Hibernate – VI

The Hibernate Query Language (HQL)

Table of Contents

1. Introduction
2. **Org.hibernate.Query** Interface
3. **From** clause (Fetching Complete Data)
4. **Select** clause (Fetching Partial Data)
5. **Where** clause (Filtering the Records)
6. Creating Dynamic Query (**Passing Runtime Parameters and Values**)
7. **Order By** clause (Sorting the Records)
8. **Group By** clause (Grouping the Records)
9. **Having** clause
10. **Insert, update and Delete** Operations
11. Aggregate Functions (**avg(), sum(), min(), max(), count()**)
12. Associations and Jions (**Inner Join, Left Outer Join, Right Outer Join, Full Join**)
13. Creating Subqueries
14. **Criteria**
15. **Criterion**
16. **Restrictions**
17. **Order**
18. **Projection**
19. **Projections**

Introduction

1. In Hibernate, we can perform the operations(save, update and delete) only on single object at a time. So to perform the bulk operations, hibernate provides a powerful query language that is Hibernate Query Language (HQL). It is similar in appearance to SQL.
2. We can say, HQL is an Object-Oriented form of SQL.
3. In HQL, there is no need to use Table-Name and Column Names. We can use Entity as Table-Name and Property Name as column name.

Advantages of HQL:

1. HQL is database independent
2. HQL supports object oriented features like Inheritance, polymorphism, Associations
3. We can perform DML operations (insert, update, delete)
4. One great feature is we can update primary key also by using Hibernate Query (HQL).

NOTE:

HQL is not case-sensitive, SeLeCt, SELECT both are same as select but in case of Entity and property name- HQL is case-sensitive.

com.kalibermind.hibernate.entity.Customer is not same as com.kalibermind.hibernate.entity.CUSTOMER

Query Interface

Query Interface provides an object-oriented representation of a Hibernate query. A Query instance is obtained by calling *createQuery()* method of *Session*. *Session* is a factory of Query.

```
Session session = factory.openSession();
String hql = "from Customer as cust";
Query query = session.createQuery(hql);
```

Mostly Used Methods

Methods	Description
List list()	Return the query results as a List.
Query setParameter(int position, Object val)	Bind a value to a JDBC-style query parameter.
Query setParameter(String name, Object val)	Bind a value to a named query parameter.
int executeUpdate()	Execute the update or delete statement.

FROM clause (Fetching Complete Data)

All possible forms of From clause-

1. from Customer
2. from Customer as cust
3. from Customer cust
4. from Customer, Order(Cartesian Product)

Example:

Customer.java

```
package com.Biditvats.domain;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "CUSTOMER_MASTER11")
public class Customer {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "CUSTOMER_ID")
	private Long id;
	@Column(name = "FIRST_NAME")
	private String firstName;
	@Column(name = "LAST_NAME")
	private String lastName;
	@Column(name = "EMAIL", unique = true)
	private String email;
	@Column(name = "MOBILE")
	private String mobile;
	@OneToMany(cascade = CascadeType.ALL)
	@JoinColumn(name = "CUSTOMER_ID")
	private List<Order> orders;

	public Customer() {
		// Do Nothing
	}

	public Customer(String firstName, String lastName, String email, String mobile) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
		this.mobile = mobile;
	}

	// Getters and Setters
}
```

Order.java

```
ckage com.Biditvats.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "ORDER_MASTER11")
public class Order {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "ORDER_ID")
	private Long id;
	@Column(name = "ITEM_NAME")
	private String itemName;
	@Column(name = "MODEL")
	private String model;
	@Column(name = "BRAND")
	private String brand;
	@Column(name = "PRICE")
	private Double price;
	@Column(name = "QUANTITY")
	private Integer quantity;

	@ManyToOne
	@JoinColumn(name="CUSTOMER_ID")
	private Customer customer;

	public Order() {
		// Do Nothing
	}
	public Order(String itemName, String model, String brand, Double price, Integer quantity) {
		super();
		this.itemName = itemName;
		this.model = model;
		this.brand = brand;
		this.price = price;
		this.quantity = quantity;
	}
    //Getters and Setters
}
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
	<session-factory>
		<!-- Data Source Details -->
		<property name="hibernate.connection.driver_class">
			com.mysql.jdbc.Driver
		</property>
		<property name="hibernate.connection.url">
			jdbc:mysql://localhost:3306/hibernatedb
		</property>
		<property name="hibernate.connection.username">root</property>
		<property name="hibernate.connection.password">password</property>
```

```
		<!-- Hibernate Properties -->
		<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
		<property name="hibernate.hbm2ddl.auto">update</property>
		<property name="hibernate.show_sql">true</property>
		<property name="hibernate.format_sql">true</property>

		<!-- Resource Mapping -->
		<mapping class="com.Biditvats.domain.Customer"/>
		<mapping class="com.Biditvats.domain.Order"/>

	</session-factory>
</hibernate-configuration>
```

