Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**науки и высшего образования**

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

**(ВлГУ)**

**Кафедра информационных систем и программной инженерии**

Лабораторная работа №3

по дисциплине "Распределенные программные системы"

по теме: Сущности.

Выполнил:

студент гр. ПРИ-117

Журавлев Н.М.

Принял:

Трифонов Д.А.

Владимир 2020 г.

ЦЕЛЬ РАБОТЫ

Освоить создание классов сущностей, в том числе путем обратного проектирования реляционной базы данных средствами NetBeans 5.5 и работу с менеджером сущностей и получить навыки работы со средой разработки.

ВАРИАНТ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

В качестве индивидуального задания была выбрана реализация веб-приложения, а именно база данных, из лабараторной работы №2

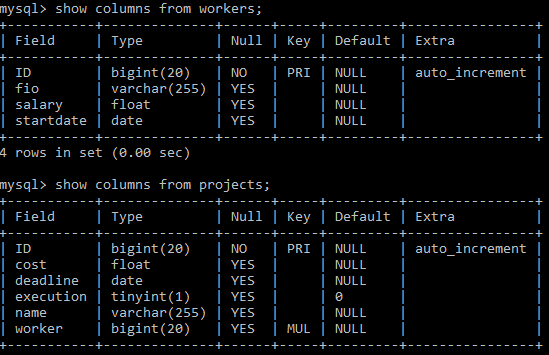


Рисунок 1. Структура базы данных.

ХОД РАБОТЫ

На первом этапе лабараторной работы были сгенерированны классы сущностей базы данных, сессионные компоненты для этих классов, а также контроллеры и представления на основе сессионных компонентов. Был создан и настроен модуль персистентности для связи приложения с базой данных. Приложение было собрано и запущено. Ниже приведены примеры структуры, сгенерированных сущностей и страниц веб-приложения.

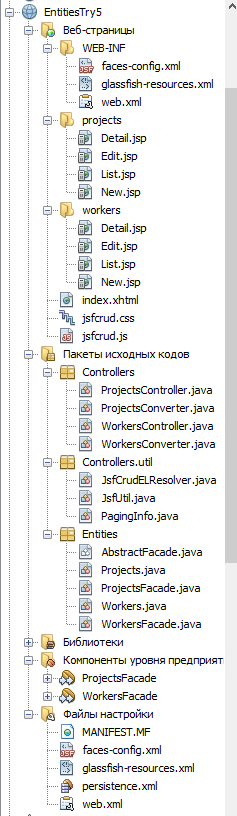


Рисунок 2. Структура автоматически сгенерированного web-приложения.

Код класса сущности Projects.java:

package Entities;

import java.io.Serializable;

import java.util.Date;

import javax.persistence.Basic;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

import javax.persistence.Temporal;

import javax.persistence.TemporalType;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

import javax.xml.bind.annotation.XmlRootElement;

@Entity

@Table(name = "projects", catalog = "lr6", schema = "")

@XmlRootElement

@NamedQueries({

@NamedQuery(name = "Projects.findAll", query = "SELECT p FROM Projects p")

, @NamedQuery(name = "Projects.findByProjectId", query = "SELECT p FROM Projects p WHERE p.projectId = :projectId")

, @NamedQuery(name = "Projects.findByName", query = "SELECT p FROM Projects p WHERE p.name = :name")

, @NamedQuery(name = "Projects.findByDeadline", query = "SELECT p FROM Projects p WHERE p.deadline = :deadline")

, @NamedQuery(name = "Projects.findByExecution", query = "SELECT p FROM Projects p WHERE p.execution = :execution")

, @NamedQuery(name = "Projects.findByCost", query = "SELECT p FROM Projects p WHERE p.cost = :cost")})

public class Projects implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Basic(optional = false)

@Column(name = "project\_id")

private Integer projectId;

@Basic(optional = false)

@NotNull

@Size(min = 1, max = 20)

@Column(name = "name")

private String name;

@Basic(optional = false)

@NotNull

@Column(name = "deadline")

@Temporal(TemporalType.DATE)

private Date deadline;

@Basic(optional = false)

@NotNull

@Column(name = "execution")

private short execution;

@Basic(optional = false)

@NotNull

@Column(name = "cost")

private float cost;

