**JPQL примеры в Eclipselink**

**Я так понял, что Создание, Удаление, Изменение информации в БД требует begin + commit. А выборка – нет.**

**Создание сущностей**

Сущности — это не что иное, как бобы или модели. В этом примере мы будем использовать **Employee** в качестве объекта. **eid** , **ename** , **salary** и **deg** — это атрибуты этой сущности. Он содержит конструктор по умолчанию, а также методы установки и получения этих атрибутов.

В показанной выше иерархии создайте пакет с именем **‘com.tutorialspoint.eclipselink.entity’** в пакете **‘src’** (Source). Создайте класс с именем **Employee.java для** данного пакета следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table

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;

}

Constructor, Getter, Setter, toString()

В приведенном выше коде мы использовали аннотацию @Entity, чтобы сделать этот класс POJO сущностью.

Прежде чем перейти к следующему модулю, нам нужно создать базу данных для реляционной сущности, которая будет регистрировать базу данных в файле **persistence.xml** . Откройте MySQL Workbench и введите hte следующий запрос.

create database jpadb

use jpadb

**persistence.xml**

Этот модуль играет решающую роль в концепции JPA. В этом XML-файле мы зарегистрируем базу данных и укажем класс сущности.

В показанной выше иерархии пакетов файл persistence.xml в пакете JPA Content выглядит следующим образом:

<?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>

В приведенном выше XML-коде **тег <persistence-unit>** определен с определенным именем для персистентности JPA. Тег **<class>** определяет класс сущности с именем пакета. Тег **<properties>** определяет все свойства, а **тег <property>** определяет каждое свойство, такое как регистрация базы данных, спецификация URL, имя пользователя и пароль. Это свойства Eclipselink. Этот файл настроит базу данных.

**Постоянство Операции**

Постоянные операции используются для взаимодействия с базой данных и являются операциями **загрузки** и **хранения** . В бизнес-компоненте все постоянные операции подпадают под классы обслуживания.

В показанной выше иерархии пакетов создайте пакет с именем **‘com.tutorialspoint.eclipselink.service’** в пакете **‘src’** (source). Все сервисные классы с именами CreateEmloyee.java, UpdateEmployee.java, FindEmployee.java и DeleteEmployee.java. подпадает под данный пакет следующим образом:

**Создать сотрудника (есть begin + commit)**

В следующем фрагменте кода показано, как создать класс Employee с именем **CreateEmployee.java** .

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Employee;

public class CreateEmployee

{

public static void main( String[ ] args )

{

1.EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

2.EntityManager entitymanager = emfactory.createEntityManager( );

3.entitymanager.getTransaction( ).begin( );

Employee employee = new Employee( );

employee.setEid( 1201 );

employee.setEname( "Gopal" );

employee.setSalary( 40000 );

employee.setDeg( "Technical Manager" );

entitymanager.persist( employee );

entitymanager.getTransaction( ).commit( );

entitymanager.close( );

emfactory.close( );

}

}

В приведенном выше коде:

1.  **createEntityManagerFactory ()** создает модуль постоянства, предоставляя то же уникальное имя, которое мы предоставляем для модуля постоянства в файле persistent.xml.

2. объект **entitymanagerfactory** создаст экземпляр **entitymanger** с помощью **метода createEntityManager ()** .

3. объект **entitymanager** создает экземпляр **entitytransaction** для управления транзакциями.

Используя объект **entitymanager** , мы можем сохранить объекты в базе данных.

После компиляции и выполнения вышеуказанной программы вы получите уведомления из библиотеки eclipselink на панели консоли IDE eclipse.

Для результата откройте рабочую среду MySQL и введите следующие запросы.

use jpadb

select \* from employee

Созданная таблица базы данных с именем **employee** будет отображена в табличном формате следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| **Ид** | **Ename** | **Оплата труда** | **градус** |
| 1201 | Гопал | 40000 | Технический менеджер |

**Обновить сотрудника (есть begin + commit)**

Чтобы обновить записи сотрудника, нам нужно извлечь существующие записи из базы данных, внести изменения и, наконец, зафиксировать их в базе данных. Класс с именем **UpdateEmployee.java** показан следующим образом:

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Employee;

public class UpdateEmployee

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

Employee employee=entitymanager.find( Employee.class, 1201 );

//before update

System.out.println( employee );

employee.setSalary( 46000 );

entitymanager.getTransaction( ).commit( );

//after update

System.out.println( employee );

entitymanager.close();

emfactory.close();

}

}

После компиляции и выполнения вышеуказанной программы вы получите уведомления из библиотеки Eclipselink на панели консоли IDE eclipse.

Для результата откройте рабочую среду MySQL и введите следующие запросы.

use jpadb

select \* from employee

Созданная таблица базы данных с именем **employee** будет отображена в табличном формате следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| **Ид** | **Ename** | **Оплата труда** | **градус** |
| 1201 | Гопал | 46000 | Технический менеджер |

Заработная плата работника, 1201 обновлена ​​до 46000.

**Найти сотрудника (нет begin и commit)**

Чтобы найти записи сотрудника, нам нужно будет извлечь существующие данные из базы данных и отобразить их. В этой операции EntityTransaction (**3-й шаг**) не применяется при извлечении записи.

Класс с именем **FindEmployee.java** выглядит следующим образом.

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Employee;

public class FindEmployee

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence

.createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager();

Employee employee = entitymanager.find( Employee.class, 1201 );

System.out.println("employee ID = "+employee.getEid( ));

System.out.println("employee NAME = "+employee.getEname( ));

System.out.println("employee SALARY = "+employee.getSalary( ));

System.out.println("employee DESIGNATION = "+employee.getDeg( ));

}

}

После компиляции и выполнения вышеуказанной программы вы получите следующий вывод из библиотеки Eclipselink на панели консоли IDE eclipse.

employee ID = 1201

employee NAME = Gopal

employee SALARY = 46000.0

employee DESIGNATION = Technical Manager

**Удаление сотрудника (есть begin + commit)**

Чтобы удалить записи сотрудника, сначала мы найдем существующие записи, а затем удалим их. Здесь EntityTransaction играет важную роль.

Класс с именем **DeleteEmployee.java** выглядит следующим образом:

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Employee;

public class DeleteEmployee

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

Employee employee=entitymanager.find( Employee.class, 1201 );

entitymanager.remove( employee );

entitymanager.getTransaction( ).commit( );

entitymanager.close( );

emfactory.close( );

}

}

После компиляции и выполнения вышеуказанной программы вы получите уведомления из библиотеки Eclipselink на панели консоли IDE eclipse.

