Министерство науки и образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных технологий

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ОТЧЕТ**

по лабораторной работе № 6

по дисциплине «Объектно-ориентированное программирование»

### Технология JDBC

ОГУ 09.03.04.4023.926 ПЗ

Руководитель

канд. техн. наук, доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. А. Тишина

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Исполнитель

Студент группы 22ПИ(ба)РПиС-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.Р. Безруков

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Оренбург 2023

**Содержание**

1. Постановка задачи. Цель, задание……………………………………….
2. Теоретические предпосылки…. …………………………………………
3. Практическая часть………………………………………………………
4. Выводы…………………………………………………………………….

Список использованных источников…………………………………………

Приложение А. Исходный текст программы…………………………………

**Постановка задачи. Цель, задание**

Задание

1 Изучите теоретический материал.

2 Создайте две таблицы базы данных со связью один ко многим для хранения данных объектов иерархии классов согласно продуктов. 3 Разработайте программное средство (ПС), реализующее работу с полученными таблицами. ПС должно содержать методы для добавления, редактирования, удаления, поиска и сохранения данных на основе SQLзапросов.

4 Протестируйте приложение.

Скрипты для создании таблицы:

CREATE TABLE employees

(

id SERIAL PRIMARY KEY,

surname CHARACTER VARYING(30),

name CHARACTER VARYING(30),

patronymic CHARACTER VARYING(30),

role\_id BIGINT,

FOREIGN KEY (role\_id) REFERENCES employee\_roles(id)

);

CREATE TABLE employee\_roles

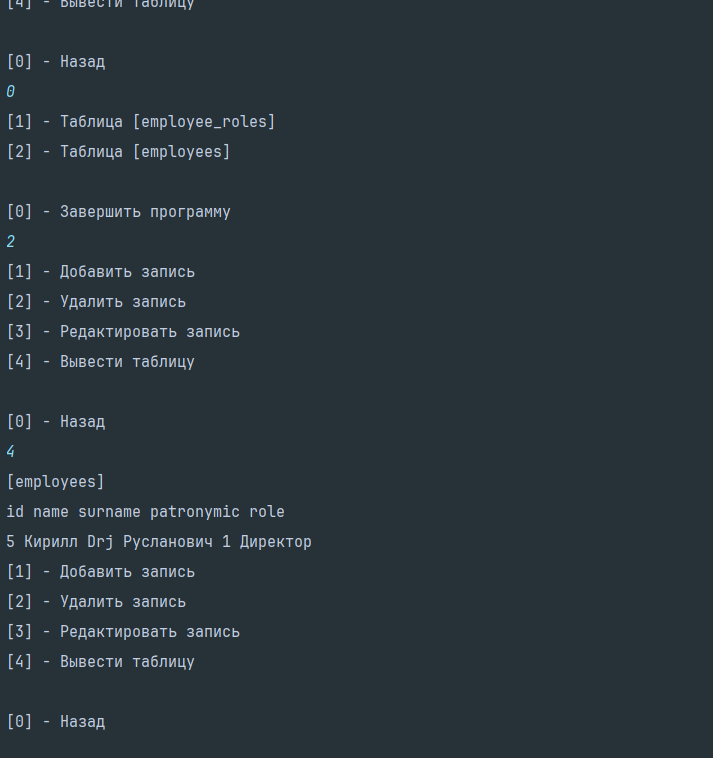
(

id SERIAL PRIMARY KEY,

name CHARACTER VARYING(30)

);

**Результат работы программы**

****

**Список литературы**

1. Вязовик, Н. А. Программирование на Java [Текст] : курс лекций / Н. А. Вязовик. - М. : Интернет-Ун-т Информ. Технологий, 2003. - 592 с. - (Основы информационных технологий). - Библиогр.: с. 585-586. - ISBN 5-9556-0006-Х.
2. Лаптев В.В. C++. Объектно-ориентированное программирование : учеб. пособие / В. В. Лаптев. - Санкт Петербург : Питер, 2008. - 464 с. - ISBN 978-5-91180-200-4
3. Лафоре, Р.  Объектно-ориентированное программирование в С++ = Object-OrientedProgrammingin C++ [Текст]  / Р. Лафоре.- 4-е изд. - CПб. : Питер, 2008. - 928 с. : ил. - (Классика ComputerScience). - Парал. тит. л. англ. - Прил.: с. 796-901. - Алф. указ.: с. 902-923. - ISBN 978-5-94723-302-5.