SaveData.java

```
package com.Biditvats.test;

import java.util.ArrayList;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.Biditvats.domain.Customer;
import com.Biditvats.domain.Order;


public class SaveCustomer {

	public static void main(String[] args) {
		Configuration configuration = new Configuration().configure();
		SessionFactory sessionFactory = configuration.buildSessionFactory();

		List<Order> orders = new ArrayList<>();
		orders.add(new Order("Mobile", "Iphone6", "Apple", 25000.00, 1));
		orders.add(new Order("Mobile", "4A", "Redmi", 7000.00, 4));
		orders.add(new Order("Laptop", "G50", "Lenovo", 35000.00, 1));

		Customer customer = new Customer("Bidit", "Vats", "Biditvats@gmail.com", "9916712669");
		customer.setOrders(orders);

		Session session = sessionFactory.openSession();
		Transaction transaction = session.beginTransaction();
		session.save(customer);
		transaction.commit();
		session.close();

		System.out.println("Record has been saved successfully!");

	}

}
```

FetchData.java

```
package com.Biditvats.test;

import java.util.*;
import org.hibernate.*;
```

```
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.Biditvats.domain.Customer;
import com.Biditvats.domain.Order;


public class FetchData {

	public static void main(String[] args) {
		Configuration configuration = new Configuration().configure();
		SessionFactory sessionFactory = configuration.buildSessionFactory();
	int id =0;
	System.out.println("Enter Customer ID:");
	Scanner s = new Scanner(System.in);
	s.nextInt();
	s.close();

	Session session = sessionFactory.openSession();

	//Fetching all object with all properties
	String hql = "from Customer as cust";
	Query query = session.createQuery(hql);
	List list = query.list();
	Iterator itr = list.iterator();

	while(itr.hasNext()) {
	      Customer cust = (Customer)itr.next();
	       System.out.println("Customer ID: " +cust.getCustId());
	       System.out.println("Customer Name: " +cust.getCustName());
	       System.out.println("Email: " +cust.getEmail());
	       System.out.println("Mobile: " +cust.getMobile());
	       System.out.println();
	 }
}
```

Select clause (Fetching Partial Data)

```
//Fetching all object with partial properties using select clause
String hql = "select ord.orderId, ord.itemName from Order as ord  ";
Query query = session.createQuery(hql);
List list = query.list();
Iterator itr = list.iterator();
      while(itr.hasNext()) {
	Object[] o = (Object[])itr.next();
	System.out.println("Order ID: "+o[0]);
	System.out.println("Item Name: "+o[1]);
	System.out.println();
      }
```

Where clause (Fetching Partial Data)

The where clause allows you to refine the list of instances returned. If no alias exists, you can refer to properties by name:

from Customer where custName='CP Verma'
from Customer as cust where cust.custName='CP Verma'

```
//Using where clause
String hql = "select ord.orderId, ord.itemName from Order as ord where ord.customer.custId=2";
Query query = session.createQuery(hql);
List list = query.list();
Iterator itr = list.iterator();
    while(itr.hasNext()) {
       Object[] o = (Object[])itr.next();
       System.out.println("Order ID: "+o[0]);
       System.out.println("Item Name: "+o[1]);
       System.out.println();
    }
```

Creating Dynamic Query (Passing Runtime Parameters and Values)

```
//Passing Runtime Values
String hql = "select ord.orderId, ord.itemName from Order as ord where ord.customer.custId=:ci ";
Query query = session.createQuery(hql);
query.setParameter("ci", id);
List list = query.list();
Iterator itr = list.iterator();
    while(itr.hasNext()) {
       Object[] o = (Object[])itr.next();
       System.out.println("Order ID: "+o[0]);
       System.out.println("Item Name: "+o[1]);
       System.out.println();
    }
String hql = "select ord.itemName from Order as ord where ord.customer.custId=? ";
Query query = session.createQuery(hql);
query.setParameter(1, id);
```

