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 №4

по дисциплине "Распределенные программные системы"

по теме: Распределенные транзакции.

Выполнил:

студент гр. ПРИ-117

Журавлев Н.М.

Принял:

Трифонов Д.А.

Владимир 2020 г.

ЦЕЛЬ РАБОТЫ

Получить сведения о распределенных транзакциях в Java EE-приложениях и закрепить их с помощью экспериментов.

ЗАДАНИЕ

1. Создать модули персистентности для двух баз данных.

2. Создать не менее одного класса сущностей для каждого модуля персистентности вручную или с помощью обратного проектирования базы данных.

3. Создать сессионные фасады для работы с сущностями.

4. Разместить компоненты и сущности для каждой базы данных в отдельные EJBмодули.

5. Разработать сессионный компонент для проведения экспериментов с распределенными транзакциями и доработать тестовое клиентское приложение для вызова его методов. Сессионный компонент должен вызывать методы фасадов для создания или изменения сущностей в обоих базах данных.

6. Провести эксперимент: дать контейнеру закончить транзакцию подтверждением и убедиться, что обновления зафиксированы.

7. Провести эксперимент: закончить транзакцию откатом в сессионном фасаде для первой базы данных и убедиться, что обновления отменены.

8. Провести эксперимент: закончить транзакцию откатом, выбросив системное исключение EJBException в сессионном фасаде для второй базы данных после обновления источника данных, и убедиться, что обновления отменены.

9. Провести эксперимент, аналогичный и.7, но обновление второй базы данных должно выполняться вне контекста транзакции, продемонстрировать результат эксперимента.

10. Провести эксперимент, аналогичный и.8, но обновление первой базы данных должно выполняться в контексте новой транзакции, продемонстрировать результат эксперимента.

ВАРИАНТ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

В качестве индивидуального задания была выбрана структура из двух баз данных. В первой находится таблица с рабочими. Во второй находится таблица с налогами. Вторая база данных должна заполняться и редактироваться в зависимости от первой.

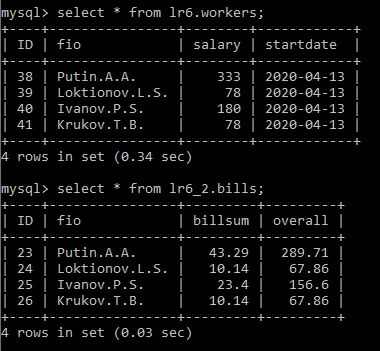


Рисунок 1. Структура баз данных.

ХОД РАБОТЫ

На первом этапе лабараторной работы был отредактирован прект (web-приложение) разработанный в лабараторной работе №3. В проект были добавлены 2 EJB-модуля (для БД с налогами и БД с рабочими). EJB модули содержат классы сущностей, локальные интерфейсы сессионных компонентов, сессионные компоненты, а также собственные модули персистентности.

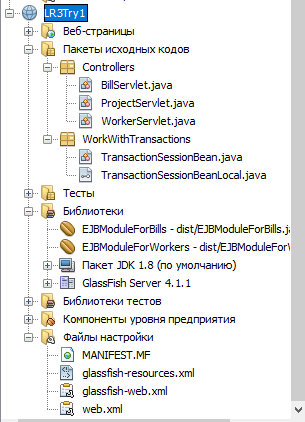


Рисунок 2. Структура web-приложения.

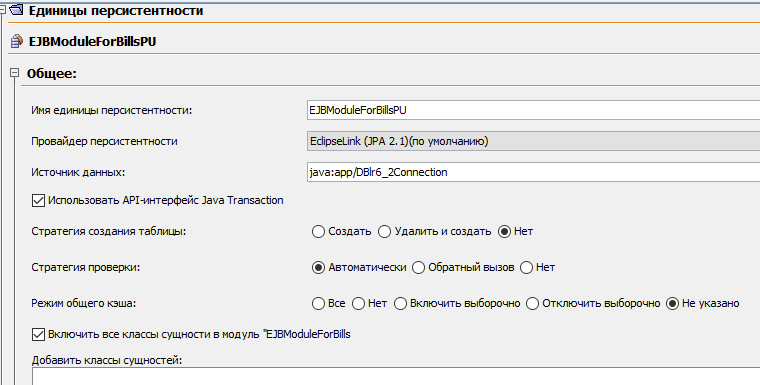


Рисунок 3. Конфигурационный файл persistance.xml. для подключения к БД lr6 (база данных с рабочими)

