Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение наукии высшего образования**

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

**Институт информационных технологий и радиотехники**

**(ИИТР)**

Кафедра информационных систем и программной инженерии

**Лабораторная работа № 04**

**по дисциплине**

**«Распределенные программные системы»**

Выполнил:

ст. гр. ПРИ-117

Хлызова В.Г.

Принял:

Трифонов Д.А.

Владимир, 2020

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 EJB-модули.
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.

ВЫПОЛНЕНИЕ РАБОТЫ

Создаём два ejb-модуля, в которых будут располагатся классы сущностей и сессионные компоненты для работы с базами данных. (Рисунки 1 и 3) В каждом модуле создаём единицу персистентности и источники данных. (Рисунки 2 и 4) Далее, проверяем возможность выполнения транзакций в окне свойств источника данных. (Рисунок 5)

Подключаем ejb-модули к веб-приложению(контроллеры и jsp-страницы взяты из прошлой лабораторной работы, контролеры и страницы для работы с новой бд созданы аналогично) Проверяем работоспособность приложения. Структура веб-приложения представлена на рисунке 6.

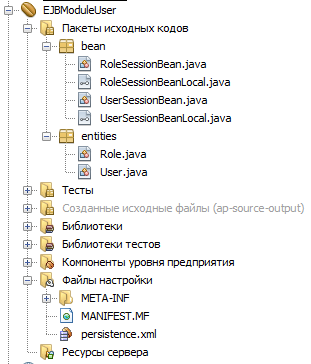


Рисунок 1. Содержимое ejb-модуля для работы с первой бд.

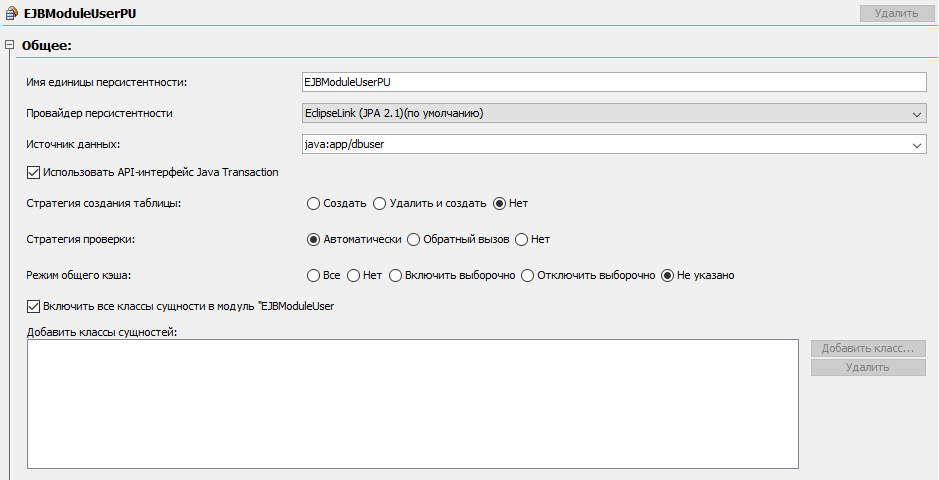


Рисунок 2. Модуль персистентности.

В этом модуле используются классы сущностей и сессионные компоненты из предыдущей лабораторной работы.

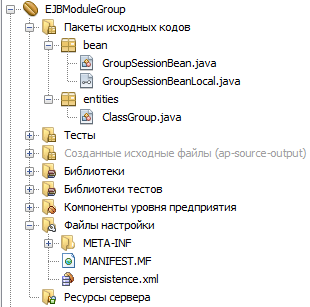


Рисунок 3. Содержимое ejb-модуля для работы со второй бд.

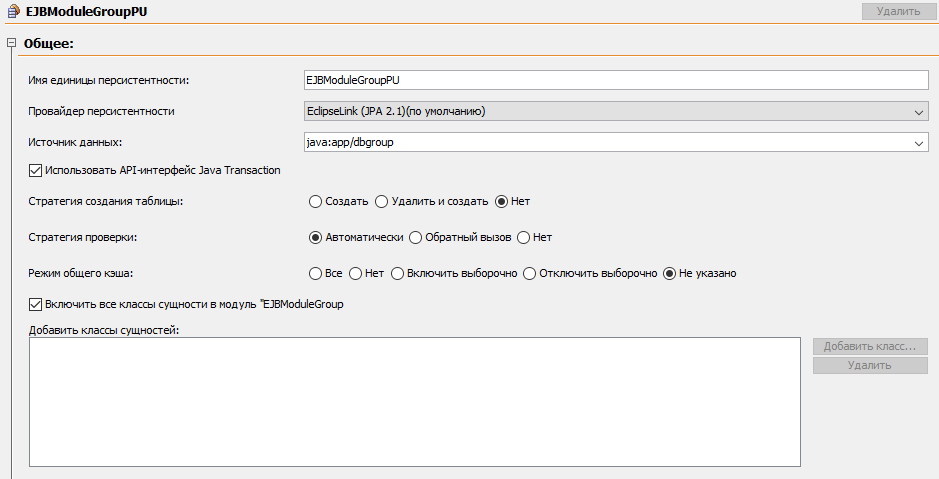


Рисунок 4. Модуль персистентности.