**Приложение А  
(*Обязательное*)  
Исходный текст программы**

Текст программы

import db.EmployeeRolesApi;  
import model.Employee;  
import table.EmployeeRolesTable;  
import table.EmployeesTable;  
  
import java.io.IOException;  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.in);  
 while(true) {  
 System.out.println("[1] - Таблица [employee\_roles]\n[2] - Таблица [employees]\n\n[0] - Завершить программу");  
 String res = in.next();  
 if(res.equals("0"))  
 break;  
 if(res.equals("1") || res.equals("2")) {  
 while(true) {  
 System.out.println("[1] - Добавить запись\n[2] - Удалить запись\n[3] - Редактировать запись\n[4] - Вывести таблицу\n\n[0] - Назад");  
 String res2 = in.next();  
 if(res2.equals("0"))  
 break;  
 switch (res2) {  
 case("1"):  
 if(res.equals("1")) {  
 System.out.print("Введите название роли: ");  
 res2 = in.next();  
 EmployeeRolesTable.addRole(res2);  
 } else {  
 System.out.print("Введите фамилию, имя, отчество и id роли(через пробел):\n");  
 res2 = "";  
 for(int i =0;i<4;i++)  
 res2 += in.next() + " ";  
 String[] temp = res2.split(" ");  
 if(temp.length != 4)  
 break;  
 EmployeesTable.add(temp[0], temp[1], temp[2], Integer.parseInt(temp[3]));  
 }  
 break;  
 case("2"):  
 if(res.equals("1")) {  
 System.out.print("Введите название роли: ");  
 res2 = in.next();  
 EmployeeRolesTable.removeRole(res2);  
 } else {  
 System.out.print("Введите id сотрудника: ");  
 int id = in.nextInt();  
 EmployeesTable.remove(id);  
 }  
 break;  
 case("3"):  
 if(res.equals("1")) {  
 System.out.print("Введите id роли: ");  
 int id = in.nextInt();  
 System.out.print("Введите новое название роли: ");  
 res2 = in.next();  
 EmployeeRolesTable.editRole(id, res2);  
 } else {  
 System.out.print("Введите id сотрудника: ");  
 int id = in.nextInt();  
 Employee employee = EmployeesTable.getEmployeeById(id);  
 if(employee == null)  
 {  
 System.err.println("Сотрудника с таким id не существует!");  
 break;  
 }  
 System.out.println(employee);  
 Map<String, String> map = Map.of(  
 "surname", "Фамилия",  
 "name", "Имя",  
 "patronymic", "Отчёство",  
 "roleId", "Роль"  
 );  
 int tI = 1;  
 Set<String> tS = map.keySet();  
 for(String str : tS) {  
 System.out.printf("[%s] - %s\n", tI++, map.get(str));  
 }  
 System.out.print("Что хотите изменить: ");  
 int tempN = in.nextInt();  
 if(tempN < 1 || tempN > tS.size())  
 break;  
 System.out.print("Введите новое значение: ");  
 String tempS = in.next();  
 tI = 1;  
 for(String str : tS)  
 if(tI++ == tempN) {  
 EmployeesTable.edit(id, str, tempS);  
 break;  
 }  
 }  
 break;  
 case("4"):  
 if(res.equals("1")) {  
 System.out.println(EmployeeRolesTable.toStr());  
 } else {  
 System.out.println(EmployeesTable.toStr());  
 }  
 break;  
 }  
 }  
 }  
 }  
 }  
}

