# Hibernate Query Language (HQL)

Hibernate Query Language (**HQL**) is same as **SQL** (Structured Query Language) but it doesn't depend 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

Some of the commonly supported clauses in HQL are:

1. HQL From: HQL **From** is same as select clause in SQL, from Employee is same as select \* from Employee. We can also create alias such as from Employee emp or from Employee as emp.
2. HQL Join: HQL supports inner join, left outer join, right outer join and full join. For example, select e.name, a.city from Employee e INNER JOIN e.address a. In this query, Employee class should have a variable named address. We will look into it in the example code.
3. Aggregate Functions: HQL supports commonly used aggregate functions such as count (\*), count(distinct x), min(), max(), avg() and sum().
4. Expressions: HQL supports arithmetic expressions (+, -, \*, /), binary comparison operators (=, >=, <=, <>, !=, like), logical operations (and, or, not) etc.
5. HQL also supports order by and group by clauses.
6. HQL also supports sub-queries just like SQL queries.
7. HQL supports DDL, DML and executing store procedures too.

### Example of HQL to get all the records

Import Query Interface from **org.hibernate.query.Query**

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

List list=query.list();  **In hibernate 5 it became**

List<Customer> li = (List<Customer>)q.getResultList();

### Example of HQL to get records with pagination

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

query.setFirstResult(5);

query.setMaxResult(10);

List<Customer> li = (List<Customer>)q.getResultList();

### Example of HQL update query

Transaction tx=session.beginTransaction();

// Create object of Transaction before calling executeUpdate method

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> li = q.getResultList();

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");

# DEPRECATED FROM 5.2

# 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)

### Restrictions class

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

1. **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.

### Example of HCQL to get all the records

1. Crietria c=session.createCriteria(Emp.**class**);//passing Class class argument
2. List list=c.list();
3. Example of HCQL to get the 10th to 20th record
4. Crietria c=session.createCriteria(Emp.**class**);
5. c.setFirstResult(10);
6. c.setMaxResult(20);
7. 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();
4. 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();

Example of ProjectionList in Hibernate

**public** **static** **void** main(String[] args) {

Configuration c = **new** Configuration();

c.configure("Hibernate.cfg.xml");

SessionFactory sf = c.buildSessionFactory();

Session ss = sf.openSession();

Criteria cc = ss.createCriteria(Student.**class**);

ProjectionList p = Projections.*projectionList*();

p.add(Projections.*property*("sName"));

p.add(Projections.*property*("sid"));

cc.setProjection(p);

List<Student> li = cc.list();

Iterator it = li.iterator();

**while**(it.hasNext()) {

Object[] o = (Object[])it.next();

System.***out***.println(o[0] + " " + o[1]);

}

}

**NEW VERSION OF CRIETERIA START HERE**

From Hibernate 5.2 documentation - This appendix covers the legacy Hibernate org.hibernate.Criteria API, which should be considered **deprecate**

Most of the methods of the org.hibernate.Criteria API are deprecated and new development is focused on the JPA javax.persistence.criteria.CriteriaQuery API. In this post, I will show you how to use the JPA CriteriaQuery API for retrieving the entities based on specific criteria in Hibernate application.

# HCQL (Hibernate CriteriaQuery 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 retrieving 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.

### CriteriaQuery  interface

The CriteriaQueryinterface provides many methods to specify criteria. The object of CriteriaQuery can be obtained by calling the **createQuery ()** method of **CriteriaBuilder** interface.

The basic steps to create a Criteria query are –

a) Create a CriteriaBuilder instance by calling the Session.getCriteriaBuilder() method.

**CriteriaBuilder builder = session.getCriteriaBuilder();**

b) Create a query object by creating an instance of the CriteriaQuery interface.

**CriteriaQuery<T> query = builder.createQuery(T.class);**

**create** **table** student with sid,sName,sFee;

### Example of HCQL to get all the records

**select** \* **from** student;

**import** java.util.List;

**import** javax.persistence.criteria.CriteriaBuilder;

**import** javax.persistence.criteria.CriteriaQuery;

**import** javax.persistence.criteria.Root;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.Metadata;

**import** org.hibernate.boot.MetadataSources;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.query.Query;

**import** org.hibernate.service.ServiceRegistry;

**public** **class** FetchData {

**public** **static** **void** main(String[] args) {

ServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure().build();

MetadataSources mds = **new** MetadataSources(ssr);

Metadata md = mds.getMetadataBuilder().build();

SessionFactory sf = md.getSessionFactoryBuilder().build();

Session s = sf.openSession();

CriteriaBuilder cb = s.getCriteriaBuilder();

CriteriaQuery<Student> cq = cb.createQuery(Student.**class**);

Root<Student> rt = cq.from(Student.**class**);

cq.select(rt);

Query<Student> q = s.createQuery(cq);

List<Student> li = q.getResultList();

System.***out***.println(li);

}

}

**select** sName **from** student;

CriteriaQuery<String> cq = cb.createQuery(String.**class**);

