# Hibernate Criteria Query Language (HCQL)

Hibernate Criteria Query Language (HCQL) is mainly used for searching and fetching records. It works on the filtration rules and logical conditions. The Criteria interface, Restriction class, and Order class are available in **org.hibernate** package provides the properties and methods to perform operations of HCQL.

The object of Criteria interface can be obtained by calling the method **createCriteria()** provided by the Session interface. In HCQL, we can only execute the **select** query. It doesn't allow to use other queries like an update, insert, and delete.

**Syntax of the Criteria query**

```
Criteria criteria =
session.createCriteria(persistantClassName.class);
```

## Advantages of Hibernate Criteria Query Language (HCQL)

- **Dynamic Queries-** HCQL queries are suitable for executing dynamic queries, whereas HQL does not support dynamic queries. It only supports static queries.
- **Pagination –** Hibernate criteria query language (HCQL) supports the concept of pagination. Pagination is a technique which divides the fetching of a result set containing several pages into a small number of pages.
- **Database Independent-** HCQL is also a database-independent query language. That means if we write programs using HCQL command, the program can be executed in all relational databases without any modification.
- **Support Projection-** The projection class is available in **org.hibernate.criterion.Projection** package which can be used to get minimum, maximum, sum, count, and average of the property values.

## Criteria Interface

The Criteria interface is made available in **org.hibernate** package which is used to retrieve entities by creating criterion objects. It is used for searching objects or data as per the given conditions.

There are many methods available in the Criteria interface to specify the criteria. Following are some methods of the Criteria interface:

1. **public Criteria add(Criteria c)**

It is used to add restrictions on the result.

- **public Criteria addOrder(Order o)**

It is used for defining the order of the result such as ascending and descending.

- **public Criteria setFirstResult(int firstResult)**

It is used to define the first number of the record to be fetched.

- **public Criteria setMaxResult(int totalResult)**

It is used to determine the total number of record to be retrieved.

- **public List list()**

It returns the list of the current searched objects.

- **public Criteria setProjection(Projection projection)**

It is used to apply projection to retrieve objects.

- **public Criteria createAlias(String associationPath, String alias)**

It is used to join the association or assigning the alias to the joined association.

## Order Class

Order class helps in sorting the records of the database table. By using order class, we can sort the records in ascending or descending order.
Following are the methods of Order class:

- **public static Order asc(String propertyName)**

It is used to sort the data in ascending order.
**Example of Order.asc**

```
Crietria criteria=session.createCriteria(Student.class);

criteria.addOrder(Order.asc("name"));

List list=criteria.list();
```

- **public static Order desc(String propertyName)**

It is used to sort the data in descending order.
**Example of Order.desc**

```
Crietria criteria=session.createCriteria(Student.class);

criteria.addOrder(Order.desc("name"));

List list=criteria.list();
```

- **public Order ignoreCase()**

It is used to ignore the space, upper, and lower case in the record, i.e., it case-insensitive.

## Restrictions Class

Restrictions class provides many methods that can be used to add restrictions to the criteria object. There are many methods available in **org.hibernate.criterion.Restrictions** package.
Following are the commonly used methods of Restrictions class:

1. **public static SimpleExpression eq(String propertyName, Object value)**

Here ‘eq’ stands for **equal**. It is used to apply an ‘equal’ constraint to the given property.

**Example of Restrictions.eq**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.eq("rollNum", 16));

List list = criteria.list();
```

- **public static SimpleExpression ge(String propertyName, Object value)**

Here ‘ge’ stands for **greater than or equal.** It is used toapply ‘greater than or equal’ constraint to the given property.

**Example of Restrictions.ge**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.ge("rollNum", 10));

List list = criteria.list();
```

- **public static SimpleExpression gt(String propertyName, Object value)**

Here ‘gt’ stands for **greater than**. It is used to apply ‘greater than’ constraint to the given property.

**Example of Restrictions.gt**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.gt("rollNum", 10));

List list = criteria.list();
```

- **public static SimpleExpression le(String propertyName, Object value)**

Here ‘le’ stands for **less than or equal.** It is used to apply ‘less than or equal’ constraint to the given property.

Example of Restrictions.le

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.le("rollNum", 16));

List list = criteria.list();
```

- **public static SimpleExpression like(String propertyName, Object value)**

It is used to apply 'like' constraint to the given property.

**Example of Restrictions.like**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.like("sname", "%SHARMA%"));

List list = criteria.list();
```

- **public static SimpleExpression lt(String propertyName, Object value)**

Here 'lt' stands for **less than.** It is used to apply 'less than' constraint to the given property.

**Example of Restrictions.lt**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.lt("rollNum", 10));

List list = criteria.list();
```

- **public static SimpleExpression ne(String propertyName, Object value)**

Here ‘ne’ stands for **not equal.** It is used to apply ‘not equal’ constraint to the given property.

**Example of Restrictions.ne**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.ne("rollNum", 7));

List list = criteria.list();
```

- **public static Criterion between(String propertyName, Object lo, Object hi)**

It is used to apply ‘between’ constraint between the lo (low) and hi (high) object values of the given property.

**Example of Restrictions.between**

```
Criteria criteria = session.createCriteria(Student.class);

criteria.add(Restrictions.between("rollNum", 10,16));

List list = criteria.list();
```

## Hibernate Named Query using Annotation

In annotation-based named query, we use **@NamedQuery** annotation inside the POJO class.

**@NamedQuery**- It is used to describe a single named query. It consists of four attributes- two of which are compulsory, and two are optional. The two required attributes are **name** and **query** which defines the query name and the query string itself. The two optional attributes are **lockMode** and **hints** which provide the replacement for setLockMode() and setHint() methods.