@JoinColumn(name = "worker\_id", referencedColumnName = "worker\_id")

@ManyToOne(optional = false)

private Workers workerId;

public Projects() {

}

public Projects(Integer projectId) {

this.projectId = projectId;

}

public Projects(Integer projectId, String name, Date deadline, short execution, float cost) {

this.projectId = projectId;

this.name = name;

this.deadline = deadline;

this.execution = execution;

this.cost = cost;

}

public Integer getProjectId() {

return projectId;

}

public void setProjectId(Integer projectId) {

this.projectId = projectId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Date getDeadline() {

return deadline;

}

public void setDeadline(Date deadline) {

this.deadline = deadline;

}

public short getExecution() {

return execution;

}

public void setExecution(short execution) {

this.execution = execution;

}

public float getCost() {

return cost;

}

public void setCost(float cost) {

this.cost = cost;

}

public Workers getWorkerId() {

return workerId;

}

public void setWorkerId(Workers workerId) {

this.workerId = workerId;

}

@Override

public int hashCode() {

int hash = 0;

hash += (projectId != null ? projectId.hashCode() : 0);

return hash;

}

@Override

public boolean equals(Object object) {

// TODO: Warning - this method won't work in the case the id fields are not set

if (!(object instanceof Projects)) {

return false;

}

Projects other = (Projects) object;

if ((this.projectId == null && other.projectId != null) || (this.projectId != null && !this.projectId.equals(other.projectId))) {

return false;

}

return true;

}

@Override

public String toString() {

return "Entities.Projects[ projectId=" + projectId + " ]";

}

}

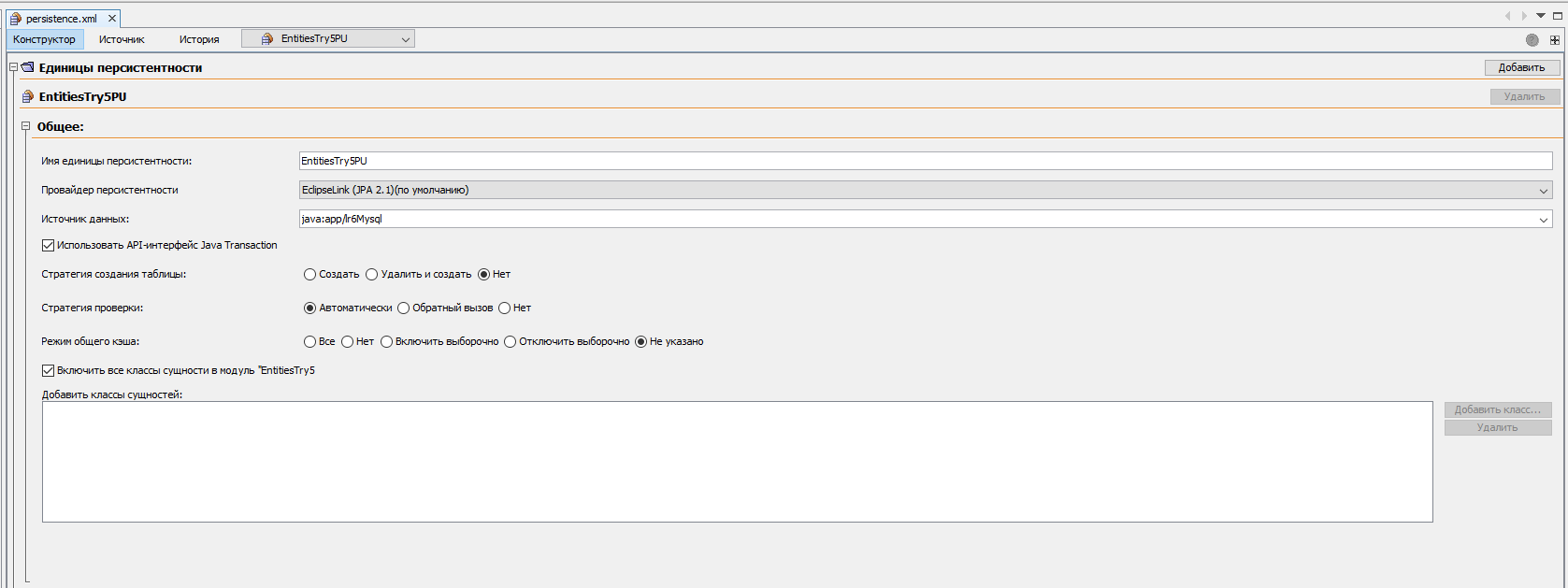


Рисунок 3. Конфигурационный файл persistance.xml.