package table;  
  
import db.EmployeesApi;  
import model.Employee;  
import model.EmployeeRole;  
  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.Queue;  
import java.util.stream.Collectors;  
  
public class EmployeesTable {  
 private static final ArrayList<Employee> employees = EmployeesApi.getTable();  
 public static Employee getEmployeeById(int id) {  
 assert employees != null;  
 for (Employee employee : employees) {  
 if(employee.getId() == id) {  
 return employee;  
 }  
 }  
 return null;  
 }  
 public static void add(String surname, String name, String patronymic, int roleId) {  
 Employee employee = EmployeesApi.addEmployee(surname, name, patronymic, roleId);  
 if(employee!=null)  
 employees.add(employee);  
 }  
  
 public static void remove(int id) {  
 Employee employee = getEmployeeById(id);  
 if(employee != null)  
 remove(employee);  
 }  
 public static void remove(Employee role) {  
 if(EmployeesApi.deleteEmployee(role.getId()))  
 employees.remove(role);  
 }  
 public static void removeByRole(EmployeeRole role) {  
 ArrayList<Employee> toRemove = new ArrayList<>();  
 for(Employee employee : employees)  
 if(employee.getRole() == role)  
 toRemove.add(employee);  
 toRemove.forEach(employees::remove);  
 toRemove.clear();  
 }  
 public static void edit(int id, String colName, String newValue) {  
 Employee employee = getEmployeeById(id);  
 if(employee != null)  
 editRole(employee, colName, newValue);  
 }  
 public static void editRole(Employee employee, String colName, String newValue) {  
 if(EmployeesApi.editEmployee(employee.getId(), colName, newValue)) {  
 assert employees != null;  
 employees.remove(employee);  
 String name = employee.getName();  
 String surname = employee.getSurname();  
 String patronymic = employee.getPatronymic();  
 EmployeeRole employeeRole = employee.getRole();  
 switch (colName) {  
 case "surname" -> surname = newValue;  
 case "name" -> name = newValue;  
 case "patronymic" -> patronymic = newValue;  
 case "roleId" -> employeeRole = EmployeeRolesTable.getRoleById(Integer.parseInt(newValue));  
 }  
 employees.add(new Employee(employee.getId(), surname, name, patronymic, employeeRole));  
 }  
 }  
 public static String toStr() {  
 return "[employees]\nid name surname patronymic role\n%s".formatted(employees.stream().sorted(Comparator.comparing(Employee::getId)).map(Employee::toString).collect(Collectors.joining("\n")));  
 }  
}

package table;  
  
import db.EmployeeRolesApi;  
import model.EmployeeRole;  
  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.stream.Collectors;  
  
public class EmployeeRolesTable {  
 private static final ArrayList<EmployeeRole> roles = EmployeeRolesApi.getTable();  
 public static EmployeeRole getRoleById(int id) {  
 assert roles != null;  
 for (EmployeeRole role : roles) {  
 if(role.getId() == id) {  
 return role;  
 }  
 }  
 return null;  
 }  
 public static void addRole(String name) {  
 EmployeeRole employeeRole = EmployeeRolesApi.addRole(name);  
 if(employeeRole!=null)  
 roles.add(employeeRole);  
 }  
  
 public static void removeRole(String name) {  
 for (EmployeeRole role : roles) {  
 if(role.getName().equals(name)) {  
 removeRole(role);  
 return;  
 }  
 }  
 }  
 public static void removeRole(EmployeeRole role) {  
 if(EmployeeRolesApi.deleteRole(role.getName())) {  
 EmployeesTable.removeByRole(role);  
 roles.remove(role);  
 }  
  
