Hibernate - Criteria Queries

Hibernate provides alternate ways of manipulating objects and in turn data available in RDBMS tables. One of the methods is Criteria API, which allows you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.

The Hibernate **Session** interface provides **createCriteria()** method, which can be used to create a **Criteria** object that returns instances of the persistence object's class when your application executes a criteria query.

Following is the simplest example of a criteria query is one, which will simply return every object that corresponds to the Employee class.

```
Criteria cr = session.createCriteria(Employee.class);
List results = cr.list();
```

Restrictions with Criteria

You can use **add()** method available for **Criteria** object to add restriction for a criteria query. Following is the example to add a restriction to return the records with salary is equal to 2000 −

```
Criteria cr = session.createCriteria(Employee.class);
cr.add(Restrictions.eq("salary", 2000));
List results = cr.list();
```

Following are the few more examples covering different scenarios and can be used as per the requirement −

```
Criteria cr = session.createCriteria(Employee.class);

// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To get records having salary less than 2000
cr.add(Restrictions.lt("salary", 2000));
```

```
// To get records having fistName starting with zara
cr.add(Restrictions.like("firstName", "zara%"));

// Case sensitive form of the above restriction.
cr.add(Restrictions.ilike("firstName", "zara%"));

// To get records having salary in between 1000 and 2000
cr.add(Restrictions.between("salary", 1000, 2000));

// To check if the given property is null
cr.add(Restrictions.isNull("salary"));

// To check if the given property is not null
cr.add(Restrictions.isNotNull("salary"));

// To check if the given property is empty
cr.add(Restrictions.isEmpty("salary"));

// To check if the given property is not empty
cr.add(Restrictions.isNotEmpty("salary"));
```

You can create AND or OR conditions using LogicalExpression restrictions as follows −

```
Criteria cr = session.createCriteria(Employee.class);

Criterion salary = Restrictions.gt("salary", 2000);
Criterion name = Restrictions.ilike("firstNname","zara%");

// To get records matching with OR conditions
LogicalExpression orExp = Restrictions.or(salary, name);
cr.add( orExp );

// To get records matching with AND conditions
LogicalExpression andExp = Restrictions.and(salary, name);
cr.add( andExp );

List results = cr.list();
```

Though all the above conditions can be used directly with HQL as explained in previous tutorial.

Pagination Using Criteria

There are two methods of the Criteria interface for pagination.

Sr.No.	Method & Description
1	**public Criteria setFirstResult(int firstResult)** This method takes an integer that represents the first row in your result set, starting with row 0.
2	**public Criteria setMaxResults(int maxResults)**

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 −

```
Criteria cr = session.createCriteria(Employee.class);
cr.setFirstResult(1);
cr.setMaxResults(10);
List results = cr.list();
```

Sorting the Results

The Criteria API provides the **org.hibernate.criterion.Order** class to sort your result set in either ascending or descending order, according to one of your object's properties. This example demonstrates how you would use the Order class to sort the result set −

```
Criteria cr = session.createCriteria(Employee.class);

// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To sort records in descening order
cr.addOrder(Order.desc("salary"));

// To sort records in ascending order
cr.addOrder(Order.asc("salary"));

List results = cr.list();
```

Projections & Aggregations

The Criteria API provides the **org.hibernate.criterion.Projections** class, which can be used to get average, maximum, or minimum of the property values. The Projections class is similar to the Restrictions class, in that it provides several static factory methods for obtaining **Projection** instances.

Following are the few examples covering different scenarios and can be used as per requirement −

```
Criteria cr = session.createCriteria(Employee.class);

// To get total row count.
cr.setProjection(Projections.rowCount());

// To get average of a property.
cr.setProjection(Projections.avg("salary"));

// To get distinct count of a property.
cr.setProjection(Projections.countDistinct("firstName"));
```

```
// To get maximum of a property.
cr.setProjection(Projections.max("salary"));

// To get minimum of a property.
cr.setProjection(Projections.min("salary"));

// To get sum of a property.
cr.setProjection(Projections.sum("salary"));
```

Criteria Queries Example

Consider the following POJO class −

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}

   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }

   public int getId() {
      return id;
   }

   public void setId( int id ) {
      this.id = id;
   }

   public String getFirstName() {
      return firstName;
   }

   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }

   public String getLastName() {
      return lastName;
   }

   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }

   public int getSalary() {
      return salary;
   }

   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

Let us create the following EMPLOYEE table to store Employee objects −

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Following will be the mapping file.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name = "Employee" table = "EMPLOYEE">

      <meta attribute = "class-description">
         This class contains the employee detail.
      </meta>

      <id name = "id" type = "int" column = "id">
         <generator class="native"/>
      </id>

      <property name = "firstName" column = "first_name" type = "string"/>
      <property name = "lastName" column = "last_name" type = "string"/>
      <property name = "salary" column = "salary" type = "int"/>

   </class>
</hibernate-mapping>
```

Finally, we will create our application class with the main() method to run the application where we will use **Criteria** queries −

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.Criteria;
import org.hibernate.criterion.Restrictions;
import org.hibernate.criterion.Projections;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try {
         factory = new Configuration().configure().buildSessionFactory();
      } catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }

      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 2000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 5000);
      Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

      /* List down all the employees */
      ME.listEmployees();

      /* Print Total employee's count */
      ME.countEmployee();

      /* Print Toatl salary */
      ME.totalSalary();
   }

   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salary){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;

      try {
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
      return employeeID;
   }

   /* Method to  READ all the employees having salary more than 2000 */
   public void listEmployees( ) {
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);
         // Add restriction.
         cr.add(Restrictions.gt("salary", 2000));
         List employees = cr.list();

         for (Iterator iterator = employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("  Salary: " + employee.getSalary());
         }
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }

   /* Method to print total number of records */
   public void countEmployee(){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);

         // To get total row count.
         cr.setProjection(Projections.rowCount());
         List rowCount = cr.list();

         System.out.println("Total Coint: " + rowCount.get(0) );
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }

   /* Method to print sum of salaries */
   public void totalSalary(){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);

         // To get total salary.
         cr.setProjection(Projections.sum("salary"));
         List totalSalary = cr.list();

         System.out.println("Total Salary: " + totalSalary.get(0) );
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }
}
```

Compilation and Execution

Here are the steps to compile and run the above mentioned application. Make sure, you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

Create hibernate.cfg.xml configuration file as explained in configuration chapter.

Create Employee.hbm.xml mapping file as shown above.

Create Employee.java source file as shown above and compile it.

Create ManageEmployee.java source file as shown above and compile it.

Execute ManageEmployee binary to run the program.

You would get the following result, and records would be created in the EMPLOYEE table.

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 5000
First Name: Mohd  Last Name: Yasee  Salary: 3000
Total Coint: 4
Total Salary: 15000
```

If you check your EMPLOYEE table, it should have the following records–

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 14 | Zara       | Ali       |   2000 |
| 15 | Daisy      | Das       |   5000 |
| 16 | John       | Paul      |   5000 |
| 17 | Mohd       | Yasee     |   3000 |
+----+------------+-----------+--------+
4 rows in set (0.00 sec)
mysql>
```