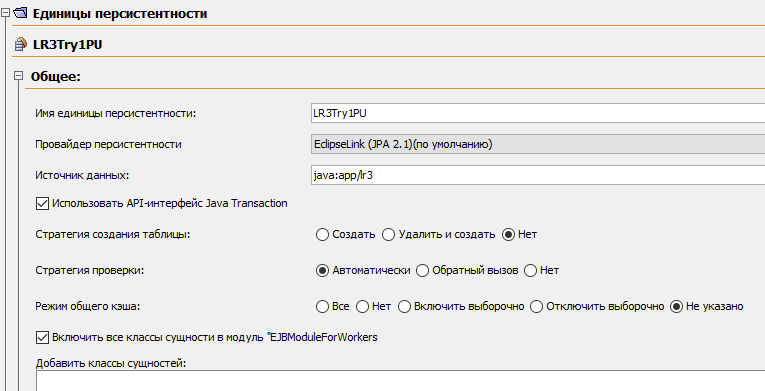


Рисунок 4. Конфигурационный файл persistance.xml. для подключения к БД lr6\_2 (база данных с налогами)

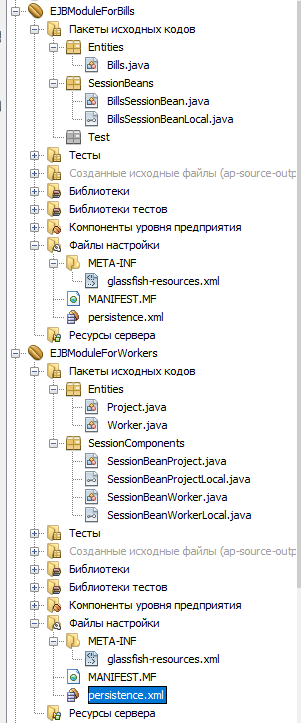


Рисунок 5. Структура EJB-модулей.

На втором этапе лабараторной работы был разработан cессионный компонент TransactionSessionBean для организации транзакций и проведения экспериментов над ними. Были разработаны две распределенные транзакции. Первая выполняет добавление записи в БД с рабочими, а так же исходя из данных о зарплате в этой записи добавляет новую запись в БД с налогами. Вторая транзакция выполняет двойное удаление: записи из БД с рабочими и соответствующей записи из БД с налогами.

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

**КЛАСЫ СУЩНОСТЕЙ**

**EJBModuleForWorkers.Entities.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;

}

}

**EJBModuleForBills.Entities.Bills.java:**

package Entities;

import java.io.Serializable;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="bills")

public class Bills implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(name="fio")

private String fio;

@Column(name="billsum")

private float billsum;

@Column(name="overall")

private float overall;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFio() {

return fio;

}

public void setFio(String fio) {

this.fio = fio;

}

public float getBillsum() {

return billsum;

}

public void setBillsum(float billsum) {

this.billsum = billsum;

}

public float getOverall() {

return overall;

}

public void setOverall(float overall) {

this.overall = overall;

}

public Bills(){}

public Bills(String fio, float billsum, float overall){

this.fio=fio;

this.billsum=billsum;

this.overall=overall;

}

}

**КЛАССЫ И ЛОКАЛЬНЫЕ ИНТЕРФЕЙСЫ СЕССИОННЫХ КОМПОНЕНТОВ**

**EJBModuleForWorkers.SessionComponents.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(int id);

public List<Worker> getAll();

public Worker getWorkerByID(int id);

public void closeEntityManager();

}

**EJBModuleForWorkers.SessionComponents.SessionBeanWorker.java:**

package SessionComponents;

import Entities.Worker;

import java.util.List;

import javax.annotation.Resource;

import javax.ejb.SessionContext;

import javax.ejb.Stateless;

import javax.ejb.TransactionAttribute;

import static javax.ejb.TransactionAttributeType.REQUIRES\_NEW;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