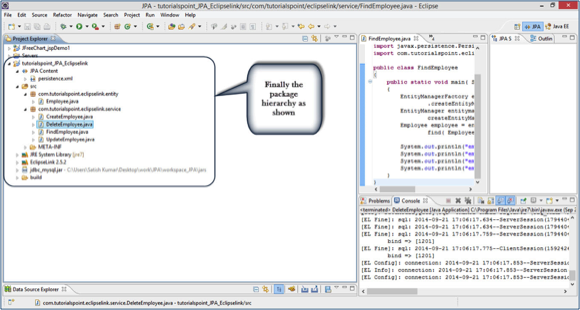
Для результата откройте рабочую среду MySQL и введите следующие запросы.

use jpadb

select \* from employee

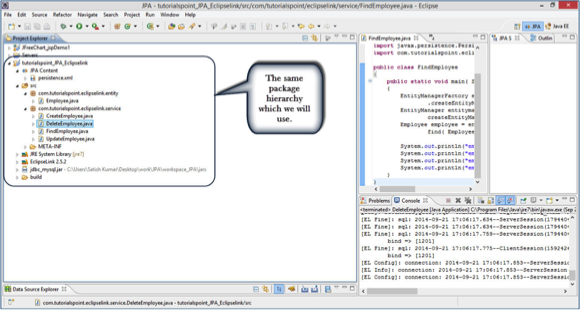
Созданная база данных с именем **employee** будет иметь нулевые записи.

После завершения всех модулей в этом примере иерархия пакетов и файлов выглядит следующим образом:



**JPA — JPQL**

В этой главе описывается JPQL и как он работает с постоянными модулями. В этой главе приведенные примеры следуют той же иерархии пакетов, которую мы использовали в предыдущей главе.



**Язык запросов персистентности Java**

JPQL расшифровывается как Java Persistence Query Language. Он используется для создания запросов к сущностям для хранения в реляционной базе данных. JPQL разработан на основе синтаксиса SQL. Но это не повлияет на базу данных напрямую.

JPQL может извлекать данные, используя предложение SELECT, может выполнять массовые обновления, используя предложение UPDATE и предложение DELETE.

**Структура запроса**

Синтаксис JPQL очень похож на синтаксис SQL. Наличие SQL-подобного синтаксиса является преимуществом, потому что SQL прост и широко используется. SQL работает непосредственно против таблиц, записей и полей реляционной базы данных, тогда как JPQL работает с классами и экземплярами Java.

Например, запрос JPQL может извлечь объект базы данных, а не набор результатов поля из базы данных, как в случае с SQL. Структура запроса JPQL выглядит следующим образом.

SELECT ... FROM ...

[WHERE ...]

[GROUP BY ... [HAVING ...]]

[ORDER BY ...]

Структура запросов JPQL DELETE и UPDATE следующая.

DELETE FROM ... [WHERE ...]

UPDATE ... SET ... [WHERE ...]

**Скалярные и агрегатные функции (нет begin и commit)**

Скалярные функции возвращают результирующие значения на основе входных значений. Агрегатные функции возвращают результирующие значения путем вычисления входных значений.

Мы будем использовать тот же пример управления сотрудниками, что и в предыдущей главе. Здесь мы пройдемся по классам обслуживания, используя скалярные и агрегатные функции JPQL.

Предположим, что таблица **jpadb.employee** содержит следующие записи.

|  |  |  |  |
| --- | --- | --- | --- |
| **Ид** | **Ename** | **Оплата труда** | **градус** |
| 1201 | Гопал | 40000 | Технический менеджер |
| 1202 | Маниша | 40000 | Proof Reader |
| 1203 | Masthanvali | 40000 | Технический писатель |
| 1204 | Сатиш | 30000 | Технический писатель |
| 1205 | Кришна | 30000 | Технический писатель |
| 1206 | Киран | 35000 | Proof Reader |

Создайте класс с именем **ScalarandAggregateFunctions.java** в пакете **com.tutorialspoint.eclipselink.service** следующим образом.

package com.tutorialspoint.eclipselink.service;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import javax.persistence.Query;

public class ScalarandAggregateFunctions

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager();

//Scalar function

Query query=entitymanager.createQuery("Select UPPER(e.ename) from Employee e");

List<String> list=query.getResultList();

for(String e:list)

{

System.out.println("Employee NAME :"+e);

}

//Aggregate function

Query query1 = entitymanager.

createQuery("Select MAX(e.salary) from Employee e");

Double result=(Double) query1.getSingleResult();

System.out.println("Max Employee Salary :"+result);

}

}

После компиляции и выполнения вышеуказанной программы вы получите следующий вывод на консольную панель Eclipse IDE.

Employee NAME :GOPAL

Employee NAME :MANISHA

Employee NAME :MASTHANVALI

Employee NAME :SATISH

Employee NAME :KRISHNA

Employee NAME :KIRAN

ax Employee Salary :40000.0

**Между, И, Как Ключевые слова (нет begin и commit)**

**Между** , **А** и **Лайк** являются основными ключевыми словами JPQL. Эти ключевые слова используются после **предложения Where** в запросе.

Создайте класс с именем **BetweenAndLikeFunctions.java** в пакете **com.tutorialspoint.eclipselink.service** следующим образом:

package com.tutorialspoint.eclipselink.service;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import javax.persistence.Query;

import com.tutorialspoint.eclipselink.entity.Employee;

public class BetweenAndLikeFunctions

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager();

//Between

Query query = entitymanager.

createQuery( "Select e " +

"from Employee e " +

"where e.salary " +

"Between 30000 and 40000" )

List<Employee> list=(List<Employee>)query.getResultList( );

for( Employee e:list )

{

System.out.print("Employee ID :"+e.getEid( ));

System.out.println("\t Employee salary :"+e.getSalary( ));

}

//Like

Query query1 = entitymanager.

createQuery("Select e " +

"from Employee e " +

"where e.ename LIKE 'M%'");

List<Employee> list1=(List<Employee>)query1.getResultList( );

for( Employee e:list1 )

{

System.out.print("Employee ID :"+e.getEid( ));

System.out.println("\t Employee name :"+e.getEname( ));

}

}

}

После компиляции и выполнения вышеуказанной программы вы получите следующий вывод на панели консоли Eclipse IDE.

Employee ID :1201 Employee salary :40000.0

Employee ID :1202 Employee salary :40000.0

Employee ID :1203 Employee salary :40000.0

Employee ID :1204 Employee salary :30000.0

Employee ID :1205 Employee salary :30000.0

Employee ID :1206 Employee salary :35000.0

Employee ID :1202 Employee name :Manisha

Employee ID :1203 Employee name :Masthanvali

**заказ (нет begin и commit)**

Чтобы упорядочить записи в JPQL, мы используем предложение ORDER BY. Использование этого предложения такое же, как в SQL, но оно касается сущностей. В следующем примере показано, как использовать предложение ORDER BY.

