JPA - CRITERIA API

The Criteria API is a predefined API used to define queries for entities. It is the alternative way of defining a JPQL query. These queries are type-safe, and portable and easy to modify by changing the syntax. Similar to JPQL it follows abstract schema *easytoeditschema* and embedded objects. The metadata API is mingled with criteria API to model persistent entity for criteria queries.

The major advantage of the criteria API is that errors can be detected earlier during compile time. String based JPQL queries and JPA criteria based queries are same in performance and efficiency.

History of criteria API

The criteria API is included into all versions of JPA therefore each step of criteria API is notified in the specifications of JPA.

- In JPA 2.0, the criteria query API, standardization of queries are developed.
- In JPA 2.1, Criteria update and delete *bulkupdateanddelete* are included.

Criteria Query Structure

The Criteria API and the JPQL are closely related and are allowed to design using similar operators in their queries. It follows javax.persistence.criteria package to design a query. The query structure means the syntax criteria query.

The following simple criteria query returns all instances of the entity class in the data source.

```
EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Entity class> cq = cb.createQuery(Entity.class);
Root<Entity> from = cq.from(Entity.class);

cq.select(Entity);
TypedQuery<Entity> q = em.createQuery(cq);
List<Entity> allitems = q.getResultList();
```

The query demonstrates the basic steps to create a criteria.

- `EntityManager` instance is used to create a `CriteriaBuilder` object.
- `CriteriaQuery` instance is used to create a query object. This query object's attributes will be modified with the details of the query.
- `CriteriaQuery.form` method is called to set the query root.
- `CriteriaQuery.select` is called to set the result list type.
- `TypedQuery<T>` instance is used to prepare a query for execution and specifying the type of the query result.
- `getResultList` method on the `TypedQuery<T>` object to execute a query. This query returns a collection of entities, the result is stored in a List.

Example of criteria API

Let us consider the example of employee database. Let us assume the jpadb.employee table contains following records:

```
Eid Ename             Salary Deg
401 Gopal          40000 Technical Manager
402 Manisha          40000 Proof reader
403 Masthanvali      35000 Technical Writer
404     Satish           30000 Technical writer
```

```
405 Krishna          30000 Technical Writer
406 Kiran           35000 Proof reader
```

Create a JPA Project in the eclipse IDE named **JPA_Eclipselink_Criteria**. All the modules of this project are shown as follows:

Creating Entities

Create a package named **com.tutorialspoint.eclipselink.entity** under **'src'** package.

Create a class named **Employee.java** under given package. The class Employee entity is shown as follows:

```java
package com.tutorialspoint.eclipselink.entity;

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

@Entity
public class Employee {
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)

   private int eid;
   private String ename;
   private double salary;
   private String deg;

   public Employee(int eid, String ename, double salary, String deg) {
      super( );
      this.eid = eid;
      this.ename = ename;
      this.salary = salary;
      this.deg = deg;
   }

   public Employee( ) {
      super();
   }

   public int getEid( ) {
      return eid;
   }

   public void setEid(int eid) {
      this.eid = eid;
   }

   public String getEname( ) {
      return ename;
   }

   public void setEname(String ename) {
      this.ename = ename;
   }

   public double getSalary( ) {
      return salary;
   }

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

   public String getDeg( ) {
      return deg;
   }

   public void setDeg(String deg) {
      this.deg = deg;
   }

   @Override
   public String toString() {
   return "Employee [eid = " + eid + ", ename = " + ename + ", salary = " + salary + ",
deg = " + deg + "]";
   }
}
```

Persistence.xml

Persistence.xml file is required to configure the database and the registration of entity classes.

Persistence.xml will be created by the eclipse IDE while cresting a JPA Project. The configuration details are user specification. The persistence.xml file is shown as follows:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<persistence version="2.0" xmlns = "http://java.sun.com/xml/ns/persistence"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

   <persistence-unit name = "Eclipselink_JPA" transaction-type = "RESOURCE_LOCAL">
      <class>com.tutorialspoint.eclipselink.entity.Employee</class>

      <properties>
         <property name = "javax.persistence.jdbc.url" value =
"jdbc:mysql://localhost:3306/jpadb"/>
         <property name = "javax.persistence.jdbc.user" value = "root"/>
         <property name = "javax.persistence.jdbc.password" value = "root"/>
         <property name = "javax.persistence.jdbc.driver"
         value="com.mysql.jdbc.Driver"/>
         <property name = "eclipselink.logging.level" value = "FINE"/>
         <property name = "eclipselink.ddl-generation"
         value="create-tables"/>
      </properties>

   </persistence-unit>
</persistence>
```

Service classes

This module contains the service classes, which implements the Criteria query part using the MetaData API initialization. Create a package named **‘com.tutorialspoint.eclipselink.service’**. The class named **CriteriaAPI.java** is created under given package. The DAO class is shown as follows:

```
package com.tutorialspoint.eclipselink.service;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

import com.tutorialspoint.eclipselink.entity.Employee;

public class CriteriaApi {
   public static void main(String[] args) {
```

```
   EntityManagerFactory emfactory = Persistence.createEntityManagerFactory(
"Eclipselink_JPA" );
   EntityManager entitymanager = emfactory.createEntityManager( );
   CriteriaBuilder criteriaBuilder = entitymanager.getCriteriaBuilder();
   CriteriaQuery<Object> criteriaQuery = criteriaBuilder.createQuery();
   Root<Employee> from = criteriaQuery.from(Employee.class);

   //select all records
   System.out.println(“Select all records”);
   CriteriaQuery<Object> select = c riteriaQuery.select(from);
   TypedQuery<Object> typedQuery = entitymanager.createQuery(select);
   List<Object> resultlist = typedQuery.getResultList();

   for(Object o:resultlist) {
      Employee e = (Employee)o;
      System.out.println("EID : " + e.getEid() + " Ename : " + e.getEname());
   }

   //Ordering the records
   System.out.println(“Select all records by follow ordering”);
   CriteriaQuery<Object> select1 = criteriaQuery.select(from);
   select1.orderBy(criteriaBuilder.asc(from.get("ename")));
   TypedQuery<Object> typedQuery1 = entitymanager.createQuery(select);
   List<Object> resultlist1 = typedQuery1.getResultList();

   for(Object o:resultlist1){
      Employee e=(Employee)o;
      System.out.println("EID : " + e.getEid() + " Ename : " + e.getEname());
   }

   entitymanager.close( );
   emfactory.close( );
   }
}
```

After compilation and execution of the above program you will get output in the console panel of Eclipse IDE as follows:

```
Select All records
EID : 401 Ename : Gopal
EID : 402 Ename : Manisha
EID : 403 Ename : Masthanvali
EID : 404 Ename : Satish
EID : 405 Ename : Krishna
EID : 406 Ename : Kiran
Select All records by follow Ordering
EID : 401 Ename : Gopal
EID : 406 Ename : Kiran
EID : 405 Ename : Krishna
EID : 402 Ename : Manisha
EID : 403 Ename : Masthanvali
EID : 404 Ename : Satish
```