**Hibernate Query Language (HQL)**

Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.

Advantage of HQL

There are many advantages of HQL. They are as follows:

* database independent
* supports polymorphic queries
* easy to learn for Java Programmer

Query Interface

It is an object oriented representation of Hibernate Query. The object of Query can be obtained by calling the createQuery() method Session interface.

The query interface provides many methods. There is given commonly used methods:

1. **public int executeUpdate()** is used to execute the update or delete query.
2. **public List list()** returns the result of the ralation as a list.
3. **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.
4. **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
5. **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
6. **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

**Example of HQL to get all the records**

Query query=session.createQuery("from Emp");//here persistent class name is Emp

List list=query.list();

**Example of HQL to get records with pagination**

Query query=session.createQuery("from Emp");

query.setFirstResult(5);

query.setMaxResult(10);

List list=query.list();//will return the records from 5 to 10th number

**Example of HQL update query**

Transaction tx=session.beginTransaction();

Query q=session.createQuery("update User set name=:n where id=:i");

q.setParameter("n","Udit Kumar");

q.setParameter("i",111);

**int** status=q.executeUpdate();

System.out.println(status);

tx.commit();

**Example of HQL delete query**

Query query=session.createQuery("delete from Emp where id=100");

//specifying class name (Emp) not tablename

query.executeUpdate();

**HQL with Aggregate functions**

You may call avg(), min(), max() etc. aggregate functions by HQL. Let's see some common examples:

**Example to get total salary of all the employees**

Query q=session.createQuery("select sum(salary) from Emp");

List<Integer> list=q.list();

System.out.println(list.get(0));

**Example to get maximum salary of employee**

Query q=session.createQuery("select max(salary) from Emp");

**Example to get minimum salary of employee**

Query q=session.createQuery("select min(salary) from Emp");

**Example to count total number of employee ID**

Query q=session.createQuery("select count(id) from Emp");

**Example to get average salary of each employees**

Query q=session.createQuery("select avg(salary) from Emp");

**HCQL (Hibernate Criteria Query Language)**

The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria. The Criteria interface provides methods to apply criteria such as retreiving all the records of table whose salary is greater than 50000 etc.

**Advantage of HCQL**

The HCQL provides methods to add criteria, so it is **easy** for the java programmer to add criteria. The java programmer is able to add many criteria on a query.

**Criteria Interface**

The Criteria interface provides many methods to specify criteria. The object of Criteria can be obtained by calling the **createCriteria()** method of Session interface.

Syntax of createCriteria() method of Session interface

public Criteria createCriteria(Class c

The commonly used methods of Criteria interface are as follows:

1. **public Criteria add(Criterion c)** is used to add restrictions.
2. **public Criteria addOrder(Order o)** specifies ordering.
3. **public Criteria setFirstResult(int firstResult)** specifies the first number of record to be retreived.
4. **public Criteria setMaxResult(int totalResult)** specifies the total number of records to be retreived.
5. **public List list()** returns list containing object.
6. **public Criteria setProjection(Projection projection)** specifies the projection.

### Restrictions class

Restrictions class provides methods that can be used as Criterion. The commonly used methods of Restrictions class are as follows:

1. **public static SimpleExpression lt(String propertyName,Object value)** sets the **less than** constraint to the given property.
2. **public static SimpleExpression le(String propertyName,Object value)** sets the **less than or equal** constraint to the given property.
3. **public static SimpleExpression gt(String propertyName,Object value)** sets the **greater than** constraint to the given property.
4. **public static SimpleExpression ge(String propertyName,Object value)** sets the **greater than or equal** than constraint to the given property.
5. **public static SimpleExpression ne(String propertyName,Object value)** sets the **not equal** constraint to the given property.
6. **public static SimpleExpression eq(String propertyName,Object value)** sets the **equal** constraint to the given property.
7. **public static Criterion between(String propertyName, Object low, Object high)** sets the **between** constraint.
8. **public static SimpleExpression like(String propertyName, Object value)** sets the **like** constraint to the given property.

### Order class

The Order class represents an order. The commonly used methods of Restrictions class are as follows:

1. **public static Order asc(String propertyName)** applies the ascending order on the basis of given property.
2. **public static Order desc(String propertyName)** applies the descending order on the basis of given property.

### Examples of Hibernate Criteria Query Language

There are given a lot of examples of HCQL.

### Example of HCQL to get all the records

1. Crietria c=session.createCriteria(Emp.**class**);//passing Class class argument
2. List list=c.list();

**Example of HCQL to get the 10th to 20th record**

1. Crietria c=session.createCriteria(Emp.**class**);
2. c.setFirstResult(10);
3. c.setMaxResult(20);
4. List list=c.list();

**Example of HCQL to get the records whose salary is greater than 10000**

1. Crietria c=session.createCriteria(Emp.**class**);
2. c.add(Restrictions.gt("salary",10000));//salary is the propertyname
3. List list=c.list();

**Example of HCQL to get the records in ascending order on the basis of salary**

1. Crietria c=session.createCriteria(Emp.**class**);
2. c.addOrder(Order.asc("salary"));
3. List list=c.list();

### HCQL with Projection

We can fetch data of a particular column by projection such as name etc. Let's see the simple example of projection that prints data of NAME column of the table only.

