Hibernate Query Language (HQL) Example

The Hibernate ORM framework provides its own query language called Hibernate Query Language or HQL for short. It is very powerful and flexible and has the following characteristics:

- **SQL similarity:** HQL's syntax is very similar to standard SQL. from SELECT, FROM, ORDER BY to arithmetic expressions and aggregate functions, etc.
- **Fully object-oriented:** HQL doesn't use real names of table and columns. It uses class and property names instead. HQL can understand inheritance, polymorphism and association.
- **Case-insensitive for keywords:** Like SQL, keywords in HQL are case-insensitive. That means SELECT, select or Select are the same.
- **Case-sensitive for Java classes and properties:** HQL considers case-sensitive names for Java classes and their properties, meaning Person and person are two different objects.

How to write HQL for executing fundamental queries (CRUD) as well as other popular ones.

The upcoming examples are provided based on assumption that a Hibernate's **SessionFactory** is opened and a transaction has been started.

How to execute HQL in Hibernate

Basically, it's fairly simple to execute HQL in Hibernate. Here are the steps:

- Write your HQL:

```
1  String hql = "Your Query Goes Here";
```

- Create a Query from the Session:

```
1  Query query = session.createQuery(hql);
```

- Execute the query: depending on the type of the query (listing or update), an appropriate method is used:
  - For a listing query (SELECT):

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

  - For an update query (INSERT, UPDATE, DELETE):

```
1  int rowsAffected = query.executeUpdate();
```

- Extract result returned from the query: depending of the type of the query, Hibernate returns different type of result set. For example:
  - Select query on a mapped object returns a list of those objects.
  - Join query returns a list of arrays of Objects which are aggregate of columns of the joined tables. This also applies for queries using aggregate functions (count, sum, avg, etc).
  -

EXAMPLES

1& 2 . Insert - Select Query Example

HQL doesn't support regular INSERT statement (you know why - because the session.save(Object) method does it perfectly). So we can only write INSERT … SELECT query in HQL. The following code snippet executes a query that inserts all rows from Category table to OldCategory table:

```
1  String hql = "insert into Category (id, name)"
2        + " select id, name from OldCategory";
3
4  Query query = session.createQuery(hql);
5
6  int rowsAffected = query.executeUpdate();
7  if (rowsAffected > 0) {
8     System.out.println(rowsAffected + "(s) were inserted");
9  }
```

Note that HQL is object-oriented, so Category and OldCategory must be mapped class names (not real table names).

3. Update Query Example

The UPDATE query is similar to SQL. The following example runs a query that updates price for a specific product:

```
String hql = "update Product set price = :price where id = :id";

Query query = session.createQuery(hql);
query.setParameter("price", 488.0f);
query.setParameter("id", 43l);

int rowsAffected = query.executeUpdate();
if (rowsAffected > 0) {
   System.out.println("Updated " + rowsAffected + " rows.");
}
```

4. Delete Query Example

Using DELETE query in HQL is also straightforward. For example:

```
String hql = "delete from OldCategory where id = :catId";

Query query = session.createQuery(hql);
query.setParameter("catId", new Long(1));

int rowsAffected = query.executeUpdate();
if (rowsAffected > 0) {
   System.out.println("Deleted " + rowsAffected + " rows.");
}
```

5. List Query Example

The following code snippet executes a query that returns all Category objects:

```
String hql = "from Category";
Query query = session.createQuery(hql);
List<Category> listCategories = query.list();

for (Category aCategory : listCategories) {
    System.out.println(aCategory.getName());
}
```

Note that in HQL, we can omit the SELECT keyword and just use the FROM instead.

6. Search Query Example

The following statements execute a query that searches for all products in a category whose name is 'Computer':

```
String hql = "from Product where category.name = 'Computer'";
Query query = session.createQuery(hql);
List<Product> listProducts = query.list();

for (Product aProduct : listProducts) {
    System.out.println(aProduct.getName());
}
```

The cool thing here is Hibernate automatically generates JOIN query between the Product and Category tables behind the scene. Thus we don't have to use explicit JOIN keyword:

```
from Product where category.name = 'Computer'
```

7. Using Named Parameters Example

You can parameterize your query using a colon before parameter name, for example **:id** indicates a placeholder for a parameter named **id**. The following example demonstrates how to write and execute a query using named parameters:

```
String hql = "from Product where description like :keyword";

String keyword = "New";
```

```
4     Query query = session.createQuery(hql);
5     query.setParameter("keyword", "%" + keyword + "%");
6
7     List<Product> listProducts = query.list();
8
9     for (Product aProduct : listProducts) {
10        System.out.println(aProduct.getName());
11    }
```

The above HQL searches for all products whose description contains the specified keyword:

```
1    from Product where description like :keyword
```

Then use the **setParameter(name, value)** method to set actual value for the named parameter:

```
1    query.setParameter("keyword", "%" + keyword + "%");
```

Note that we want to perform a LIKE search so the percent signs must be used outside the query string, unlike traditional SQL.

8. Join Query Example

HQL supports the following join types (similar to SQL):

- **inner join** (can be abbreviated as **join**).
- **left outer join** (can be abbreviated as **left join**).
- **right outer join** (can be abbreviated as **right join**).
- **full join**

For example, the following code snippet executes a query that retrieves results which is a join between two tables Productand Category:

```
1     String hql = "from Product p inner join p.category";
2
3     Query query = session.createQuery(hql);
4     List<Object[]> listResult = query.list();
5
6     for (Object[] aRow : listResult) {
7         Product product = (Product) aRow[0];
8         Category category = (Category) aRow[1];
9         System.out.println(product.getName() + " - " + category.getName());
```

```
10  }
```

Using the **join** keyword in HQL is called **explicit join**. Note that a JOIN query returns a list of Object arrays, so we need to deal with the result set differently:

```
1   List<Object[]> listResult = query.list();
```

HQL provides **with** keyword which can be used in case you want to supply extra join conditions. For example:

```
1   from Product p inner join p.category with p.price > 500
```

That joins the Product and Category together with a condition specifies that product's price must be higher than 500.

As stated earlier, we can write **implicit join** query which uses dot-notation. For example:

```
1   from Product where category.name = 'Computer'
```

That result in **inner join** in the resulting SQL statement.

9. Sort Query Example

Sorting in HQL is very similar to SQL using ORDER BY clause follows by a sor direction ASC(ascending) or DESC(descending).

```
String hql = "from Product order by price ASC";

Query query = session.createQuery(hql);
List<Product> listProducts = query.list();

for (Product aProduct : listProducts) {
   System.out.println(aProduct.getName() + "\t - " + aProduct.getPrice());
}
```

10. Group By Query Example

Using GROUP BY clause in HQL is similar to SQL. The following query summarizes price of all products grouped by each category.

```
select sum(p.price), p.category.name from Product p group by category
```

```
String hql = "select sum(p.price), p.category.name from Product p group by category";

Query query = session.createQuery(hql);
List<Object[]> listResult = query.list();

for (Object[] aRow : listResult) {
   Double sum = (Double) aRow[0];
String category = (String) aRow[1];
   System.out.println(category + " - " + sum);
}
```