Результаты запуска и тестирования сгенерированного приложения.

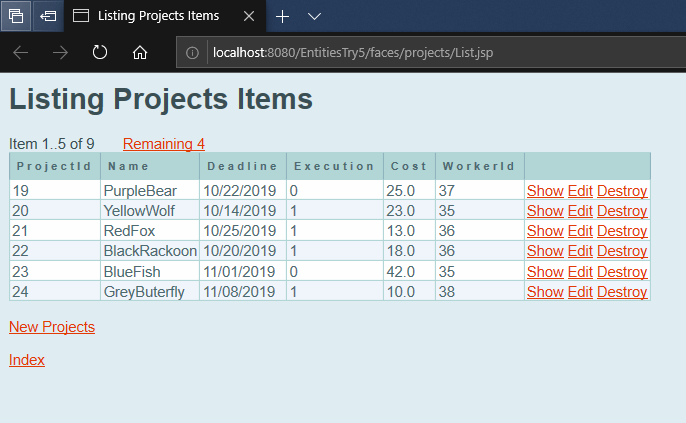


Рисунок 4. Вывод всех записей таблицы базы данных.

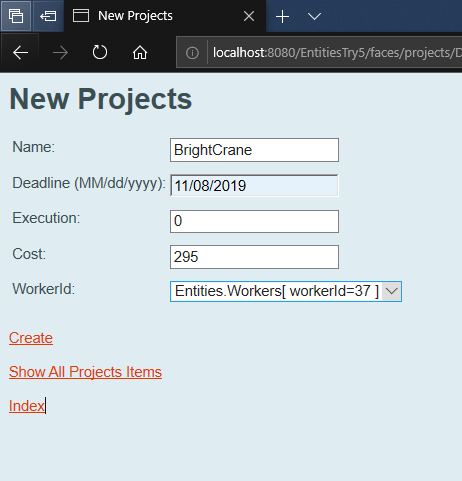


Рисунок 5. Добавление записи в таблицу базы данных.

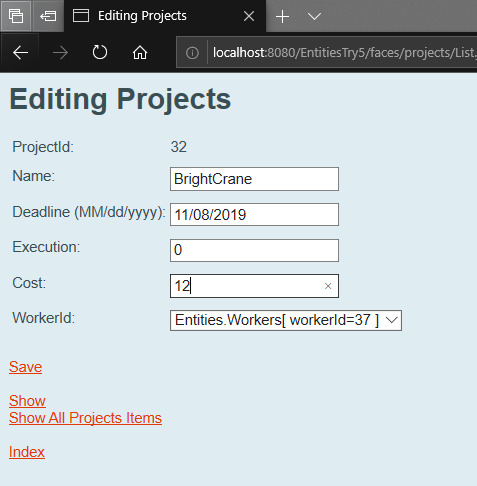


Рисунок 6. Редактирование записи в таблице базы данных.

На втором этапе лабараторной работы было разработано web-приложение, в котором классы сущностей, сессионные компоненты, контроллеры и представления создавались вручную. В процессе тестирования разрабатываемого приложения база данных была удалена, поэтому данные во втором приложении будут расходится с данными из приложения созданного автоматически. Однако, связные таблицы и их структура остались теми же.

В ходе разработке веб-приложения были созданы классы сущностей базы данных с полями повторяющими столбцы соответствующих таблиц. Были использованы аннотации класса javax.persistance: @Table (для отметки таблицы), @Entity (для применения свойств сущности), @Id (для определения первичного ключа таблицы), @GeneratedValue (для определения свойства auto\_increment у первичного ключа), @Column (для определения столбца таблицы), @JoinColumn (для определения столбца, который связывает дву таблицы), @ManyToOne (для определения кратности связи между таблицами).

После создания классов сущностей были созданы сессионные компоненты, реализующие локальные интерфейсы. В них были объявлены поля EntityManager с аннотацией @PersistanceContext, с помощью которых были использованы методы доступа к данным и созданы запросы на языке JPQL.