@Stateless

@TransactionAttribute(REQUIRES\_NEW)

public class SessionBeanWorker implements SessionBeanWorkerLocal {

@PersistenceContext(unitName = "LR3Try1PU")

private EntityManager entity\_manager;

@Resource

private SessionContext ctx;

@Override

public void add(Worker worker) {

entity\_manager.persist(worker);

//ctx.setRollbackOnly();

}

@Override

public void update(Worker worker) {

entity\_manager.merge(worker);

}

@Override

public void remove(int id) {

Query query;

query = entity\_manager.createQuery("DELETE FROM Worker w WHERE w.id = :first");

query.setParameter("first", id);

int rowCount = query.executeUpdate();

}

@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);

}

@Override

public void closeEntityManager() {

this.entity\_manager.close();

}

}

**EJBModuleForBills.SessionComponents.** **BillsSessionBeanLocal.java:**

package SessionBeans;

import Entities.Bills;

import java.util.List;

import javax.ejb.Local;

@Local

public interface BillsSessionBeanLocal {

public void add(Bills bill);

public void update(Bills bill);

public List<Bills> getAll();

public void remove(Bills bill);

public Bills getBillByID(int id);

public Bills getBillByFamily(String fio);

}

**EJBModuleForBills.SessionComponents.** **BillsSessionBean.java:**

package SessionBeans;

import Entities.Bills;

import java.sql.SQLException;

import java.util.List;

import javax.annotation.Resource;

import javax.ejb.EJBException;

import javax.ejb.SessionContext;

import javax.ejb.Stateless;

import javax.ejb.TransactionAttribute;

import static javax.ejb.TransactionAttributeType.NOT\_SUPPORTED;

import static javax.ejb.TransactionAttributeType.REQUIRES\_NEW;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

@Stateless

@TransactionAttribute(REQUIRES\_NEW)

public class BillsSessionBean implements BillsSessionBeanLocal {

@PersistenceContext(unitName = "EJBModuleForBillsPU")

private EntityManager entity\_manager;

@Resource

private SessionContext context;

@Override

public void add(Bills bill) {

try{

entity\_manager.persist(bill);

//throw new Exception();

}catch(Exception e){

//entity\_manager.getTransaction().rollback();

System.out.println("Ошибка при добавлении в таблицу налогов:");

System.out.println(e.getMessage());

}

}

@Override

public void update(Bills bill) {

entity\_manager.merge(bill);

}

@Override

public List<Bills> getAll() {

List<Bills> bills;

bills = entity\_manager.createQuery("SELECT b FROM Bills b").getResultList();

return bills;

}

@Override

public void remove(Bills bill) {

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

}

@Override

public Bills getBillByID(int id) {

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

}

@Override

public Bills getBillByFamily(String fio) {

Bills bill = new Bills();

Query query = entity\_manager.createQuery("SELECT b FROM Bills b WHERE b.fio = :first");

query.setParameter("first", fio);

bill = (Bills)query.getSingleResult();

return bill;

}

}

**LR3Try1. WorkWithTransactions.TransactionSessionBeanLocal.java:**

package WorkWithTransactions;

import Entities.Worker;

import javax.ejb.Local;

@Local

public interface TransactionSessionBeanLocal {

public void doubleAdditing(Worker worker);

public void doubleRemoving(int id);

}

**LR3Try1.WorkWithTransactions.TransactionSessionBean.java:**

package WorkWithTransactions;

import Entities.Bills;

import Entities.Worker;

import SessionBeans.BillsSessionBeanLocal;

import SessionComponents.SessionBeanWorkerLocal;

import javax.annotation.Resource;

import javax.ejb.EJB;

import javax.ejb.EJBException;

import javax.ejb.SessionContext;

import javax.ejb.Stateful;

import javax.ejb.TransactionManagement;

import javax.ejb.TransactionManagementType;

import javax.transaction.HeuristicMixedException;

import javax.transaction.HeuristicRollbackException;

import javax.transaction.NotSupportedException;

import javax.transaction.RollbackException;

import javax.transaction.SystemException;

@Stateful

@TransactionManagement(TransactionManagementType.BEAN)

public class TransactionSessionBean implements TransactionSessionBeanLocal {

@EJB

SessionBeanWorkerLocal W;