Order By clause (Sorting the Records) asc, desc

```
//Order By Clause
 String hql = "from Customer as cust order by cust.custId desc ";
Query query = session.createQuery(hql);
 List list = query.list();
 Iterator itr = list.iterator();
    while(itr.hasNext()) {
       Customer cust= (Customer)itr.next();
       System.out.println("Customer ID: "+cust.getCustId());
       System.out.println("Customer Name: "+ cust.getCustName());
       System.out.println("Email: "+ cust.getEmail()););
       System.out.println("Mobile: "+ cust.getMobile()););
       System.out.println();
    }
```

Group By clause (Grouping the Records)

```
//Group By Clause
 String hql = "from Order as ord group by ord.itemName ";
Query query = session.createQuery(hql);
```

```
List list = query.list();
Iterator itr = list.iterator();
    while(itr.hasNext()) {
       Order order= (Order)itr.next();
       System.out.println("Order ID: "+order.getCustId());
       System.out.println("Item Name: "+ order.getItemName());
       System.out.println("Brand: "+ order.getBrand()););
       System.out.println("Price: "+ order.getPrice()););
       System.out.println();
    }
```

Having clause

```
//having Clause
String hql = "select ord.itemName from Order ord group by ord.brand having ord.brand in('Lenovo')";
Query query = session.createQuery(hql);
List list = query.list();
Iterator itr = list.iterator();
    while(itr.hasNext()) {
       System.out.println("Item Name: "+itr.next());
    }
```

Insert, Update and Delete Operations

- ✓ In insert query there is one limitation i.e. we can't insert our own values manually.
- ✓ Here we will take example to transfer object from one table to another or we can say dump data from one table to another.
- ✓ Here same as previous application all the file same, just we take **NewOrder.java** and we map it in New **newOrder.hbm.xml** file.

Note : All Queries required Transaction

```
String hql = "insert into NewOrder(orderId,productName,brand,price)
                select o.orderId,o.productName,o.brand,o.price from Order o";
Query query = session.createQuery(hql);
int i = query.executeUpdate( );

//String hql = "update Order set brand='SAMSUNG', price= 27000.00 where orderId=2";
String hql = "update Order set orderId=222  where orderId=2";  //Update PK
Query query = session.createQuery(hql);
int i = query.executeUpdate( );

String hql = "Delete Order where id=222";
Query query = session.createQuery(hql);
int i = query.executeUpdate( );
```

Aggregate Functions - avg(), sum(), min(), max(), count()

```
//Using Aggregate Functions
//String hql = "select min(ord.price) from Order as ord";
//String hql = "select max(ord.price) from Order as ord";
//String hql = "select count(ord.price) from Order as ord";
```

```
String hql = "select avg(ord.price) from Order as ord";
Query query = session.createQuery(hql);
 List list = query.list();
 Iterator itr = list.iterator();
      while(itr.hasNext()) {
         Double price= (Double)itr.next();
         System.out.println("Average of Price: "+price);
      }
```

Associations and joins

```
//Join of two Tables
String hql = "from Order as ord where ord.customer.custName like '%n%'";
Query query = session.createQuery(hql);
List list = query.list();
Iterator itr = list.iterator();
while(itr.hasNext()) {
   Order order= (Order)itr.next();
   System.out.println("Order ID: "+order.getCustId());
   System.out.println("Item Name: "+ order.getItemName());
   System.out.println("Brand: "+ order.getBrand()););
   System.out.println("Price: "+ order.getPrice()););
   System.out.println();
}
//Left join
String hql = “select cust.custName, ord.itemName from Order as ord left join ord.customer cust”;

//Right join
String hql = “select cust.custName, ord.itemName from Order as ord right join ord.customer cust”;
Query query = session.createQuery(hql);
 List list = query.list();
 Iterator itr = list.iterator();
    while(itr.hasNext()) {
       Object[ ] o= (Object[])itr.next();
       System.out.println("Customer Name: "+ o[0]);
       System.out.println("Item Name: "+ o[1]);
    }
```

Sub Queries

```
//Sub Queries
String hql = "from Order as ord where ord.price > (select avg(ord1.price) from Order as ord1)";
Query query = session.createQuery(hql);
List list = query.list();
Iterator itr = list.iterator();
      while(itr.hasNext()) {
         Order order= (Order)itr.next();
         System.out.println("Order ID: "+order.getCustId());
         System.out.println("Item Name: "+ order.getItemName());
```

```
        System.out.println("Brand: "+ order.getBrand()););
        System.out.println("Price: "+ order.getPrice()););
        System.out.println();
    }
```

