JPQL Hands-On Activity

Java Persistence Query language

JPQL is Java Persistence Query Language defined in JPA specification. It is used to create queries against entities to store in a relational database. JPQL is developed based on SQL syntax. But it won’t affect the database directly.

JPQL can retrieve information or data using the SELECT clause, can do bulk updates using the UPDATE clause and DELETE clause. EntityManager.createQuery() API will support querying language.

Query Structure

JPQL syntax is very similar to the syntax of SQL. Having SQL like syntax is an advantage because SQL is a simple structured query language and many developers are using it in applications. SQL works directly against relational database tables, records and fields, whereas JPQL works with Java classes and instances.

Task Overview

1. We will create JPA Project and Configure Jar Files for JPA AND MariaDB,
2. We will create two packages one is for Pojo/Model/EntityManager and second package is for Service.
3. We will create Two Pojo/Model/EntityManager Packages, One is for Employees and Second is for OrderDetails
4. We will create Four Class for Service under Service package
5. At the end we have to configure Persistence.xml for mapping

Steps are define below:

1. Create JPA Project on IDE, then attached JPA Jars and MariaDB Jar
2. Create a package **"com.test.jpa.employee",** Create an **entity class** named as **"EmployeeEntity.java"** under the package. Add below code and save it.

Remember: Entities are nothing Just simple class, It contains default constructor, setter and getter methods of those attributes.

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
 @Table(name="employees")
public class EmployeeEntity {
  @Id
 int employeeNumber;
 String firstName;
 String lastName;
 String email;
 int officeCode;

 public EmployeeEntity(int employeeNumber, String firstName, String
lastName, String email, int officeCode)
 {
        super();
     this.employeeNumber = employeeNumber;
     this.firstName = firstName;
     this.lastName = lastName;
     this.email = email;
     this.officeCode =officeCode;
 }

 public EmployeeEntity() {
     super();
 }
 public int getOfficecode() {
      return officeCode;
}
public void setOfficecode(int officecode) {
      this.officeCode = officecode;
}

 public String getEmail() {
      return email;
}
public void setEmail(String email) {
      this.email = email;
}


 public int getEmployeeName() {
      return employeeNumber;
}
public void setEmployeeName(int i) {
      this.employeeNumber = i;
}
public String getFirstName() {
      return firstName;
}
public void setFirstName(String firstName) {
      this.firstName = firstName;
}
public String getLastName() {
      return lastName;
}
public void setLastName(String lastName) {
      this.lastName = lastName;
}
}
```

3. Create a package **"com.test.jpa.employee",** Create an **entity class** named as **"orderEntity.java"** under the package. Add below code and save it

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
 @Table(name="orderdetails")
public class orderEntity {
      @Id
      int orderNumber;
      int orderLineNumber;
      String productCode;
      int priceEach;
      int quantityordered;



      public orderEntity() {
            super();
            // TODO Auto-generated constructor stub
      }


      public orderEntity(int priceEach, int quantityordered, int
orderNumber, int orderLineNumber, String productCode) {
        super();
        this.orderNumber = orderNumber;
        this.priceEach = priceEach;
        this.quantityordered = quantityordered;
        this.productCode = productCode;
        this.orderLineNumber = orderLineNumber;
    }
    public int getOrderNumbers() {
        return orderNumber;
    }

    public void setOrderNumbers(int orderNumber) {
        this.orderNumber = orderNumber;
    }
    public int getOrderLineNumber() {
        return orderLineNumber;
    }

    public void setOrderLineNumber(int orderLineNumber) {
        this.orderLineNumber = orderLineNumber;
    }

    public String getProductCode() {
        return productCode;
    }

    public void setProductCode(String productCode) {
        this.productCode = productCode;
    }
    public int getPriceEach() {
        return priceEach;
    }
    /**
     * @param priceEach the priceEach to set
     */
    public void setPriceEach(int priceEach) {
        this.priceEach = priceEach;
    }

    public int getQuantityorder() {
        return quantityordered;
    }
    /**
     * @param quantityorder the quantityorder to set
     */
    public void setQuantityorder(int quantityordered) {
        this.quantityordered = quantityordered;
    }
```

