HIBERNATE - QUERY LANGUAGE

Hibernate Query Language *HQL* is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries which in turns perform action on database.

Although you can use SQL statements directly with Hibernate using Native SQL but I would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies.

Keywords like SELECT , FROM and WHERE etc. are not case sensitive but properties like table and column names are case sensitive in HQL.

FROM Clause

You will use **FROM** clause if you want to load a complete persistent objects into memory. Following is the simple syntax of using FROM clause:

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

If you need to fully qualify a class name in HQL, just specify the package and class name as follows:

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

AS Clause

The **AS** clause can be used to assign aliases to the classes in your HQL queries, specially when you have long queries. For instance, our previous simple example would be the following:

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

The **AS** keyword is optional and you can also specify the alias directly after the class name, as follows:

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

SELECT Clause

The **SELECT** clause provides more control over the result set than the from clause. If you want to obtain few properties of objects instead of the complete object, use the SELECT clause. Following is the simple syntax of using SELECT clause to get just first_name field of the Employee object:

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

It is notable here that **Employee.firstName** is a property of Employee object rather than a field of the EMPLOYEE table.

WHERE Clause

If you want to narrow the specific objects that are returned from storage, you use the WHERE clause. Following is the simple syntax of using WHERE clause:

```
String hql = "FROM Employee E WHERE E.id = 10";
Query query = session.createQuery(hql);
List results = query.list();
```

ORDER BY Clause

To sort your HQL query's results, you will need to use the **ORDER BY** clause. You can order the results by any property on the objects in the result set either ascending *ASC* or descending *DESC*. Following is the simple syntax of using ORDER BY clause:

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query query = session.createQuery(hql);
List results = query.list();
```

If you wanted to sort by more than one property, you would just add the additional properties to the end of the order by clause, separated by commas as follows:

```
String hql = "FROM Employee E WHERE E.id > 10 " +
             "ORDER BY E.firstName DESC, E.salary DESC ";
Query query = session.createQuery(hql);
List results = query.list();
```

GROUP BY Clause

This clause lets Hibernate pull information from the database and group it based on a value of an attribute and, typically, use the result to include an aggregate value. Following is the simple syntax of using GROUP BY clause:

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

Using Named Paramters

Hibernate supports named parameters in its HQL queries. This makes writing HQL queries that accept input from the user easy and you do not have to defend against SQL injection attacks. Following is the simple syntax of using named parameters:

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id",10);
List results = query.list();
```

UPDATE Clause

Bulk updates are new to HQL with Hibernate 3, and deletes work differently in Hibernate 3 than they did in Hibernate 2. The Query interface now contains a method called executeUpdate for executing HQL UPDATE or DELETE statements.

The **UPDATE** clause can be used to update one or more properties of an one or more objects. Following is the simple syntax of using UPDATE clause:

```
String hql = "UPDATE Employee set salary = :salary "  +
             "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("salary", 1000);
query.setParameter("employee_id", 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. Following is the simple syntax of using DELETE clause:

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

INSERT Clause

HQL supports **INSERT INTO** clause only where records can be inserted from one object to another object. Following is the simple syntax of using INSERT INTO clause:

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)"  +
             "SELECT firstName, lastName, salary FROM old_employee";
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

Aggregate Methods

HQL supports a range of aggregate methods, similar to SQL. They work the same way in HQL as in SQL and following is the list of the available functions:

S.N.	Functions	Description
1	avg(*propertyname*)	The average of a property's value
2	count(*propertyname*) *or* (*)	The number of times a property occurs in the results
3	max(*propertyname*)	The maximum value of the property values
4	min(*propertyname*)	The minimum value of the property values
5	sum(*propertyname*)	The sum total of the property values

The **distinct** keyword only counts the unique values in the row set. The following query will return only unique count:

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

Pagination using Query

There are two methods of the Query interface for pagination.

S.N.	Method & Description
1	**Query setFirstResult***intstartPosition* This method takes an integer that represents the first row in your result set, starting with row 0.
2	**Query setMaxResults***intmaxResult*

This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

Using above two methods together, we can construct a paging component in our web or Swing application. Following is the example which you can extend to fetch 10 rows at a time:

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
query.setFirstResult(1);
query.setMaxResults(10);
List results = query.list();
```