**4 ЛАБРАТОРНАЯ РАБОТА №4**

**ИЗУЧЕНИЕ ОБЪЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ**

* 1. **Цель работы**

Изучить технологию объектно-реляционного отображения JPA.

* 1. **Краткие теоретические сведения**

При разработке приложения, которое требует осуществления доступа к данным, необходимо упростить разработку такого приложения, увеличив эффективность и скорость работы с получаемыми данными. Поэтому данная проблема остаётся актуальной и сегодняшнее время.

ОБЪЕКТНО-РЕЛЯЦИОННОЕ отображение (ORM) – это техника программирования, которая связывает реляционную базу данных с концепциями объектно-ориентированного программирования и создаёт ’’виртуальную базу данных объектов’’. Причинами появления и развития этой техники программирования являются следующие:

1. Работа переносится с уровня работы с записями БД на другой уровень – уровень работы с объектами;
2. Большая гибкость организации доступа к данным (объект может состоять из полей различных таблиц);
3. Поддержка доступа нескольких пользователей к одним и тем же объектам (работа с persistent объектами)
4. Однообразный доступ к различным БД – разработчик не принимает во внимание сервер БД, с каким он работает (PostgreSQL, MySQL, Oracle и. д.);
5. Упрощение организации связи между объектами – ссылки на объекты;
6. Уменьшение времени доступа к данным за счёт использования механизма кэширования.

На сегодняшний день существуют различные Фреймворки, использующие данную технику программирования. Вот некоторые из них: Hibernate, iBATIS, Java Data Objects (JDO), JPOX, Cayenne, TopLink.

JPA не является новой технологией, а, скорее, это собрание идей лучших из имеющихся технологий, таких как Hibernate, TopLink и JDO. Как результат JPA является стандартизованной спецификацией, входящей в J2EE5, что позволяет строить слой сохранения данных независимо от каких-либо конкретных провайдеров. Т.е. реализаций спецификации JPA может быть много, одной из таких, например, является фреймворк OpenJPA или тот же Hibernate.

* **Persistence** – класс содержит вспомогательные статические методы для получения EntityManagerFactory независимым от поставщика способом;
* **EntityManagerFactory** – интерфейс, реализация которого является фабрикой для создания объектов EntityManager;
* **EntityManager** – является основным JPA интерфейсом используемый в приложениях. Каждый EntityManager управляет набором хранимых объектов и содержит API для вставки новых объектов и удаления существующих. С каждым EntityManager связан свой EntityTransaction и, также, EntityManager выступает фабрикой для объектов Query.
* **Entity** – сущность, которая является хранимым объектом.
* **EntityTransaction** – объект, который производит управления транзакциями при выполнении операций с хранимыми объектами Entity. Операции группируются и либо выполняются полностью, либо нет, оставляя хранилище данных в неизменном состоянии.
* **Query** – интерфейс для выполнения запросов по нахождению хранимых объектов, которые удовлетворяют заданным критериям. JPA поддерживает запросы на объектном языке Java Persistence Query Language (JPQL) и стандартном структурированном языке Structured Query Language (SQL). Получить экземпляры Query можно из объекта EntityManager.
  1. **Выполнение работы**
     1. **Packege ua.cn.stu.database**

package ua.cn.stu.database;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import javax.persistence.Query;

import javax.swing.table.DefaultTableModel;

import model.IJpaModel;

import model.Test;

public class JpaController {

private EntityManagerFactory emf;

private EntityManager em;

public JpaController() {

super();

this.emf = Persistence.createEntityManagerFactory("JTAP4");

}

public List<Test> getList() {

this.em = emf.createEntityManager();

Query q = em.createNamedQuery("Test.findAll");

List<Test> list = q.getResultList();

return list;

}

/\*

\* 0 - add 1 - update 2 - remove

\*/

public boolean operateObject(Test test, int mode) {

boolean isOk = false;

Class<? extends Test> clazz = test.getClass();

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

if (mode == 0 && !ifExist(clazz, test)) {

em.persist(test);

isOk = true;

} else if (mode == 1) {

Test tst = (Test) em.find(clazz, test.getId());

tst.update(test);

isOk = true;

} else if (mode == 2) {

Object obj = em.find(clazz, test.getId());

if (obj == null)

return false;

em.remove(obj);

System.out.println("OK");

isOk = true;

}

em.getTransaction().commit();

} catch (Exception e) {

e.printStackTrace();

em.getTransaction().rollback();

} finally {

em.close();

}

return isOk;

}

public boolean ifExist(Class<? extends IJpaModel> clazz, Test test) {

List<? extends Test> list = getList();

if (list != null && list.size() != 0)

for (Object current : list)

if (current.equals(test))

return true;

return false;

}

public DefaultTableModel getTableModel(List<Test> list) {

if (list.size() == 0)

return new DefaultTableModel();

String[] headrs = list.get(0).getHeadrs();

Object[][] arrData = new Object[list.size()][headrs.length];

for (int i = 0; i < list.size(); i++) {

arrData[i] = list.get(i).getData();

}

return new DefaultTableModel(arrData, headrs);}}

**4.3.2 Packege ua.cn.stu.model**

**package** model;

**public** **interface** IJpaModel {

**public** String[] getHeadrs();

**public** Object[] getData();

}

**package** model;

**import** java.io.Serializable;

**import** javax.persistence.\*;

/\*\*

\* The persistent class for the "Test" database table.

\*

\*/

@Entity

@Table(name = "\"Test\"")

@NamedQuery(name = "Test.findAll", query = "SELECT t FROM Test t")

**public** **class** Test **implements** Serializable, IJpaModel {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

**private** Integer id;

@Column(name = "\"BoolVariable\"")

**private** Boolean boolVariable;

@Column(name = "\"Name\"")

**private** String name;

@Column(name = "\"SecondName\"")

**private** String secondName;

**public** Test() {

}

**public** Test(Integer id, Boolean boolVariable, String name, String secondName) {

**super**();

**this**.id = id;

**this**.boolVariable = boolVariable;

**this**.name = name;

**this**.secondName = secondName;

}

**public** Integer getId() {

**return** **this**.id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** Boolean getBoolVariable() {

**return** **this**.boolVariable;

}

**public** **void** setBoolVariable(Boolean boolVariable) {

**this**.boolVariable = boolVariable;

}

**public** String getName() {

**return** **this**.name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getSecondName() {

**return** **this**.secondName;

}

**public** **void** setSecondName(String secondName) {

**this**.secondName = secondName;

}

@Override

**public** String[] getHeadrs() {

**return** **new** String[] { "id", "Name", "Second Name", "BoolVariable" };

}

@Override

**public** Object[] getData() {

**return** **new** String[] { String.*valueOf*(id), name, secondName, String.*valueOf*(boolVariable) };

}

**public** **void** update(Test tst) {

**this**.id = tst.getId();

**this**.boolVariable = tst.getBoolVariable();

**this**.name = tst.getName();

**this**.secondName = tst.getSecondName();

}

}

* 1. **Выводы**

В ходе выполнения данной лабораторной работы были изучены основы работы с технологией объектно-реляционного отображения JPA. Создано приложение, которое может производить простейшие операции доступа и модификации данных с применением JPA.