from Employee
```

Note: empid, name are the columns Employee is the table.

In HQL

```
select e.empId,e.name from Employee e
```

It is also possible to load or select the object from the database by passing run time values into the query, in this case we can use either ” ? ” symbol or : label in an HQL command, the index number of ” ? ” will starts from zero and not one. select e from Employee e where e.name=?

```
[ or ]
select e from Employee e where e.name=:mukesh
[ or ]
from Employee e where e.name=?
[ or ]
from Employee e where e.name=:mukesh
```

Hibernate Parameter Binding

Parameter binding is the process of binding a Java variable with an HQL statement.Hibernate parameter binding is like prepared statements in any normal SQL language.

A simple select query Without parameter binding

```
String name="Mukesh";
Query query = session.createQuery("from Employee where employeeName =
'"+name+"' ");
```

There is two types of query parameter binding in the Hibernate,positional parameter and named parameter.Hibernate recommend to use the named parameters since it is more flexible and powerful compare to the positional parameter.

1. Named parameters

In named parameters query string will use parameters in the variable name that can be replaced at runtime.The actual value will be substituted at runtime using the setParameter() method.

You define a named place holder by writing the place holder name after a colon (:) as :holder name and use it within a setXXX method without the colon (:) as "holder name".

setParameter

The setParameter is smart enough to discover the parameter data type for you.

```
Query query = session.createQuery("from Student where studentId = :id ");
query.setParameter("id", 5);
```

setString

You can use setString to tell Hibernate this parameter date type is String.

```
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql)
.setString("stockCode", "1234")
.list();
```

The above code tells Hibernate to insert an object of type String in the query. You can also use setInteger,setBoolean etc for the corresponding types.

setProperties

By using setProperties you can pass an object into the parameter binding. Hibernate will automatic check the object's properties and match with the colon parameter.

```
Stock stock = new Stock();
stock.setStockCode("1234");
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql)
.setProperties(stock)
.list();
```

2. Positional parameters

This approach will use question mark (?) to define a named parameter, and you have to set your parameter according to the position sequence. Example:

```
String hql = "from Stock s where s.stockCode = ? and s.stockName = ?";
List result = session.createQuery(hql)
.setString(0, "1234")
.setParameter(1, "HUL")
.list();
```

The basic disadvantage of this method is that if you change the orders of the parameters on the query you have to change the parameter binding code. For Example:

```
String hql = "from Stock s where s.stockName = ? and s.stockCode = ?";
List result = session.createQuery(hql)
.setParameter(0, "HUL")
.setString(1, "1234")
.list();
```

Selecting fields in HQL :

The select clause is used to obtain few properties of objects instead of the complete object.

The "from ClassName" syntax returns a List of the class type with all fields in it. You can instead select one or more fields selectively than the table itself. If you are selecting a single column in the result, it will return a List of that field type.

Syntax: –

```
Query query = session.createQuery("Select name from User");

    List<String> names = (List<String>) query.list();

    for (String n : names) {

      System.out.println(n);

    }

```

Selecting multiple fields in HQL

If you select more than one field in an HQL statement, query.list will return a list of lists, one list each for each field.

Syntax: –

```
Query query = session.createQuery("Select id, name from User");

    List<Object[]> users= (List<Object[]>)query.list();

    for(Object[] user: users){

        Integer id = (Integer)user[0];

        System.out.println(id);

        String name = (String)user[1];

        System.out.println(name);

    }
```