Создайте класс **Ordering.java** в пакете **com.tutorialspoint.eclipselink.service** следующим образом:

package com.tutorialspoint.eclipselink.service;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import javax.persistence.Query;

import com.tutorialspoint.eclipselink.entity.Employee;

public class Ordering

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager();

//Between

Query query = entitymanager.

createQuery( "Select e " +

"from Employee e " +

"ORDER BY e.ename ASC" );

List<Employee> list=(List<Employee>)query.getResultList( );

for( Employee e:list )

{

System.out.print("Employee ID :"+e.getEid( ));

System.out.println("\t Employee Name :"+e.getEname( ));

}

}

}

Скомпилировав и выполнив вышеуказанную программу, вы получите следующий вывод на панели консоли Eclipse IDE.

Employee ID :1201 Employee Name :Gopal

Employee ID :1206 Employee Name :Kiran

Employee ID :1205 Employee Name :Krishna

Employee ID :1202 Employee Name :Manisha

Employee ID :1203 Employee Name :Masthanvali

Employee ID :1204 Employee Name :Satish

**Именованные Запросы (нет begin и commit)**

Аннотация @NamedQuery определяется как запрос с предопределенной строкой запроса, которую нельзя изменить. В отличие от динамических запросов, именованные запросы могут улучшить организацию кода, отделяя строки запросов JPQL от POJO. Он также передает параметры запроса вместо динамического встраивания литералов в строку запроса и, следовательно, создает более эффективные запросы.

Прежде всего, добавьте аннотацию @NamedQuery к классу сущностей **Employee с** именем **Employee.java** в пакете **com.tutorialspoint.eclipselink.entity** следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

@Entity

@Table

@NamedQuery(query = "Select e from Employee e where e.eid = :id",

name = "find employee by id")

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;

}

Constructor, Getter, Setter, toString() }

Создайте класс с именем **NamedQueries.java** в пакете **com.tutorialspoint.eclipselink.service** следующим образом:

package com.tutorialspoint.eclipselink.service;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import javax.persistence.Query;

import com.tutorialspoint.eclipselink.entity.Employee;

public class NamedQueries

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager();

Query query = entitymanager.createNamedQuery("find employee by id");

query.setParameter("id", 1204);

List<Employee> list = query.getResultList( );

for( Employee e:list )

{

System.out.print("Employee ID :"+e.getEid( ));

System.out.println("\t Employee Name :"+e.getEname( ));

}

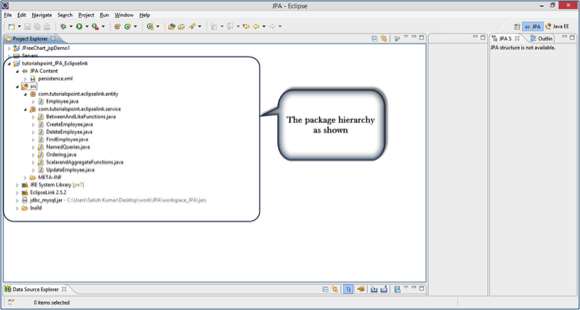
}

}

После компиляции и выполнения вышеуказанной программы вы получите следующий вывод на панели консоли Eclipse IDE.

Employee ID :1204 Employee Name :Satish

После добавления всех вышеперечисленных классов иерархия пакетов выглядит следующим образом:



**Стремительный и ленивый выбор**

Наиболее важной концепцией JPA является создание дубликата базы данных в кеш-памяти. При выполнении транзакций с базой данных JPA сначала создает дублированный набор данных, и только когда они фиксируются с помощью диспетчера сущностей, изменения вносятся в базу данных.

Есть два способа получения записей из базы данных.

**Нетерпеливый выбор**

При активном извлечении связанные дочерние объекты загружаются автоматически при извлечении определенной записи.

**Ленивый выбор**

При отложенном извлечении связанные объекты не загружаются автоматически, если вы специально не запрашиваете их. Прежде всего, он проверяет наличие связанных объектов и уведомляет. Позже, если вы вызовете какой-либо из методов получения этой сущности, он получит все записи.

Ленивая выборка возможна, когда вы пытаетесь получить записи в первый раз. Таким образом, копия всей записи уже хранится в кэш-памяти. С точки зрения производительности, ленивая выборка предпочтительнее.

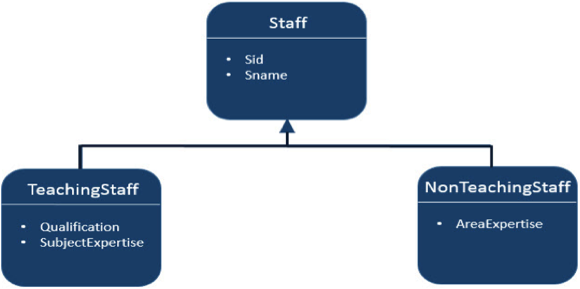
**JPA — Расширенные сопоставления**

JPA — это библиотека, выпущенная со спецификациями Java. Следовательно, он поддерживает все объектно-ориентированные концепции для сохранения сущности. До сих пор мы закончили с основами реляционного отображения объектов. В этой главе рассказывается о сложных сопоставлениях между объектами и реляционными сущностями.

**Стратегии наследования**

Наследование является основной концепцией любого объектно-ориентированного языка, поэтому мы можем использовать отношения наследования или стратегии между сущностями. JPA поддерживает три типа стратегий наследования: SINGLE\_TABLE, JOINED\_TABLE и TABLE\_PER\_CONCRETE\_CLASS.

Давайте рассмотрим пример. Следующая диаграмма показывает три класса, а именно. Сотрудники, TeachingStaff и NonTeachingStaff, а также их отношения.



На приведенной выше диаграмме Staff является сущностью, а TeachingStaff и NonTeachingStaff являются дочерними сущностями Staff. Здесь мы будем использовать приведенный выше пример, чтобы продемонстрировать все три три стратегии наследования.

**Стратегия единого стола**

Стратегия с одним столом берет все поля классов (как супер, так и подклассы) и отображает их в единую таблицу, известную как стратегия SINGLE\_TABLE. Здесь значение дискриминатора играет ключевую роль в дифференцировании значений трех объектов в одной таблице.