@EJB

BillsSessionBeanLocal B;

Bills bill = new Bills();

Worker worker = new Worker();

@Resource

javax.transaction.UserTransaction transaction;

@Resource

private SessionContext ctx;

@Override

public void doubleAdditing(Worker worker) {

try{

transaction.begin();

W.add(worker);

bill.setFio(worker.getFio());

bill.setBillsum((float) (worker.getSalary()\*0.13));

bill.setOverall((float) (worker.getSalary()-worker.getSalary()\*0.13));

B.add(bill);

transaction.commit();

}catch(IllegalStateException | SecurityException | EJBException | NotSupportedException | SystemException | RollbackException | HeuristicMixedException | HeuristicRollbackException e){

System.out.println("Произошла ошибка в транзакции добавления: ");

System.out.println(e.getMessage());

System.out.println(e.getClass());

}

}

@Override

public void doubleRemoving(int id) {

try{

transaction.begin();

Worker workerToDelete = W.getWorkerByID(id);

W.remove(id);

bill = B.getBillByFamily(workerToDelete.getFio());

B.remove(bill);

transaction.commit();

}

catch(IllegalStateException | SecurityException | HeuristicMixedException | HeuristicRollbackException | NotSupportedException | RollbackException | SystemException e){

System.out.println("Произошла ошибка в транзакции удаления: ");

System.out.println(e.getMessage());

System.out.println(e.getClass());

}

}

}

**КОНТРОЛЛЕРЫ**

**LR3Try1.Controllers.WorkerServletjava:**

package Controllers;

import Entities.Worker;

import SessionComponents.SessionBeanWorkerLocal;

import WorkWithTransactions.TransactionSessionBeanLocal;

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();

@EJB

TransactionSessionBeanLocal T;

@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);

T.doubleAdditing(worker);

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

dispatcher.forward(request, response);

break;

case ("Delete"):

int id = Integer.parseInt(request.getParameter("workerid"));

//W.remove(id);

T.doubleRemoving(id);

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

dispatcher.forward(request,response);

default:

}

}

@Override

public String getServletInfo() {

return "Short description";

}

}

**LR3Try1.Controllers.BillServletjava:**

package Controllers;

import Entities.Bills;

import SessionBeans.BillsSessionBeanLocal;

import java.io.IOException;

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 BillServlet extends HttpServlet {

@EJB

BillsSessionBeanLocal B;

private Bills bill = new Bills();

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

List<Bills> bills = B.getAll();

request.setAttribute("bills", bills);

RequestDispatcher dispatcher = request.getRequestDispatcher("/showBills.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"):

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

bill.setBillsum(Float.parseFloat(request.getParameter("parambillsum")));

bill.setOverall(Float.parseFloat(request.getParameter("paramoverall")));

B.update(bill);

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

dispatcher.forward(request, response);

break;

case ("Create"):

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

float billsum = Float.parseFloat(request.getParameter("pbillsum"));

float overall = Float.parseFloat(request.getParameter("poverall"));

bill = new Bills(fio,billsum,overall);

B.add(bill);

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

dispatcher.forward(request, response);

break;

case ("Delete"):

int id = Integer.parseInt(request.getParameter("billid"));

bill = B.getBillByID(id);

B.remove(bill);

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

dispatcher.forward(request,response);

default:

}

}

@Override

public String getServletInfo() {

return "Short description";

}

}

**ВЫПОЛНЕНИЕ ЭКСПЕРИМЕНТОВ**

**Эксперимент 1**

1. Провести эксперимент: дать контейнеру закончить транзакцию подтверждением и убедиться, что обновления зафиксированы**.**

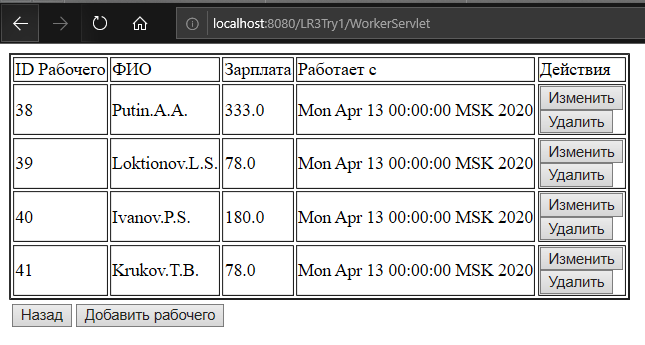


Рисунок 6. Таблица в базе данных с рабочими до добавления.