Criteria Queries

- ✓ A criterion is an Interface belongs to **org.hibernate.criterion package**.
- ✓ The interface **org.hibernate.Criteria** represents a query against a particular persistent class.
- ✓ The Session is a factory for Criteria instances.
- ✓ Criteria is only for selecting the data from the database,
- ✓ we can select complete objects as well as partial objects also,
- ✓ Using projections concept we can select partial objects
- ✓ A criterion is suitable for executing dynamic queries too.

```
Criteria crit = session.createCriteria (Product.class);
List<Product> results = crit.list( );
```

Using Restrictions with Criteria

- ✓ The Criteria API makes it easy to use restrictions in our queries to selectively retrieve objects
- ✓ We can make any restrictions to a Criteria object with the add () method. For e.g.: "**greater than**", "**less than**", "**equals to**" ,"**Not equals to**" .....
- ✓ The add() method takes an **org.hibernate.criterion.Criterion** object that represents an individual restriction.
- ✓ We can create more than one restriction for a criteria query.

(1)Restrictions.eq ( ) Example

To retrieve objects that have a property value that "equals" your restriction, use the **eq()** method on **Restrictions**, as follows:

```
Criteria crit = session.createCriteria (Orders.class);
Crit.add (Restrictions.eq ("productName","Apple"));
List<Product> results = crit.list();
```

Above query will search all **productName** having name as **"Apple".**

(2)Restrictions.ne ( ) Example

To retrieve objects that have a property value that "not equals to" your restriction, use the **ne()** method on **Restrictions**, as follows:

```
Criteria crit = session.createCriteria (Orders.class);
crit.add (Restrictions.ne ("productName","Apple"));
List<Product> results = crit.list();
```

Above query will search all **productName** having not name as **"Apple".**

(3)Restrictions.like ( ) and Restrictions.ilike ()

Instead of searching for exact matches, we can retrieve all objects that have a property matching part of a given pattern. To do this, we can create an SQL LIKE clause, with either the **like()** or the **ilike()** method.

The **ilike()** method is case-insensitive.

```
Criteria crit = session.createCriteria (Orders.class);
```

crit.add (Restrictions.like (“productName”,”Ipho%”,MatchMode.ANYWHERE));
List<Product> results = crit.list();

Above example **MatchMode** object to specify how to match the specified value to the stored data.

The **MatchMode** object has four different matches:

- ✓ **ANYWHERE** : Anyplace in the string
- ✓ **END** : The end of the string
- ✓ **EXACT** : An exact match
- ✓ **START** : The beginning of the string

(4)Restrictions.isNull ( ) and Restrictions.isNotNull ( )

The isNull( ) and isNotNull ( ) restrictions allow you to do a search for objects that have or do not null

Criteria crit = session.createCriteria (Orders.class);
crit.add (Restrictions.isNull (“productName”));
List<Product> results = crit.list();

(5)Restrictions.gt ( ) , Restrictions.ge ( ), Restrictions.lt ( ) and Restrictions.le ( )

Several restrictions we can use like … The **greater-than comparison is gt( )**, the **greater-than-or-equal-to comparison is ge( )**, the **less-than comparison is lt( )**, and the **less-than-or-equal-to comparison is le( )**.

For e.g. we can do a quick retrieval of all products with prices over **88000.00 like this**

Criteria crit = session.createCriteria (Orders.class);
crit.add (Restrictions.gt (“price”, 88000.00));
List<Product> results = crit.list();

(6)Combining Two or More Criteria

- ✓ We can combine **AND** and **OR** restrictions in logical expressions.
- ✓ If we want to have two restrictions that return objects that satisfy either or both of the restrictions, we need to use the **or ( )** method with the Restrictions class.

```
Criteria crit = session.createCriteria (Orders.class);
Criterion priceLessThan = Restrictions.lt(“price”,88000.0);
Criterion productLike = Restrictions.ilike(“productName”, “Iphone %”);
LogicalExpression orExp = Restrictions.or (priceLessThan, productLike);
crit.add (orExp);
List results = crit.list();
```

(7)Using Disjunction Objects with Criteria

- ✓ If we wanted to create or an OR expression with two or more different criteria

(For example, “**price > 88000.0** OR **productName like Iphone%** OR **brand not like Apple %**”)

- ✓ We can obtain this from the **disjunction ( ) factory** method on the Restrictions class.
- ✓ Disjunction is more convenient than building a tree of OR expressions in code.
- ✓ To represent AND expression with more than two criteria, we can use the **conjuction( )** method
- ✓ The conjuction can be more convenient than building a tree of AND expressions in code.