Давайте рассмотрим приведенный выше пример. TeachingStaff и NonTeachingStaff являются подклассами персонала. Согласно концепции наследования, подкласс наследует свойства своего суперкласса. Поэтому sid и sname являются полями, которые принадлежат как TeachingStaff, так и NonTeachingStaff. Создать проект JPA. Все модули этого проекта следующие:

**Создание сущностей**

Создайте пакет с именем **‘com.tutorialspoint.eclipselink.entity’** в пакете **‘src’** . Создайте новый Java-класс с именем **Staff.java** в указанном пакете. Класс сущности персонала отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import java.io.Serializable;

import javax.persistence.DiscriminatorColumn;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity

@Table

@Inheritance( strategy = InheritanceType.SINGLE\_TABLE )

@DiscriminatorColumn( name="type" )

public class Staff implements Serializable

{

@Id

@GeneratedValue( strategy = GenerationType.AUTO )

private int sid;

private String sname;

public Staff( int sid, String sname )

{

super( );

this.sid = sid;

this.sname = sname;

}

public Staff( )

{

super( );

}

Getter, Setter }

В приведенном выше коде **@DescriminatorColumn** указывает имя поля **(тип),** а его значения показывают оставшиеся поля (Teaching и NonTeachingStaff).

Создайте подкласс (класс) для класса Staff с именем **TeachingStaff.java** в пакете **com.tutorialspoint.eclipselink.entity** . Класс сущности TeachingStaff отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@DiscriminatorValue( value="TS" )

public class TeachingStaff extends Staff

{

private String qualification;

private String subjectexpertise;

public TeachingStaff( int sid, String sname,

String qualification,String subjectexpertise )

{

super( sid, sname );

this.qualification = qualification;

this.subjectexpertise = subjectexpertise;

}

public TeachingStaff( )

{

super( );

}

Getter, Setter }

Создайте подкласс (класс) для класса Staff с именем **NonTeachingStaff.java** в пакете **com.tutorialspoint.eclipselink.entity** . Класс Entity NonTeachingStaff показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@DiscriminatorValue( value = "NS" )

public class NonTeachingStaff extends Staff

{

private String areaexpertise;

public NonTeachingStaff( int sid, String sname,

String areaexpertise )

{

super( sid, sname );

this.areaexpertise = areaexpertise;

}

public NonTeachingStaff( )

{

super( );

}

Getter, Setter }

**persistence.xml**

Persistence.xml содержит информацию о конфигурации базы данных и информацию о регистрации классов сущностей. XML-файл отображается следующим образом:

<?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.Staff</class>

<class>com.tutorialspoint.eclipselink.entity.NonTeachingStaff</class>

<class>com.tutorialspoint.eclipselink.entity.TeachingStaff</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>

**Сервисный класс**

Классы обслуживания являются частью реализации бизнес-компонента. Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** .

Создайте класс с именем **SaveClient.java** в указанном пакете для хранения полей класса Staff, TeachingStaff и NonTeachingStaff. Класс SaveClient отображается следующим образом:

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.NonTeachingStaff;

import com.tutorialspoint.eclipselink.entity.TeachingStaff;

public class SaveClient

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Teaching staff entity

TeachingStaff ts1=new TeachingStaff(

1,"Gopal","MSc MEd","Maths");

TeachingStaff ts2=new TeachingStaff(

2, "Manisha", "BSc BEd", "English");

//Non-Teaching Staff entity

NonTeachingStaff nts1=new NonTeachingStaff(

3, "Satish", "Accounts");

NonTeachingStaff nts2=new NonTeachingStaff(

4, "Krishna", "Office Admin");

//storing all entities

entitymanager.persist(ts1);

entitymanager.persist(ts2);

entitymanager.persist(nts1);

entitymanager.persist(nts2);

entitymanager.getTransaction().commit();

entitymanager.close();

emfactory.close();

}

}

После компиляции и запуска вышеуказанной программы вы получите уведомления на панели консоли Eclipse IDE. Проверьте MySQL верстак для вывода. Вывод в табличном формате показан следующим образом:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sid** | **Тип** | **$ name** | **Areaexpertise** | **Квалификация** | **Subjectexpertise** |
| 1 | TS | Гопал |  | MSC MED | математика |
| 2 | TS | Маниша |  | BSC BED | английский |
| 3 | NS | Сатиш | Счета |  |  |
| 4 | NS | Кришна | Администратор офиса |  |  |

Наконец, вы получите одну таблицу, содержащую поле всех трех классов со столбцом дискриминатора с именем **Type** (field).

**Стратегия объединенного стола**

Стратегия присоединяемой таблицы состоит в том, чтобы совместно использовать ссылочный столбец, содержащий уникальные значения, для присоединения к таблице и выполнения простых транзакций. Давайте рассмотрим тот же пример, что и выше.

Создать проект JPA. Все модули проекта показаны ниже.

**Создание сущностей**

Создайте пакет с именем **‘com.tutorialspoint.eclipselink.entity’** в пакете **‘src’** . Создайте новый Java-класс с именем **Staff.java** в указанном пакете. Класс сущности персонала отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import java.io.Serializable;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity

@Table

@Inheritance( strategy = InheritanceType.JOINED )

public class Staff implements Serializable

{

@Id

@GeneratedValue( strategy = GenerationType.AUTO )

private int sid;

private String sname;

public Staff( int sid, String sname )

{

super( );

this.sid = sid;

this.sname = sname;

}

public Staff( )

{

super( );

}

Getter, Setter }

Создайте подкласс (класс) для класса Staff с именем **TeachingStaff.java** в пакете **com.tutorialspoint.eclipselink.entity** . Класс сущности TeachingStaff отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@PrimaryKeyJoinColumn(referencedColumnName="sid")

public class TeachingStaff extends Staff

{

private String qualification;

private String subjectexpertise;

public TeachingStaff( int sid, String sname,

String qualification,String subjectexpertise )

{

super( sid, sname );

this.qualification = qualification;

this.subjectexpertise = subjectexpertise;

}

public TeachingStaff( )

{

super( );

}

Getter, Setter }

Создайте подкласс (класс) для класса Staff с именем **NonTeachingStaff.java** в пакете **com.tutorialspoint.eclipselink.entity** . Класс Entity NonTeachingStaff показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@PrimaryKeyJoinColumn(referencedColumnName="sid")

public class NonTeachingStaff extends Staff

{

private String areaexpertise;

public NonTeachingStaff( int sid, String sname,

String areaexpertise )

{

super( sid, sname );

this.areaexpertise = areaexpertise;

}

public NonTeachingStaff( )

{

super( );

}

Getter, Setter }

**persistence.xml**

Файл Persistence.xml содержит информацию о конфигурации базы данных и информацию о регистрации классов сущностей. XML-файл отображается следующим образом:

<?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.Staff</class>

<class>com.tutorialspoint.eclipselink.entity.NonTeachingStaff</class>

<class>com.tutorialspoint.eclipselink.entity.TeachingStaff</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>

**Сервисный класс**

Классы обслуживания являются частью реализации бизнес-компонента. Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** .