Далее были созданы контроллеры, в которых были инъецированы сессионные компоненты с помошью аннотации @EJB и веб-страницы для отображения пользовательского интерфейса. Их код опущен, чтобы не загромождать отчет. Со всеми исходниками можно ознакомится в удаленномрепозитории: https://github.com/seeyuointhespring/JavaLabs/tree/master/LR3Try1.

Ниже приведены коды разработанных классов, страниц, конфигурационных файлов и скриншоты выполнения веб-приложения.

**КЛАСЫ СУЩНОСТЕЙ**

**Worker.java:**

package Entities;

import java.io.Serializable;

import java.util.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.Temporal;

@Entity

@Table(name="workers")

public class Worker implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name="fio")

private String fio;

@Column(name="salary")

private float salary;

@Column(name="startdate")

@Temporal(javax.persistence.TemporalType.DATE)

private Date startdate;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFio() {

return fio;

}

public void setFio(String fio) {

this.fio = fio;

}

public float getSalary() {

return salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

public Date getStartdate() {

return startdate;

}

public void setStartdate(Date startdate) {

this.startdate = startdate;

}

public Worker(){}

public Worker(String fio, float salary, Date startdate){

this.fio=fio;

this.salary=salary;

this.startdate=startdate;

}

}

**Project.java**

package Entities;

import java.io.Serializable;

import java.util.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

import javax.persistence.Temporal;

@Entity

@Table(name="projects")

public class Project implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name="name")

private String name;

@Column(name="deadline")

@Temporal(javax.persistence.TemporalType.DATE)

private Date deadline;

@Column(name="execution")

private boolean execution;

@Column(name="cost")

private float cost;

@JoinColumn(name="worker", referencedColumnName = "id")

@ManyToOne(optional=false)

private Worker worker;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Date getDeadline() {

return deadline;

}

public void setDeadline(Date deadline) {

this.deadline = deadline;

}

public boolean isExecution() {

return execution;

}

public void setExecution(boolean execution) {

this.execution = execution;

}

public float getCost() {

return cost;

}

public void setCost(float cost) {

this.cost = cost;

}

public Worker getWorker() {

return worker;

}

public void setWorker(Worker worker) {

this.worker = worker;

}

}

**КЛАССЫ И ЛОКАЛЬНЫЕИНТЕРФЕЙСЫ СЕССИОННЫХ КОМПОНЕНТОВ**

**SessionBeanWorkerLocal.java:**

package SessionComponents;

import Entities.Worker;

import java.util.List;

import javax.ejb.Local;

@Local

public interface SessionBeanWorkerLocal {

public void add(Worker worker);

public void update(Worker worker);

public void remove(Worker worker);

public List<Worker> getAll();

public Worker getWorkerByID(int id);

}

**SessionBeanProjectLocal.java:**

package SessionComponents;

import Entities.Project;

import java.util.List;

import javax.ejb.Local;

@Local

public interface SessionBeanProjectLocal{

public List<Project> getAll();

public List<Project> getExecuted();

}

**SessionBeanWorker.java:**

package SessionComponents;

import Entities.Worker;

import java.util.List;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Stateless

public class SessionBeanWorker implements SessionBeanWorkerLocal {

@PersistenceContext(unitName = "LR3Try1PU")

private EntityManager entity\_manager;

@Override

public void add(Worker worker) {

entity\_manager.persist(worker);

}

@Override

public void update(Worker worker) {

entity\_manager.merge(worker);

}

@Override

public void remove(Worker worker) {

entity\_manager.remove(entity\_manager.merge(worker));

}

@Override

public List<Worker> getAll() {

List<Worker> workers;

workers = entity\_manager.createQuery("SELECT w FROM Worker w").getResultList();

return workers;

}

@Override

public Worker getWorkerByID(int id) {

return entity\_manager.find(Worker.class, id);

}

}

**SessionBeanProject.java:**

package SessionComponents;

import Entities.Project;

import java.util.List;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Stateless

public class SessionBeanProject implements SessionBeanProjectLocal {

@PersistenceContext(unitName = "LR3Try1PU")

private EntityManager entity\_manager;

@Override

public List<Project> getAll(){

List<Project> projects;

projects = entity\_manager.createQuery("SELECT p FROM Project p").getResultList();

return projects;