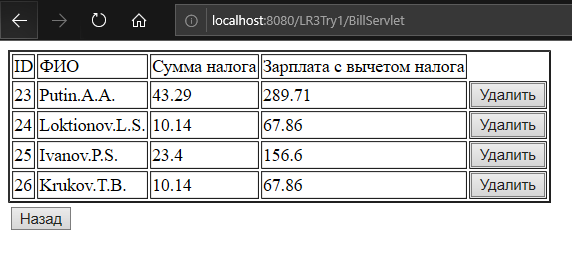


Рисунок 7. Таблица в базе данных с налогами до добавления.

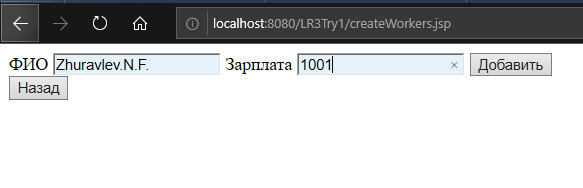


Рисунок 8. Добавление новой записи.

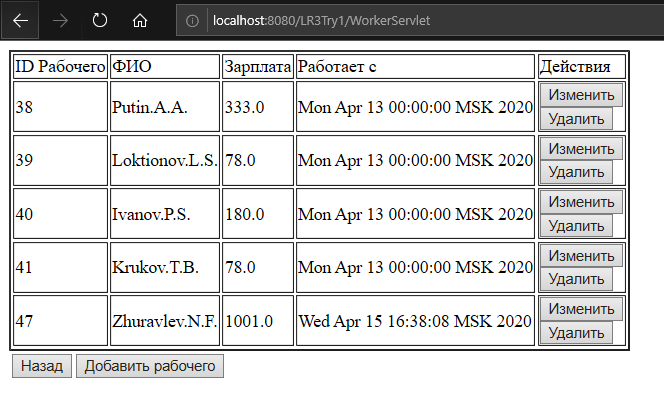


Рисунок 9. Таблица в базе данных с рабочими после добавления.

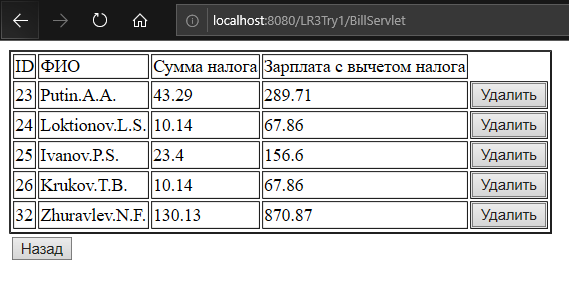


Рисунок 10. Таблица в базе данных с налогами полсе добавления.

**Эксперимент 2**

2. Провести эксперимент: закончить транзакцию откатом в сессионном фасаде для первой базы данных и убедиться, что обновления отменены.

Для проведения эксперимента был изменен сессионный компонент. Для проведения отката транзакции был добавлен метод объекта класса сессионного контекста setRollbackOnly.

Ниже приведен измененный код сессионного компонента SessionBeanWorker.java.

@Stateless

@TransactionAttribute(REQUIRES\_NEW)

public class SessionBeanWorker implements SessionBeanWorkerLocal {

@PersistenceContext(unitName = "LR3Try1PU")

private EntityManager entity\_manager;

@Resource

private SessionContext ctx;

@Override

public void add(Worker worker) {

entity\_manager.persist(worker);

ctx.setRollbackOnly();

}

…

}

В результате запуска приложения и попытки добавления записи в таблицу рабочих, произошел откат транзакция, изменения были отменены и таблица базы данных осталась неизменной.

**Эксперимент 3**

3. Провести эксперимент: закончить транзакцию откатом, выбросив системное исключение EJBException в сессионном фасаде для второй базы данных после обновления источника данных, и убедиться, что обновления отменены.