Создайте класс с именем **SaveClient.java** в указанном пакете для хранения полей класса Staff, TeachingStaff и NonTeachingStaff. Тогда класс SaveClient показывается следующим образом:

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.NonTeachingStaff;

import com.tutorialspoint.eclipselink.entity.TeachingStaff;

public class SaveClient

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Teaching staff entity

TeachingStaff ts1=new TeachingStaff(

1,"Gopal","MSc MEd","Maths");

TeachingStaff ts2=new TeachingStaff(

2, "Manisha", "BSc BEd", "English");

//Non-Teaching Staff entity

NonTeachingStaff nts1=new NonTeachingStaff(

3, "Satish", "Accounts");

NonTeachingStaff nts2=new NonTeachingStaff(

4, "Krishna", "Office Admin");

//storing all entities

entitymanager.persist(ts1);

entitymanager.persist(ts2);

entitymanager.persist(nts1);

entitymanager.persist(nts2);

entitymanager.getTransaction().commit();

entitymanager.close();

emfactory.close();

}

}

После компиляции и запуска вышеуказанной программы вы получите уведомления на панели консоли Eclipse IDE. Для вывода проверьте MySQL Workbench.

Здесь создаются три таблицы, и результат таблицы **персонала** отображается в табличном формате.

|  |  |  |
| --- | --- | --- |
| **Sid** | **DTYPE** | **$ name** |
| 1 | Преподавательский состав | Гопал |
| 2 | Преподавательский состав | Маниша |
| 3 | NonTeachingStaff | Сатиш |
| 4 | NonTeachingStaff | Кришна |

Результат таблицы **TeachingStaff** отображается следующим образом:

|  |  |  |
| --- | --- | --- |
| **Sid** | **Квалификация** | **Subjectexpertise** |
| 1 | MSC MED | математика |
| 2 | BSC BED | английский |

В приведенной выше таблице sid — это внешний ключ (таблица персонала формы справочного поля). Результат таблицы **NonTeachingStaff** отображается следующим образом:

|  |  |
| --- | --- |
| **Sid** | **Areaexpertise** |
| 3 | Счета |
| 4 | Администратор офиса |

Наконец, три таблицы создаются с использованием соответствующих полей, а поле SID используется всеми тремя таблицами. В таблице Staff SID является первичным ключом. В двух оставшихся таблицах (TeachingStaff и NonTeachingStaff) SID является внешним ключом.

**Таблица для стратегии класса**

Стратегия для каждого класса заключается в создании таблицы для каждого подобъекта. Таблица Staff будет создана, но она будет содержать нулевые значения. Значения полей таблицы Staff должны совместно использоваться таблицами TeachingStaff и NonTeachingStaff.

Давайте рассмотрим тот же пример, что и выше.

**Создание сущностей**

Создайте пакет с именем **‘com.tutorialspoint.eclipselink.entity’** в пакете **‘src’** . Создайте новый Java-класс с именем **Staff.java** в указанном пакете. Класс сущности персонала отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import java.io.Serializable;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity

@Table

@Inheritance( strategy = InheritanceType.TABLE\_PER\_CLASS )

public class Staff implements Serializable

{

@Id

@GeneratedValue( strategy = GenerationType.AUTO )

private int sid;

private String sname;

public Staff( int sid, String sname )

{

super( );

this.sid = sid;

this.sname = sname;

}

public Staff( )

{

super( );

}

Getter, Setter }

Создайте подкласс (класс) для класса Staff с именем **TeachingStaff.java** в пакете **com.tutorialspoint.eclipselink.entity** . Класс сущности TeachingStaff отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

public class TeachingStaff extends Staff

{

private String qualification;

private String subjectexpertise;

public TeachingStaff( int sid, String sname,

String qualification,String subjectexpertise )

{

super( sid, sname );

this.qualification = qualification;

this.subjectexpertise = subjectexpertise;

}

public TeachingStaff( )

{

super( );

}

Getter, Setter }

Создайте подкласс (класс) для класса Staff с именем **NonTeachingStaff.java** в пакете **com.tutorialspoint.eclipselink.entity** . Класс Entity NonTeachingStaff показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

public class NonTeachingStaff extends Staff

{

private String areaexpertise;

public NonTeachingStaff( int sid, String sname,

String areaexpertise )

{

super( sid, sname );

this.areaexpertise = areaexpertise;

}

public NonTeachingStaff( )

{

super( );

}

Getter, Setter }

**persistence.xml**

Файл Persistence.xml содержит информацию о конфигурации базы данных и информацию о регистрации классов сущностей. XML-файл отображается следующим образом:

<?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.Staff</class>

<class>com.tutorialspoint.eclipselink.entity.NonTeachingStaff</class>

<class>com.tutorialspoint.eclipselink.entity.TeachingStaff</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>

**Сервисный класс**

Классы обслуживания являются частью реализации бизнес-компонента. Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** .

Создайте класс с именем **SaveClient.java** в указанном пакете для хранения полей класса Staff, TeachingStaff и NonTeachingStaff. Класс SaveClient отображается следующим образом:

package com.tutorialspoint.eclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.NonTeachingStaff;

import com.tutorialspoint.eclipselink.entity.TeachingStaff;

public class SaveClient

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Teaching staff entity

TeachingStaff ts1=new TeachingStaff(

1,"Gopal","MSc MEd","Maths");

TeachingStaff ts2=new TeachingStaff(

2, "Manisha", "BSc BEd", "English");

//Non-Teaching Staff entity

NonTeachingStaff nts1=new NonTeachingStaff(

3, "Satish", "Accounts");

NonTeachingStaff nts2=new NonTeachingStaff(

4, "Krishna", "Office Admin");

//storing all entities

entitymanager.persist(ts1);

entitymanager.persist(ts2);

entitymanager.persist(nts1);

entitymanager.persist(nts2);

entitymanager.getTransaction().commit();

entitymanager.close();

emfactory.close();

}

}

После компиляции и запуска вышеуказанной программы вы получите уведомления на панели консоли Eclipse IDE. Для вывода проверьте MySQL Workbench.

Здесь создаются три таблицы, а таблица **Staff** содержит нулевые записи.

Результат **TeachingStaff** отображается следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sid** | **Квалификация** | **$ name** | **Subjectexpertise** |
| 1 | MSC MED | Гопал | математика |
| 2 | BSC BED | Маниша | английский |

Приведенная выше таблица TeachingStaff содержит поля сущностей Staff и TeachingStaff.