1. Criteria c=session.createCriteria(Emp.**class**);
2. c.setProjection(Projections.property("name"));
3. List list=c.list();

**Hibernate Named Query**

The hibernate named query is way to use any query by some meaningful name. It is like using alias names. The Hibernate framework provides the concept of named queries so that application programmer need not to scatter queries to all the java code.

There are two ways to define the named query in hibernate:

* by annotation
* by mapping file.

### Hibernate Named Query by annotation

If you want to use named query in hibernate, you need to have knowledge of @NamedQueries and @NamedQuery annotations.

**@NameQueries** annotation is used to define the multiple named queries.

**@NameQuery** annotation is used to define the single named query.

Let's see the example of using the named queries:

1. @NamedQueries(
2. {
3. @NamedQuery(
4. name = "findEmployeeByName",
5. query = "from Employee e where e.name = :name"
6. )
7. }
8. )

### Example of Hibernate Named Query by annotation

In this example, we are using annotations to defined the named query in the persistent class. There are three files only:

* Employee.java
* hibernate.cfg.xml
* FetchDemo

In this example, we are assuming that there is em table in the database containing 4 columns id, name, job and salary and there are some records in this table.

#### Employee.java

It is a persistent class that uses annotations to define named query and marks this class as entity.

1. **package** com.javatpoint;
3. **import** javax.persistence.\*;
4. **import** javax.persistence.Entity;
5. **import** javax.persistence.GeneratedValue;
6. **import** javax.persistence.Id;
8. @NamedQueries(
9. {
10. @NamedQuery(
11. name = "findEmployeeByName",
12. query = "from Employee e where e.name = :name"
13. )
14. }
15. )
17. @Entity
18. @Table(name="em")
19. **public** **class** Employee {
21. **public** String toString(){**return** id+" "+name+" "+salary+" "+job;}
23. **int** id;
24. String name;
25. **int** salary;
26. String job;
27. @Id
28. @GeneratedValue(strategy=GenerationType.AUTO)
30. //getters and setters
31. }

#### hibernate.cfg.xml

It is a configuration file that stores the informations about database such as driver class, url, username, password and mapping class etc.

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3. "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
6. <hibernate-configuration>
8. <session-factory>
9. <property name="hbm2ddl.auto">update</property>
10. <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
11. <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
12. <property name="connection.username">system</property>
13. <property name="connection.password">oracle</property>
14. <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>
15. <mapping **class**="com.javatpoint.Employee"/>
16. </session-factory>
18. </hibernate-configuration>

#### FetchData.java

It is a java class that uses the named query and prints the informations based on the query. The **getNamedQuery** method uses the named query and returns the instance of Query.

1. **package** com.javatpoint;
3. **import** java.util.Iterator;
4. **import** java.util.List;
6. **import** org.hibernate.cfg.AnnotationConfiguration;
7. **import** org.hibernate.\*;
9. **public** **class** FetchData {
10. **public** **static** **void** main(String[] args) {
12. AnnotationConfiguration configuration=**new** AnnotationConfiguration();
13. configuration.configure("hibernate.cfg.xml");
14. SessionFactory sFactory=configuration.buildSessionFactory();
15. Session session=sFactory.openSession();
17. //Hibernate Named Query
18. Query query = session.getNamedQuery("findEmployeeByName");
19. query.setString("name", "amit");
21. List<Employee> employees=query.list();
23. Iterator<Employee> itr=employees.iterator();
24. **while**(itr.hasNext()){
25. Employee e=itr.next();
26. System.out.println(e);
27. }
29. session.close();
31. }
32. }

### Hibernate Named Query by mapping file

If want to define named query by mapping file, you need to use **query** element of hibernate-mapping to define the named query.

In such case, you need to create hbm file that defines the named query. Other resources are same as given in the above example except Persistent class Employee.java where you don't need to use any annotation and hibernate.cfg.xml file where you need to specify mapping resource of the hbm file.

The hbm file should be like this:

#### emp.hbm.xml

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3. "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
6. <hibernate-mapping>
7. <**class** name="com.javatpoint.Employee" table="em">
8. <id name="id">
9. <generator **class**="native"></generator>
10. </id>
11. <property name="name"></property>
12. <property name="job"></property>
13. <property name="salary"></property>
14. </**class**>
16. <query name="findEmployeeByName">
17. <![CDATA[from Employee e where e.name = :name]]>
18. </query>
20. </hibernate-mapping>

The persistent class should be like this:

#### Employee.java

1. **package** com.javatpoint;
2. **public** **class** Employee {
3. **int** id;
4. String name;
5. **int** salary;
6. String job;
7. //getters and setters
8. }

Now include the mapping resource in the hbm file as:

#### hibernate.cfg.xml

1. <mapping resource="emp.hbm.xml"/>