4. Configuration in **Persistence.xml file**. As shown below.
   Note: Do not forget to replace the persistence unit name
   <persistence-unit name="JPA-PQLTEST"> , You must replace "JPA-PQLTEST" with your project Name

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
	<persistence-unit name="JPA-PQLTEST">
	<class>com.test.jpa.employee.EmployeeEntity</class>
	<class>com.test.jpa.employee.orderEntity</class>
		<properties>
			<property name="javax.persistence.jdbc.driver"
value="org.mariadb.jdbc.Driver"/>
			<property name="javax.persistence.jdbc.url"
value="jdbc:mariadb://localhost/classicmodels"/>
			<property name="eclipselink.logging.level" value="FINE"/>
			<property name="javax.persistence.jdbc.user" value="root"/>
			<property name="javax.persistence.jdbc.password"
value="password"/>
			<property name="eclipselink.ddl-generation"
value="create-or-extend-tables"/>
		</properties>
	</persistence-unit>
</persistence>
```

5. Service: Create a package as **"com.test.jpa.services".** Create a **FindingEmployee** class under the package. Add below code in it.
   Note: You have to replace "JPA-PQLTEST" with your project Name or persistence unit

```
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

import com.test.jpa.employee.EmployeeEntity;
import com.test.jpa.employee.orderEntity;
```

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

           EntityManagerFactory emfactory =
Persistence.createEntityManagerFactory( "JPA-PQLTEST" );
           EntityManager entitymanager =
emfactory.createEntityManager();

//------------------------basic function-----------------------
                Query query =entitymanager.createQuery("Select
e.firstName from EmployeeEntity e");
                List<String> list = query.getResultList();

                for(String e:list) {
                    System.out.println("Employee NAME :"+e);
                 }

 //------- where  clause example-------------------------------

           Query  sql_two = entitymanager.createQuery("Select o from
orderEntity o where o.productCode like '%S24_1937%'");
            //sql_two.setParameter("givenID", "S24_1937");
           List<orderEntity> list_two =sql_two.getResultList();
           for(orderEntity result :list_two) {
               System.out.println("Order NAME :"+
result.getOrderNumbers());
               System.out.println("Price :"+ result.getPriceEach());
            }


//-------------------- between --------------------------

           Query sql_three = entitymanager.createQuery( "Select o " +
"from orderEntity o " +
            "where o.orderNumber " + "Between 100 and 2000" );
           List<orderEntity> list_three =sql_three.getResultList();
                for(orderEntity result :list_three) {
                    System.out.println("Order NAME :"+
result.getOrderNumbers());
                    System.out.println("Price :"+
result.getPriceEach());
                 }

 //---------Aggregate function example ----------------------

                 Query query1 = entitymanager.createQuery("Select
MAX(o.priceEach) from orderEntity o");
                 int result = (int) query1.getSingleResult();  //
Return object
                 System.out.println("Maximum Price:" + result);
            entitymanager.close();
            emfactory.close( );
        }
}
```

6. Create an **UpdatingEmployee** class under the package. Add below code in it.
   Note: You have to replace "JPA-PQLTEST" with your project Name or persistence unit

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

import com.test.jpa.employee.EmployeeEntity;

public class UpdatingEmployee {

     public static void main(String[] args) {
          // TODO Auto-generated method stub
          EntityManagerFactory emf =
Persistence.createEntityManagerFactory( "JPA-PQLTEST" );
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin( );

         Query query = em.createQuery( "update EmployeeEntity
SET email='abcd@gmail.com' where employeeNumber = 1088");
         query.executeUpdate();

        em.getTransaction().commit();
        em.close();
        emf.close();
     }

}
```

Eager and Lazy Loading

The main concept of JPA is to make a duplicate copy of the database in cache memory. While transacting with the database, first it will affect duplicate data and only when it is committed using entity manager, the changes are effected into the database.

There are two ways of fetching records from the database - eager fetch and lazy fetch.

Eager fetch

Fetching the whole record while finding the record using Primary Key.

Lazy fetch

It checks for the availability of notifications with the primary key if it exists. Then later if you call any of the getter methods of that entity then it fetches the whole.

But lazy fetch is possible when you try to fetch the record for the first time. That way, a copy of the whole record is already stored in cache memory. Performance wise, lazy fetch is preferable.