Результат **NonTeachingStaff** отображается следующим образом:

|  |  |  |
| --- | --- | --- |
| **Sid** | **Areaexpertise** | **$ name** |
| 3 | Счета | Сатиш |
| 4 | Администратор офиса | Кришна |

Приведенная выше таблица NonTeachingStaff содержит поля сущностей Staff и NonTeachingStaff.

**JPA — отношения сущностей**

Эта глава проведет вас через отношения между сущностями. Как правило, отношения между таблицами в базе данных более эффективны. Здесь классы сущностей рассматриваются как реляционные таблицы (концепция JPA), поэтому отношения между классами сущностей таковы:

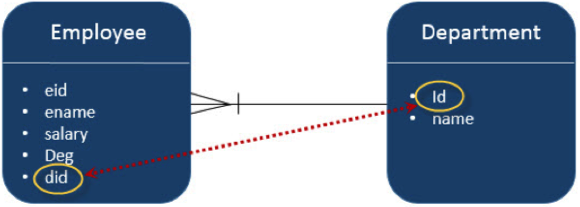
* @ManyToOne Relation
* @OneToMany Relation
* @OneToOne Relation
* @ManyToMany Relation

**@ManyToOne Relation**

Отношение «многие к одному» между сущностями существует, когда на одну сущность (столбец или набор столбцов) ссылается другая сущность (столбец или набор столбцов), содержащая уникальные значения. В реляционных базах данных эти отношения применяются с использованием внешнего ключа / первичного ключа между таблицами.

Давайте рассмотрим пример отношения между сотрудниками и подразделениями. Однонаправленным образом, то есть от сотрудника к отделу, применимо отношение «многие к одному». Это означает, что каждая запись сотрудника содержит один идентификатор отдела, который должен быть первичным ключом в таблице отдела. Здесь в таблице Employee идентификатор отдела — это внешний ключ.

На следующей диаграмме показано отношение «многие к одному» между двумя таблицами.



Создайте проект JPA в IDE затмения с именем **JPA\_Eclipselink\_MTO** . Все модули этого проекта обсуждаются ниже.

**Создание сущностей**

Следуйте приведенной выше схеме для создания сущностей. Создайте пакет с именем **‘com.tutorialspoin.eclipselink.entity’** в пакете **‘src’** . Создайте класс с именем **Department.java для** данного пакета. Класс Отдел сущности показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Department

{

@Id

@GeneratedValue( strategy=GenerationType.AUTO )

private int id;

private String name;

Getter, Setter }

Создайте в этом отношении второй объект — класс **объектов** **Employee с** именем **Employee.java** в **пакете com.tutorialspoint.eclipselink.entity** . Класс сущности Employee отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToOne;

@Entity

public class Employee

{

@Id

@GeneratedValue( strategy= GenerationType.AUTO )

private int eid;

private String ename;

private double salary;

private String deg;

@ManyToOne

private Department department;

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();

}

Getter, Setter

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

}

**persistence.xml**

Файл Persistence.xml необходим для настройки базы данных и регистрации классов сущностей.

Persitence.xml будет создан IDE затмения при создании проекта JPA. Детали конфигурации являются пользовательскими спецификациями. Файл persistence.xml отображается следующим образом:

<?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>

<class>com.tutorialspoint.eclipselink.entity.Department</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>

**Сервисные классы**

Этот модуль содержит классы обслуживания, которые реализуют реляционную часть с использованием атрибута инициализации. Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** . Класс DAO с именем **ManyToOne.java** создается под данным пакетом. Класс DAO показан следующим образом:

package com.tutorialspointeclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Department;

import com.tutorialspoint.eclipselink.entity.Employee;

public class ManyToOne

{

public static void main( String[ ] args )

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Create Department Entity

Department department = new Department();

department.setName("Development");

//Store Department

entitymanager.persist(department);

//Create Employee1 Entity

Employee employee1 = new Employee();

employee1.setEname("Satish");

employee1.setSalary(45000.0);

employee1.setDeg("Technical Writer");

employee1.setDepartment(department);

//Create Employee2 Entity

Employee employee2 = new Employee();

employee2.setEname("Krishna");

employee2.setSalary(45000.0);

employee2.setDeg("Technical Writer");

employee2.setDepartment(department);

//Create Employee3 Entity

Employee employee3 = new Employee();

employee3.setEname("Masthanvali");

employee3.setSalary(50000.0);

employee3.setDeg("Technical Writer");

employee3.setDepartment(department);

//Store Employees

entitymanager.persist(employee1);

entitymanager.persist(employee2);

entitymanager.persist(employee3);

entitymanager.getTransaction().commit();

entitymanager.close();

emfactory.close();

}

}

После компиляции и запуска вышеуказанной программы вы получите уведомления на панели консоли Eclipse IDE. Для вывода проверьте MySQL Workbench. В этом примере созданы две таблицы.

Передайте следующий запрос в интерфейсе MySQL, и результат таблицы **Department** будет отображаться следующим образом:

Select \* from department

|  |  |
| --- | --- |
| **Я БЫ** | **название** |
| 101 | развитие |

Передайте следующий запрос в интерфейсе MySQL, и результат таблицы **Employee** будет отображаться следующим образом.

Select \* from employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ид** | **градус** | **Ename** | **Оплата труда** | **department\_id** |
| 102 | Технический писатель | Сатиш | 45000 | 101 |
| 103 | Технический писатель | Кришна | 45000 | 101 |
| 104 | Технический писатель | Masthanwali | 50000 | 101 |

В приведенной выше таблице Deparment\_Id — это внешний ключ (поле ссылки) из таблицы Department.

**@OneToMany Relation**

В этом отношении каждая строка одного объекта ссылается на множество дочерних записей в другом объекте. Важно то, что у дочерних записей не может быть нескольких родителей. В отношении «один ко многим» между таблицей A и таблицей B каждая строка в таблице A может быть связана с одной или несколькими строками в таблице B.

Давайте рассмотрим приведенный выше пример. Предположим, что таблицы Employee и Department в вышеприведенном примере связаны в обратном однонаправленном порядке, тогда отношение становится отношением «один ко многим». Создайте проект JPA в IDE затмения с именем **JPA\_Eclipselink\_OTM** . Все модули этого проекта обсуждаются ниже.