Для проведения эксперимента был изменен сессионный компонент. Для проведения отката транзакции был добавлен метод объекта класса сессионного контекста setRollbackOnly в оператор catch при поимке исключения.

Ниже приведен измененный код сессионного компонента BillsSessionBean.java.

@Stateless

@TransactionAttribute(REQUIRES\_NEW)

public class BillsSessionBean implements BillsSessionBeanLocal {

@PersistenceContext(unitName = "EJBModuleForBillsPU")

private EntityManager entity\_manager;

@Resource

private SessionContext context;

@Override

public void add(Bills bill) {

try{

entity\_manager.persist(bill);

throw new EJBException();

}catch(Exception e){

context.setRollbackOnly();

System.out.println("Ошибка при добавлении в таблицу налогов:");

System.out.println(e.getMessage());

System.out.println(e.getClass());

}

}

…

}

В результате запуска приложения и попытки добавления записи в таблицу рабочих и налогов, произошел выброс EJBException и описанный откат транзакции, изменения были отменены и таблица базы данных осталась неизменной.

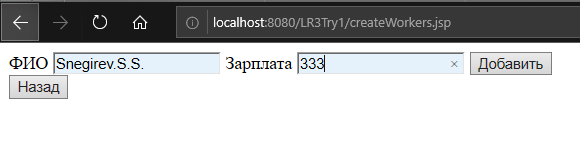


Рисунок 11. Попытка добавление новой записи.

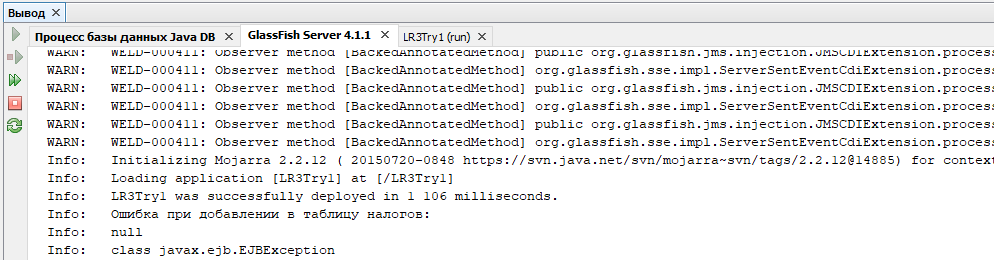


Рисунок 12. Полученное исключение.

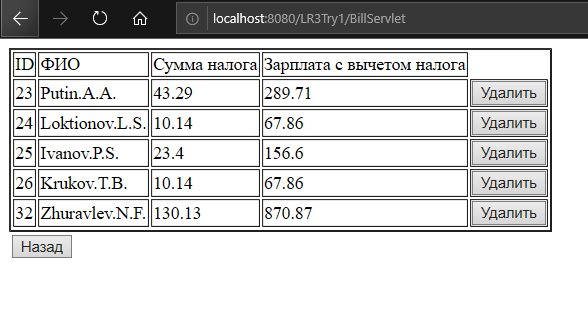


Рисунок 13. Таблица базы данных осталасть без изменений.

**Эксперимент 4**

Эксперименты 4 и 5 проводились над транзакцие удаления TransactionSessionBean.doubleRemoving

4 .Провести эксперимент, аналогичный эксперименту 2, но обновление второй базы данных должно выполняться вне контекста транзакции, продемонстрировать результат эксперимента.

Для проведения эксперимента был изменен сессионный компонент. Для проведения отката удаления вне контекста транзакции были отредактированы методы класса SessionBeanWorker для удаления, а именно были дабавлены аннотации @TransactionAttribute со значением NOT\_SUPPORTED, а также метод executeUpdate был заменен так как он не может использоваться вне контекста транзакции.