}

@Override

public List<Project> getExecuted() {

List<Project> projects;

projects = entity\_manager.createQuery("SELECT p FROM Project p where p.execution=true").getResultList();

return projects;

}

}

**КОНТРОЛЛЕРЫ**

**WorkerServlet.java:**

package Controllers;

import Entities.Worker;

import SessionComponents.SessionBeanWorkerLocal;

import java.io.IOException;

import java.util.Date;

import java.util.List;

import javax.ejb.EJB;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class WorkerServlet extends HttpServlet {

@EJB

SessionBeanWorkerLocal W;

private Worker worker = new Worker();

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

List<Worker> workers = W.getAll();

request.setAttribute("workers", workers);

RequestDispatcher dispatcher = request.getRequestDispatcher("/showWorkers.jsp");

dispatcher.forward(request, response);

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String param = request.getParameter("param");

RequestDispatcher dispatcher;

switch (param){

case ("Update"):

worker.setFio(request.getParameter("paramfio"));

worker.setSalary(Float.parseFloat(request.getParameter("paramsalary")));

W.update(worker);

dispatcher = request.getRequestDispatcher("/index.jsp");

dispatcher.forward(request, response);

break;

case ("Create"):

String fio = request.getParameter("pfio");

Date startdate = new Date();

float salary = Float.parseFloat(request.getParameter("psalary"));

worker = new Worker(fio,salary,startdate);

W.add(worker);

dispatcher = request.getRequestDispatcher("/index.jsp");

dispatcher.forward(request, response);

break;

default:

dispatcher=request.getRequestDispatcher("index.jsp");

dispatcher.forward(request,response);

}

}

@Override

public String getServletInfo() {

return "Short description";

}

}

**ProjectServlet.java:**

package Controllers;

import Entities.Project;

import Entities.Worker;

import SessionComponents.SessionBeanProjectLocal;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Date;

import java.util.List;

import javax.ejb.EJB;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ProjectServlet extends HttpServlet {

@EJB

SessionBeanProjectLocal P;

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String param = request.getParameter("param");

RequestDispatcher dispatcher;

switch (param){

case ("All"):

List<Project> projects = P.getAll();

request.setAttribute("projects", projects);

dispatcher = request.getRequestDispatcher("/showProjects.jsp");

dispatcher.forward(request, response);

break;

case ("Exe"):

List<Project> projects2 = P.getExecuted();

request.setAttribute("projects", projects2);

dispatcher = request.getRequestDispatcher("/showExeProjects.jsp");

dispatcher.forward(request, response);

break;

default:

dispatcher=request.getRequestDispatcher("index.jsp");

dispatcher.forward(request,response);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

}

@Override

public String getServletInfo() {

return "Short description";

}

}

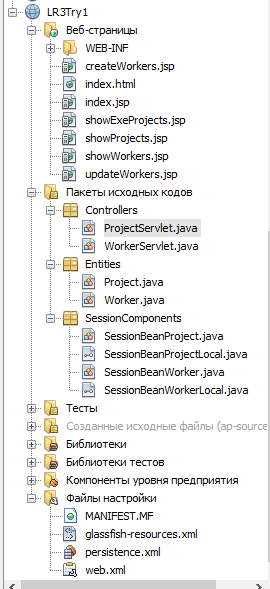


Рисунок 7. Структура веб-приложения.