 }  
 public static void editRole(int id, String newName) {  
 EmployeeRole role = getRoleById(id);  
 if(role != null)  
 editRole(role, newName);  
 }  
 public static void editRole(EmployeeRole role, String newName) {  
 if(EmployeeRolesApi.editRole(role.getId(), newName)) {  
 assert roles != null;  
 roles.remove(role);  
 roles.add(new EmployeeRole(role.getId(), newName));  
 }  
 }  
 public static String toStr() {  
 return "[employee\_roles]\nid name\n%s".formatted(roles.stream().sorted(Comparator.comparing(EmployeeRole::getId)).map(EmployeeRole::toString).collect(Collectors.joining("\n")));  
 }  
}

package model;  
  
import java.lang.reflect.Field;  
import java.util.Arrays;  
import java.util.stream.Collectors;  
  
public class Model {  
 @Override  
 public String toString() {  
 StringBuilder sb = new StringBuilder();  
 Field[] fields = this.getClass().getDeclaredFields();  
 for (Field field : fields) {  
 field.setAccessible(true);  
 try {  
 Object value = field.get(this);  
 sb.append(value).append(" ");  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
 }  
 return sb.toString().trim();  
 }  
}

package model;  
  
import table.EmployeeRolesTable;  
  
public class EmployeeRole extends Model {  
 private final int id;  
 private final String name;  
  
 public EmployeeRole(int id, String name) {  
 this.id = id;  
 this.name = name;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }

package model;  
  
public class Employee extends Model {  
 private final int id;  
 private final String name;  
 private final String surname;  
 private final String patronymic;  
 private final EmployeeRole role;  
  
 public Employee(int id, String surname, String name, String patronymic, EmployeeRole role) {  
 this.id = id;  
 this.name = name;  
 this.surname = surname;  
 this.patronymic = patronymic;  
 this.role = role;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public EmployeeRole getRole() {  
 return role;  
 }  
}

package db;  
  
import model.Employee;  
import model.EmployeeRole;  
import table.EmployeeRolesTable;  
  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
  
public class EmployeesApi {  
  
 private static int getIdLastRow() throws SQLException, ClassNotFoundException {  
 var temp = DataBaseApi.getTable("employees");  
 return Integer.parseInt(temp.get(temp.size()-1).get(0));  
 }  
 public static Employee addEmployee(String surname, String name, String patronymic, int roleId) {  
 try {  
 DataBaseApi.executeQuery("INSERT INTO employees (surname, name, patronymic, role\_id) values('%s', '%s', '%s', %s)".formatted(surname, name, patronymic, roleId));  
 int id = getIdLastRow();  
 return new Employee(id, surname, name, patronymic, EmployeeRolesTable.getRoleById(roleId));  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return null;  
 }  
  
 public static boolean deleteEmployee(int id) {  
 try {  
 DataBaseApi.executeQuery("DELETE FROM employees WHERE id='%s'".formatted(id));  
 return true;  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return false;  
 }  
  
 public static boolean editEmployee(int id, String colName, String value) {  
 try {  
 DataBaseApi.executeQuery("UPDATE employees SET %s='%s' WHERE id=%s".formatted(colName, value, id));  
 return true;  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return false;  
 }  
  
 public static ArrayList<Employee> getTable() {  
 try {  
 ArrayList<Employee> roles = new ArrayList<>();  
 ArrayList<ArrayList<String>> temp = DataBaseApi.getTable("employees");  
 temp.forEach(x-> roles.add(new Employee(Integer.parseInt(x.get(0)), x.get(1), x.get(2), x.get(3), EmployeeRolesTable.getRoleById(Integer.parseInt(x.get(4))))));  
 return roles;  
  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return null;  
 }  
}

package db;  
  
import model.EmployeeRole;  
  
import java.sql.SQLException;  
import java.util.ArrayList;  
  
public class EmployeeRolesApi {  
 private static int getIdLastRow() throws SQLException, ClassNotFoundException {  
 var temp = DataBaseApi.getTable("employee\_roles");  
 return Integer.parseInt(temp.get(temp.size()-1).get(0));  
 }  
 public static String getName(int id) throws SQLException, ClassNotFoundException {  
 return DataBaseApi.getData("employee\_roles", "name", "id", Integer.toString(id));  
 }  
 public static EmployeeRole addRole(String name) {  
 try {  
 DataBaseApi.executeQuery("INSERT INTO employee\_roles (name) values('%s')".formatted(name));  
 int id = getIdLastRow();  
 return new EmployeeRole(id, name);  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return null;  
 }  
  