Ниже приведен измененный код сессионного компонента BillsSessionBean.java.

public class SessionBeanWorker implements SessionBeanWorkerLocal {

…

@Override

@TransactionAttribute(NOT\_SUPPORTED)

public void remove(int id) {

Worker worker = getWorkerByID(id);

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

//Query query;

//query = entity\_manager.createQuery("DELETE FROM Worker w WHERE w.id = :first");

//query.setParameter("first", id);

//int rowCount = query.executeUpdate();

ctx.setRollbackOnly();

}

@Override

@TransactionAttribute(NOT\_SUPPORTED)

public Worker getWorkerByID(int id) {

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

}…

}

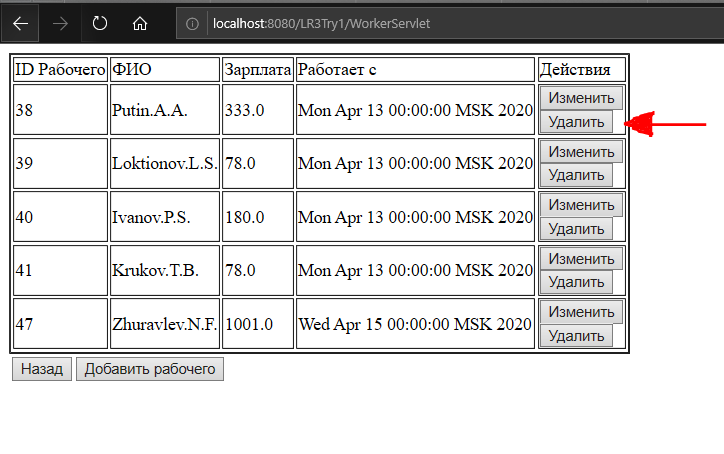
В результате запуска приложения и попытки удаления записи в таблице рабочих, произошел описанный откат транзакции, изменения были отменены и таблица базы данных осталась неизменной.

**Эксперимент 5**

5. Провести эксперимент, аналогичный и.3, но обновление первой базы данных должно выполняться в контексте новой транзакции, продемонстрировать результат эксперимента.

Эксперимент был проведен в ранее проведенных экспериментах так как классы сессионных компонентов с методами изменения данных были аннотированы @TransactionAttribute со значением REQUIRED\_NEW и выполнялись в контексте транзакции.

В качестве примера приведен процесс удаления данных из двух баз данных (операции выполняются в контексте транзакции)

 Рисунок 14. Удаление записи из таблицы рабочих.

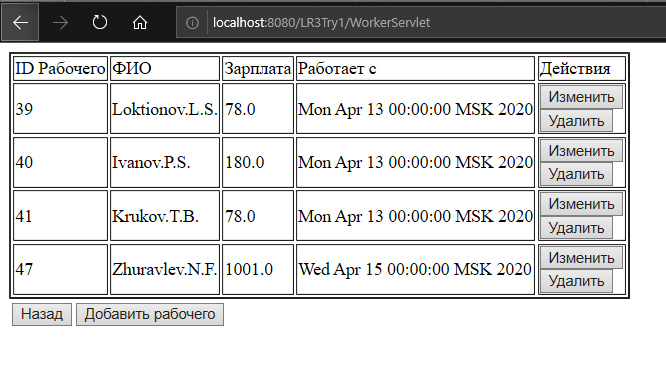


Рисунок 15. Подтверждение удаления записи из таблицы рабочих (первая БД).

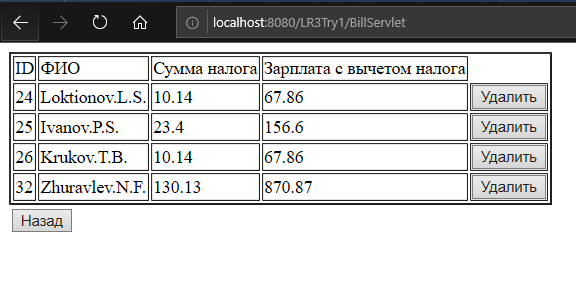


Рисунок 16. Подтверждение удаления записи из таблицы налогов (вторая БД).

Исходный код проекта приведен здесь: https://github.com/seeyuointhespring/JavaLabs/tree/master/LR4\_Transactions

Код находится в конечном состоянии после проведения экспериментов, все изменения, произведенные в коде в результате выполнения экспериментов закомментированы.

Приложение было реализовано средствами IDE NetBeans 8.2 и развернуто на сервере веб-приложений GlassFish 4.1.1

ВЫВОД

В ходе лабораторной работы были получены сведения о распределенных транзакциях в JavaEE-приложениях и закреплены с помощью экспериментов.