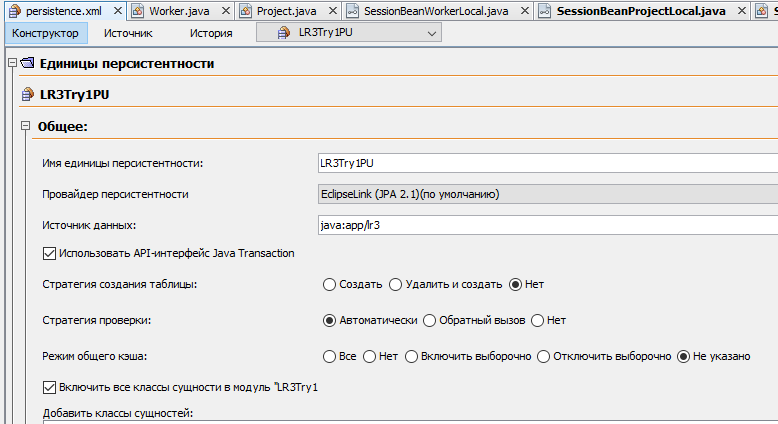
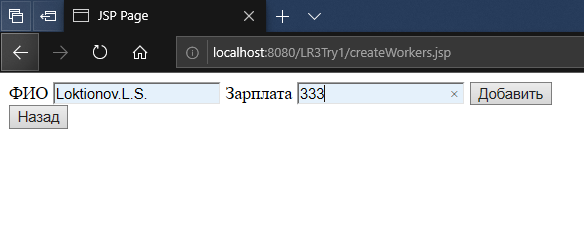
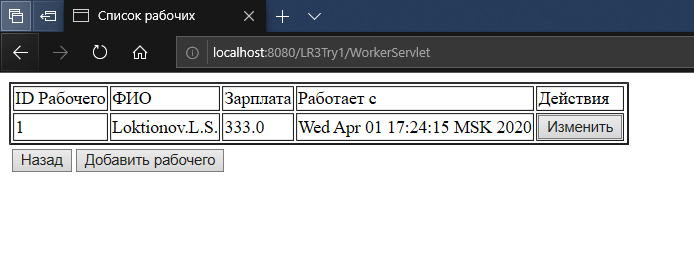


Рисунок 8. Конфигурационный файл persistance.xml

Результаты работы приложения

 Рисунок 9. Добавление записи в таблицу Workers.

Рисунок 10. Отображение добавленной записи.

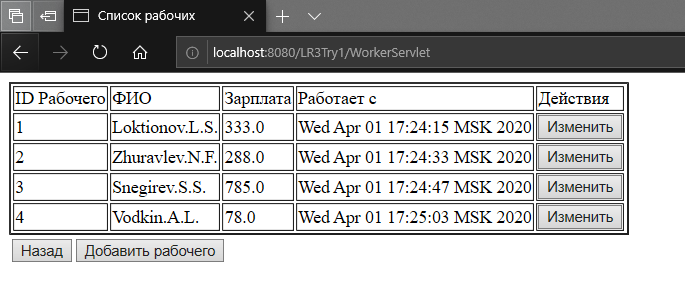


Рисунок 11. Вывод всех записей таблицы Workers.

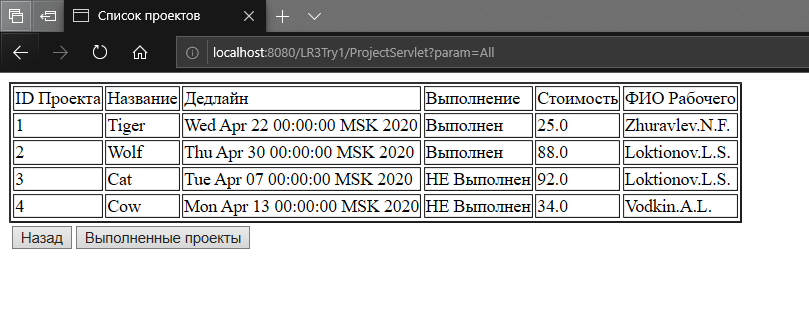


Рисунок 12. Вывод всех записей таблицы Projects.

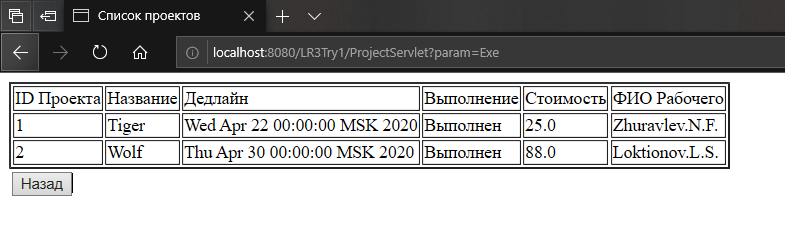


Рисунок 13. Вывод всех записей таблицы Projects где значение в пое выполнение равняется true.

ВЫВОД

В ходе лабораторной работы был освоен процесс создания классов сущностей, в том числе путем обратного проектирования реляционной базы данных средствами NetBeans 5.5 и процесс работы с менеджером сущностей, получены навыки работы со средой разработки.