 public static boolean deleteRole(String name) {  
 try {  
 DataBaseApi.executeQuery("DELETE FROM employee\_roles WHERE name='%s'".formatted(name));  
 return true;  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return false;  
 }  
  
 public static boolean editRole(int id, String newName) {  
 try {  
 DataBaseApi.executeQuery("UPDATE employee\_roles SET name='%s' WHERE id=%s".formatted(newName, id));  
 return true;  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return false;  
 }  
  
 public static ArrayList<EmployeeRole> getTable() {  
 try {  
 ArrayList<EmployeeRole> roles = new ArrayList<>();  
 ArrayList<ArrayList<String>> temp = DataBaseApi.getTable("employee\_roles");  
 temp.forEach(x-> roles.add(new EmployeeRole(Integer.parseInt(x.get(0)), x.get(1))));  
 return roles;  
  
 }  
 catch (Exception e) {  
 System.err.println(e.getMessage());  
 }  
 return null;  
 }  
}

package db;  
  
import org.postgresql.util.PSQLException;  
  
import java.sql.\*;  
import java.util.ArrayList;  
  
public class DataBaseApi {  
 private final static String url = "jdbc:postgresql://ep-jolly-pond-03245931.eu-central-1.aws.neon.tech/carservdb?user=Croshaw&password=QxqO2vl5mrGD&sslmode=require";  
 private static Connection connection;  
 private static Statement statement;  
 private static DatabaseMetaData databaseMetaData;  
  
 private static void openConnection() throws ClassNotFoundException, SQLException {  
 Class.forName("org.postgresql.Driver");  
 connection = DriverManager.getConnection(url);  
 statement = connection.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);  
 databaseMetaData = connection.getMetaData();  
 }  
  
 private static void closeConnection() throws SQLException {  
 statement.close();  
 connection.close();  
 databaseMetaData = null;  
 }  
  
 public static ArrayList<String> getTablesNames() throws SQLException, ClassNotFoundException {  
 openConnection();  
 ResultSet tables = databaseMetaData.getTables(null, null, "%", new String[]{"TABLE"});  
 ArrayList<String> result = new ArrayList<>();  
 while(tables.next()) {  
 result.add(tables.getString("TABLE\_NAME"));  
 }  
 closeConnection();  
 return result;  
 }  
  
 public static void executeQuery(String query) throws SQLException, ClassNotFoundException {  
 openConnection();  
 ResultSet rs;  
 try {  
 rs = statement.executeQuery(query);  
 rs.close();  
 }  
 catch (PSQLException e) {  
   
 }  
 closeConnection();  
 }  
  
 public static ArrayList<ArrayList<String>> getTable(String tableName) throws SQLException, ClassNotFoundException {  
 openConnection();  
 ResultSet rs = statement.executeQuery("SELECT \* FROM %s".formatted(tableName));  
 ArrayList<ArrayList<String>> result = new ArrayList<>();  
 int colCount = rs.getMetaData().getColumnCount();  
 while(rs.next()) {  
 ArrayList<String> temp = new ArrayList<>();  
 for(int i = 1; i <= colCount; i++)  
 temp.add(rs.getString(i));  
 result.add(temp);  
 }  
 rs.close();  
 closeConnection();  
 return result;  
 }  
  
 public static String getData(String tableName, String columnName, String searchName, String searchValue) throws SQLException, ClassNotFoundException {  
 openConnection();  
 String result = "";  
 ResultSet rs = statement.executeQuery("SELECT %s FROM %s WHERE %s=%s".formatted(columnName, tableName, searchName, searchValue));  
 if(rs.first())  
 result = rs.getString(1);  
 rs.close();  
 closeConnection();  
 return result;  
 }  
}