**Создание сущностей**

Следуйте приведенной выше схеме для создания сущностей. Создайте пакет с именем **‘com.tutorialspoin.eclipselink.entity’** в пакете **‘src’** . Создайте класс с именем **Department.java для** данного пакета. Класс Отдел сущности показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import java.util.List;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

@Entity

public class Department

{

@Id

@GeneratedValue( strategy=GenerationType.AUTO )

private int id;

private String name;

@OneToMany( targetEntity=Employee.class )

private List employeelist;

Getter, Setter

public List getEmployeelist()

{

return employeelist;

}

public void setEmployeelist(List employeelist)

{

this.employeelist = employeelist;

}

}

Создайте в этом отношении вторую сущность — класс сущности **Employee с** именем **Employee.java** в **пакете com.tutorialspoint.eclipselink.entity** . Класс сущности Employee отображается следующим образом:

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();

}

Getter, Setter

}

**persistence.xml**

Файл persistence.xml выглядит следующим образом:

<?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>

<class>com.tutorialspoint.eclipselink.entity.Department</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>

**Сервисные классы**

Этот модуль содержит классы обслуживания, которые реализуют реляционную часть с использованием атрибута инициализации. Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** . Класс DAO с именем **OneToMany.java** создается под данным пакетом. Класс DAO показан следующим образом:

package com.tutorialspointeclipselink.service;

import java.util.List;

import java.util.ArrayList;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Department;

import com.tutorialspoint.eclipselink.entity.Employee;

public class OneToMany

{

public static void main(String[] args)

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Create Employee1 Entity

Employee employee1 = new Employee();

employee1.setEname("Satish");

employee1.setSalary(45000.0);

employee1.setDeg("Technical Writer");

//Create Employee2 Entity

Employee employee2 = new Employee();

employee2.setEname("Krishna");

employee2.setSalary(45000.0);

employee2.setDeg("Technical Writer");

//Create Employee3 Entity

Employee employee3 = new Employee();

employee3.setEname("Masthanvali");

employee3.setSalary(50000.0);

employee3.setDeg("Technical Writer");

//Store Employee

entitymanager.persist(employee1);

entitymanager.persist(employee2);

entitymanager.persist(employee3);

//Create Employeelist

List<Employee> emplist = new ArrayList();

emplist.add(employee1);

emplist.add(employee2);

emplist.add(employee3);

//Create Department Entity

Department department= new Department();

department.setName("Development");

department.setEmployeelist(emplist);

//Store Department

entitymanager.persist(department);

entitymanager.getTransaction().commit();

entitymanager.close();

emfactory.close();

}

}

После компиляции и выполнения вышеуказанной программы вы получите уведомления на панели консоли Eclipse IDE. Для вывода проверьте MySQL Workbench следующим образом.

В этом проекте созданы три таблицы. Передайте следующий запрос в интерфейсе MySQL, и результат таблицы Department\_employee будет отображаться следующим образом:

Select \* from department\_Id;

|  |  |
| --- | --- |
| **department\_id** | **Employee\_Eid** |
| 254 | 251 |
| 254 | 252 |
| 254 | 253 |

В приведенной выше таблице **deparment\_id** и **employee\_id** являются внешними ключами (справочными полями) из таблиц отделов и сотрудников.

Передайте следующий запрос в интерфейсе MySQL, и результат таблицы отделов будет отображен в табличном формате следующим образом.

Select \* from department;

|  |  |
| --- | --- |
| **Я БЫ** | **название** |
| 254 | развитие |

Передайте следующий запрос в интерфейс MySQL, и результат таблицы сотрудников будет отображаться следующим образом:

Select \* from employee;

|  |  |  |  |
| --- | --- | --- | --- |
| **Ид** | **градус** | **Ename** | **Оплата труда** |
| 251 | Технический писатель | Сатиш | 45000 |
| 252 | Технический писатель | Кришна | 45000 |
| 253 | Технический писатель | Masthanwali | 50000 |

**@OneToOne Relation**

В отношениях «один-к-одному» один элемент может быть связан только с одним другим элементом. Это означает, что каждая строка одного объекта относится к одной и только одной строке другого объекта.

Давайте рассмотрим приведенный выше пример. **Сотрудник** и **отдел** в обратной однонаправленной манере, отношение один-к-одному. Это означает, что каждый сотрудник принадлежит только одному отделу. Создайте проект JPA в IDE затмения с именем **JPA\_Eclipselink\_OTO** . Все модули этого проекта обсуждаются ниже.

**Создание сущностей**

Следуйте приведенной выше схеме для создания сущностей. Создайте пакет с именем **‘com.tutorialspoin.eclipselink.entity’** в пакете **‘src’** . Создайте класс с именем **Department.java для** данного пакета. Класс Отдел сущности показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Department

{

@Id

@GeneratedValue( strategy=GenerationType.AUTO )

private int id;

private String name;

Getter, Setter }

Создайте в этом отношении вторую сущность — класс сущности **Employee с** именем **Employee.java** в **пакете com.tutorialspoint.eclipselink.entity** . Класс сущности Employee отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

@Entity

public class Employee

{

@Id

@GeneratedValue( strategy= GenerationType.AUTO )

private int eid;

private String ename;

private double salary;

private String deg;

@OneToOne

private Department department;

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();

}

Getter, Setter

public Department getDepartment()

{

return department;

}

public void setDepartment(Department department)

{

this.department = department;

}

}

**persistence.xml**

Файл Persistence.xml выглядит следующим образом:

<?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>

<class>com.tutorialspoint.eclipselink.entity.Department</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>

**Сервисные классы**

Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** . Класс DAO с именем **OneToOne.java** создается под данным пакетом. Класс DAO показан следующим образом:

package com.tutorialspointeclipselink.service;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Department;

import com.tutorialspoint.eclipselink.entity.Employee;

public class OneToOne

{

public static void main(String[] args)

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Create Department Entity

Department department = new Department();

department.setName("Development");

//Store Department

entitymanager.persist(department);

//Create Employee Entity

Employee employee = new Employee();

employee.setEname("Satish");

employee.setSalary(45000.0);

employee.setDeg("Technical Writer");

employee.setDepartment(department);

//Store Employee

entitymanager.persist(employee);

entitymanager.getTransaction().commit();

entitymanager.close();

emfactory.close();

}

}

После компиляции и выполнения вышеуказанной программы вы получите уведомления на панели консоли Eclipse IDE. Для вывода проверьте MySQL Workbench следующим образом.