Root<Student> rt = cq.from(Student.**class**);

cq.select(rt.get("sName"));

Query<String> q = s.createQuery(cq);

List<String> li = q.getResultList();

System.***out***.println(li);

**select** sName,sid **from** student;

CriteriaQuery<Object[]> cq = cb.createQuery(Object[].**class**);

Root<Student> rt = cq.from(Student.**class**);

cq.multiselect(rt.get("sName"),rt.get("sid"));

Query<Object[]> q = s.createQuery(cq);

List<Object[]> li = q.getResultList();

li.forEach(i -> System.***out***.println(Arrays.*toString*(i)));

**select** \* **from** student **where** sid = 1;

cq.select(rt).where(cb.equal(rt.get("sid"), 1));

**select** \* **from** student **where** sid > 1;

cq.select(rt).where(cb.gt(rt.get("sid"), 1));

Like this we have....

* 1. gt(Greater then)
  2. lt(Less then)
  3. ge(Greater then equal to)
  4. le(Less then equal to)
  5. notEqual
  6. between

cq.select(rt).where(cb.between(rt.get("sid"), 1,2));

**select** \* **from** student **where** sName like ‘%o%’;contain char o

cq.select(rt).where(cb.like(rt.get("sName"), "%o%")));

For start with "S%"

For Ends with "%y"

**select** \* **from** student **where** sid is null;

cq.select(rt).where(cb.isNull(rt.get("sid"))));

**select** \* **from** student **where** sid is not null;

cq.select(rt).where(cb.isNotNull(rt.get("sid"))));

To add two expressions with logical operations OR Operator:

**select** \* **from** student **where** sid = 1 OR sName = “Rahul”;

Root<Student> rt = cq.from(Student.**class**);

Predicate[] p = **new** Predicate[2];

p[0] = cb.equal(rt.get("sid"), 1);

p[1] = cb.equal(rt.get("sName"), "Pam");

cq.select(rt).where(cb.or(p));

To add two expressions with logical operations AND Operator:

**select** \* **from** student **where** sid = 1 AND sName = “Rahul”;

Root<Student> rt = cq.from(Student.**class**);

Predicate[] p = **new** Predicate[2];

p[0] = cb.equal(rt.get("sid"), 1);

p[1] = cb.equal(rt.get("sName"), "Pam");

cq.select(rt).where(cb.and(p));

Combine two conditions with and, or operator

**select \* from** student **where (qid = 1 or qid = 2) and (qid = 1 and qname = 'A');**

Predicate[] p1 = **new** Predicate[2];

p1[0] = cb.equal(rt.get("qid"),1);

p1[1] = cb.equal(rt.get("qid"),2);

Predicate[] p2 = **new** Predicate[2];

p2[0] = cb.equal(rt.get("qid"),1);

p2[1] = cb.equal(rt.get("qname"),"A");

cr.select(rt).where(cb.and(cb.or(p1),cb.and(p2)));

**Sorting orderBy in Asc or Desc**

Root<Student> rt = cq.from(Student.**class**);

cq.orderBy(cb.desc(rt.get("sid")));

cq.select(rt);

**Update using CriteriaBuilder**

**update** student **set** sName = “Sam” **where** qid = 1;

CriteriaUpdate<Item> criteriaUpdate = cb.createCriteriaUpdate(Student.**class**);

Root<Item> root = criteriaUpdate.from(Student.**class**);

criteriaUpdate.set("itemPrice", newPrice);

criteriaUpdate.where(cb.equal(root.get("itemPrice"), oldPrice));

Transaction transaction = session.beginTransaction();

session.createQuery(criteriaUpdate).executeUpdate();

transaction.commit();

**delete from** student **where** qid = 1;

CriteriaDelete<Item> criteriaDelete = cb.createCriteriaDelete(Student.**class**);

Root<Item> root = criteriaDelete.from(Student.**class**);

criteriaDelete.where(cb.equal(root.get("itemPrice"), oldPrice));

Transaction transaction = session.beginTransaction();

session.createQuery(criteriaDelete).executeUpdate();

transaction.commit();

Aggregate functions in Hibernate

1 ) Get row count:

CriteriaQuery<Long> cq = cb.createQuery(Long.**class**);

Root<Student> rt = cq.from(Student.**class**);

cq.multiselect(cb.count(rt.get("sid")));

Query<Long> q = s.createQuery(cq);

List<Long> li = q.getResultList();

System.***out***.println(li);

2 ) Get Average :

CriteriaQuery<Long> cq = cb.createQuery(Double.**class**);

Root<Student> rt = cq.from(Student.**class**);

cq.multiselect(cb.avg(rt.get("sid")));

Query<Double> q = s.createQuery(cq);

List<Double> li = q.getResultList();

System.***out***.println(li);

Other useful aggregate methods that are available are ***sum()*,** ***max()***, ***min()*, *count()*** etc.

Example of HCQL to get the 1st to 2nd record Paginations

Query<Student> q = s.createQuery(cq);

q.setFirstResult(1);

q.setMaxResults(2);

List<Student> li = q.getResultList();