Let's take an example that uses the disjunction:

```
Criteria crit = session.createCriteria (Orders.class);
Criterion priceLessThan = Restrictions.lt("price",88000.0);
Criterion name = Restrictions.ilike("productName", "Iphone");
Criterion productBrand = Restrictions.ilike("brand", "Apple");
Disjunction disjunction = Restrictions.disjunction ();
disjunction.add (priceLessThan);
disjunction.add (name);
disjunction.add (productBrand);
crit.add (disjunction);
List results = crit.list();
```

(8)Obtaining a Unique Result

- ✓ Sometimes we required only zero or one object from a given query.
- ✓ If we want to obtain a single Object reference instead of a List,
- ✓ We use **uniqueResult( )** method on the Criteria object returns an object or null.Otherwise,throw a NonUniqueResultException exception.

```
Criteria crit = session.createCriteria (Orders.class);
Criterion price = Restrictions.gt("price",new Double(88000.0));
crit.setMaxResults(1);
Orders orders = (Orders) crit.uniqueResult ();
```

A result set that would have included more than one result, except that it was limited with the **setMaxResults ()** method.

(9)Obtaining Distinct Result

- ✓ If we want to work with distinct results from a criteria query,
- ✓ Hibernate provides a result transformer for distinct entitie,
- ✓ org.hibernate.transform.DistinctRootEntityResultTransformer, which ensures that no duplicates will be in our query's result set. Instead of using SELECT DISTINCT with SQL.

```
Criteria crit = session.createCriteria (Orders.class);
Criterion price = Restrictions.gt("price",new Double(88000.0));
crit.setResultTransformer (DistinctRootEntityResultTransformer.INSTANCE)
List<Orders> results = crit.list ();
```

(10)Sorting the Query's Results

- ✓ Sorting the query's results works much the same way with criteria as it is with HQL or SQL.
- ✓ The Criteria API provides the org.hibernate.criterion.Order class to sort your result set in either ascending or descending order.

For **E.g.** Order class:

```
Criteria crit = session.createCriteria (Orders.class);
```

```
crit.add (Restrictions.gt("price",88000.0));
crit.addOrder (Order.desc ("price"));
List<Orders> results = crit.list ();
```

We can add more than one Order object to the Criteria object.

Adding Projections and Aggregates

- ✓ Instead of working with objects from the result set, you can treat the results from the result set as a set of rows and columns, also known as a projection of the data.
- ✓ The class **org.hibernate.criterion.Projections** is a factory for Projection instances. We can apply a projection to a query by calling **setProjection()**.
- ✓ To use projections, First we get the **org.hibernate.criterion.Projection** object
- ✓ The Projections class is similar to the Restrictions class first we get the Projection instances.
- ✓ After getting a Projection object, add it to Criteria object with the **setProjection()** method.

Single Aggregate (Getting Row Count)

```
Criteria crit = session.createCriteria (Orders.class);
crit.setProjection (Projections.rowCount ());
List<Long> results = crit.list();
```

Other aggregate functions are available as following:

- ✓ **avg(String propertyName)** :Gives the average of a property's value
- ✓ **count(String propertyName)** :Counts the number of times a property occurs
- ✓ **countDistinct(String propertyName)** :Counts the number of unique values the property contains.
- ✓ **max(String propertyName)** :Calculates the maximum value of the property values
- ✓ **min(String propertyName)** :Calculates the minimum value of the property values
- ✓ **sum(String propertyName)** :Calculates the sum total of the property values.

Multiple Aggregates

- ✓ We can apply more than one projection to a given Criteria object.
- ✓ To add multiple projections, get a projection list from the **projectionList ( )** method on the Projections class.
- ✓ The **org.hibernate.criterion.ProjectionList** object has an add ( ) method that takes a Projection object.
- ✓ We can pass the projections list to the **setProjection ( )** method on the Criteria object because ProjectionList implements the Projection interface.

```
Criteria crit = session.createCriteria (Orders.class);

ProjectionList projList = Projections.projectionList();

projList.add (Projections.max ("price"));

projList.add (Projections.min ("price"));

projList.add (Projections.avg ("price"));

projList.add (Projections.countDistinct("productName"));

crit.setProjection (projList);

List<object[]> results = crit.list ();
```

Selected Columns

Another use of projections is to retrieve individual properties, rather than entities. For instance, we can retrieve just the productName and brand from our orders table, instead of loading the entire object representation into memory.

```
Criteria crit = session.createCriteria (Orders.class);

ProjectionList projList = Projections.projectionList ();

projList.add (Projections.property (“productName”));

projList.add (Projections.property (“brand”));

crit.setProjection (projList);

crit.addOrder (Order.asc (“price”));

List<object []> results = crit.list ();
```