В приведенном выше примере созданы две таблицы. Передайте следующий запрос в интерфейс MySQL, и результат таблицы отделов будет отображаться следующим образом:

Select \* from department

|  |  |
| --- | --- |
| **Я БЫ** | **название** |
| 301 | развитие |

Передайте следующий запрос в интерфейс MySQL, и результат таблицы **сотрудников** будет отображаться следующим образом:

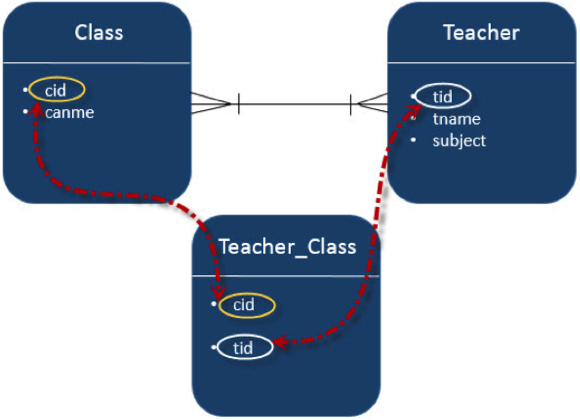
Select \* from employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ид** | **градус** | **Ename** | **Оплата труда** | **department\_id** |
| 302 | Технический писатель | Сатиш | 45000 | 301 |

**@ManyToMany Relation**

Отношение «многие ко многим» — это когда одна или несколько строк одного объекта связаны с несколькими строками другого объекта.

Давайте рассмотрим пример отношения между двумя сущностями: **классом** и **учителем** . В двунаправленной манере и класс, и учитель имеют отношение «многие к одному». Это означает, что каждая запись класса указывается набором учителя (идентификаторы учителя), которые должны быть первичными ключами в таблице учителя и храниться в таблице Teacher\_Class и наоборот. Здесь таблица Teachers\_Class содержит оба поля внешнего ключа. Создайте проект JPA в IDE затмения с именем **JPA\_Eclipselink\_MTM** . Все модули этого проекта обсуждаются ниже.



**Создание сущностей**

Создайте объекты, следуя схеме, показанной на диаграмме выше. Создайте пакет с именем **‘com.tutorialspoin.eclipselink.entity’** в пакете **‘src’** . Создайте класс с именем **Clas.java** под данным пакетом. Класс Отдел сущности показан следующим образом:

package com.tutorialspoint.eclipselink.entity;

import java.util.Set;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

@Entity

public class Clas

{

@Id

@GeneratedValue( strategy = GenerationType.AUTO )

private int cid;

private String cname;

@ManyToMany(targetEntity=Teacher.class)

private Set teacherSet;

public Clas()

{

super();

}

public Clas(int cid,

String cname, Set teacherSet)

{

super();

this.cid = cid;

this.cname = cname;

this.teacherSet = teacherSet;

}

Getter, Setter

public Set getTeacherSet()

{

return teacherSet;

}

public void setTeacherSet(Set teacherSet)

{

this.teacherSet = teacherSet;

}

}

Создайте в этом отношении вторую сущность — класс **сущности** **-работодателя с** именем **Teacher.java** в **пакете com.tutorialspoint.eclipselink.entity** . Класс сущности Employee отображается следующим образом:

package com.tutorialspoint.eclipselink.entity;

import java.util.Set;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

@Entity

public class Teacher

{

@Id

@GeneratedValue( strategy = GenerationType.AUTO )

private int tid;

private String tname;

private String subject;

@ManyToMany(targetEntity=Clas.class)

private Set clasSet;

public Teacher()

{

super();

}

public Teacher(int tid, String tname, String subject,

Set clasSet)

{

super();

this.tid = tid;

this.tname = tname;

this.subject = subject;

this.clasSet = clasSet;

}

Getter, Setter public Set getClasSet()

{

return clasSet;

}

public void setClasSet(Set clasSet)

{

this.clasSet = clasSet;

}

}

**persistence.xml**

Файл Persistence.xml выглядит следующим образом:

<?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>

<class>com.tutorialspoint.eclipselink.entity.Department</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>

**Сервисные классы**

Создайте пакет в пакете **‘src’ с** именем **‘com.tutorialspoint.eclipselink.service’** . Класс DAO с именем **ManyToMany.java** создается под данным пакетом. Класс DAO показан следующим образом:

package com.tutorialspoint.eclipselink.service;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import com.tutorialspoint.eclipselink.entity.Clas;

import com.tutorialspoint.eclipselink.entity.Teacher;

public class ManyToMany

{

public static void main(String[] args)

{

EntityManagerFactory emfactory = Persistence.

createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

//Create Clas Entity

Clas clas1=new Clas(0,"1st",null);

Clas clas2=new Clas(0,"2nd",null);

Clas clas3=new Clas(0,"3rd",null);

//Store Clas

entitymanager.persist(clas1);

entitymanager.persist(clas2);

entitymanager.persist(clas3);

//Create Clas Set1

Set<Clas> classSet1 = new HashSet();

classSet1.add(clas1);

classSet1.add(clas2);

classSet1.add(clas3);

//Create Clas Set2

Set<Clas> classSet2 = new HashSet();

classSet2.add(clas3);

classSet2.add(clas1);

classSet2.add(clas2);

//Create Clas Set3

Set<Clas> classSet3 = new HashSet();

classSet3.add(clas2);

classSet3.add(clas3);

classSet3.add(clas1);

//Create Teacher Entity

Teacher teacher1 = new Teacher(0,

"Satish","Java",classSet1);

Teacher teacher2 = new Teacher(0,

"Krishna","Adv Java",classSet2);

Teacher teacher3 = new Teacher(0,

"Masthanvali","DB2",classSet3);

//Store Teacher

entitymanager.persist(teacher1);

entitymanager.persist(teacher2);

entitymanager.persist(teacher3);

entitymanager.getTransaction( ).commit( );

entitymanager.close( );

emfactory.close( );

}

}

В этом примере проекта созданы три таблицы. Передайте следующий запрос в интерфейсе MySQL, и результат таблицы teacher\_clas будет отображаться следующим образом:

Select \* form teacher\_clas

|  |  |
| --- | --- |
| **Teacher\_tid** | **Classet\_cid** |
| 354 | 351 |
| 355 | 351 |
| 356 | 351 |
| 354 | 352 |
| 355 | 352 |
| 356 | 352 |
| 354 | 353 |
| 355 | 353 |
| 356 | 353 |

В приведенной выше таблице **teacher\_tid** является внешним ключом из таблицы учителя, а **classet\_cid** является внешним ключом из таблицы классов. Поэтому разные учителя отводятся в разные классы.

Передайте следующий запрос в интерфейсе MySQL, и результат таблицы учителя будет отображен следующим образом:

Select \* from teacher

|  |  |  |
| --- | --- | --- |
| **Врем** | **Предмет** | **TName** |
| 354 | Джава | Сатиш |
| 355 | Adv Java | Кришна |
| 356 | DB2 | Masthanvali |

Передайте следующий запрос в интерфейсе MySQL, и результат таблицы **clas** будет отображен следующим образом:

Select \* from clas

|  |  |
| --- | --- |
| **Cid** | **CNAME** |
| 351 | первый |
| 352 | второй |
| 353 | третий |