Класс сущности для второй бд:

package entities;

import java.io.Serializable;

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.Table;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

@Entity

@Table(name = "ClassGroup")

public class ClassGroup implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Basic(optional = false)

@Column(name = "id")

private Integer id;

@Basic(optional = false)

@NotNull

@Size(min = 1, max = 20)

@Column(name = "name")

private String name;

@Basic(optional = false)

@NotNull

@Size(min = 1, max = 50)

@Column(name = "description")

private String description;

@Basic(optional = false)

@NotNull

@Column(name = "numberOfStudents")

private int numberOfStudents;

public ClassGroup() {

}

public ClassGroup(Integer id) {

this.id = id;

}

public ClassGroup(Integer id, String name, String description, int numberOfStudents) {

this.id = id;

this.name = name;

this.description = description;

this.numberOfStudents = numberOfStudents;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public int getNumberOfStudents() {

return numberOfStudents;

}

public void setNumberOfStudents(int numberOfStudents) {

this.numberOfStudents = numberOfStudents;

}

}

Сессионный компоненты для работы со второй бд:

package bean;

import entities.ClassGroup;

import java.util.List;

import javax.annotation.Resource;

import javax.ejb.SessionContext;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Stateless

public class GroupSessionBean implements GroupSessionBeanLocal {

@PersistenceContext(unitName = "EJBModuleGroupPU")

private EntityManager em;

@Override

public List<ClassGroup> findAll() {

List <ClassGroup> groupList

=em.createQuery("SELECT g FROM ClassGroup g").getResultList();

return groupList;

}

@Resource

private SessionContext context;

@Override

public void create(ClassGroup g) {

em.persist(g);

if (g.getNumberOfStudents()>100) context.setRollbackOnly();

}

@Override

public void edit(ClassGroup g) {

em.merge(g);

}

@Override

public void remove(ClassGroup g) {

em.remove(em.merge(g));

}

@Override

public ClassGroup find(int id) {

return em.find(ClassGroup.class, id);

}

}

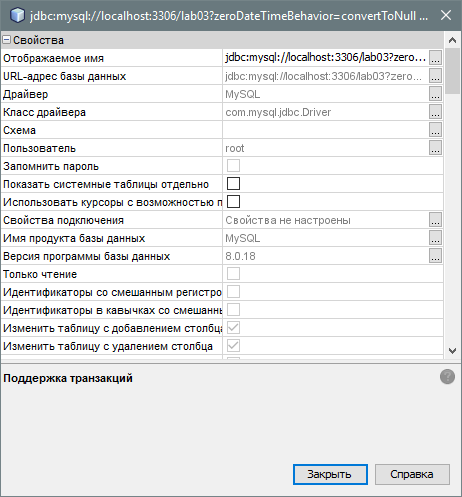


Рисунок 5. Окно свойств источника данных.

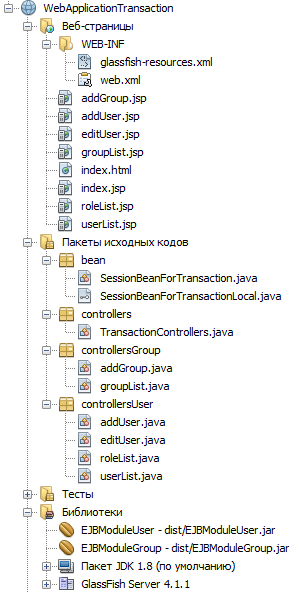


Рисунок 6. Структура веб приложения.

Разрабатываем сессионный компонент для проведения экспериментов с распределенными транзакциями. Во время работы приложения, пользователи и группы накапливаются в списки на добавление пользователя, на добавление группы и на изменение пользователя. После нажатия на кнопку происходит транзакция в определенную бд.

Локальный интерфейс компонента:

package bean;

import entities.ClassGroup;

import entities.User;

import javax.ejb.Local;

@Local

public interface SessionBeanForTransactionLocal {

//метод для транзакции в бд пользователей

public void addUser();

//метод для транзакции в бд групп

public void addGroup();

public void addGroupList(ClassGroup g);

public void addUserList(User u);

public void addUserEditList(User u);

}

Сессионный компонент:

package bean;

import entities.ClassGroup;

import entities.User;

import java.util.ArrayList;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.annotation.Resource;

import javax.ejb.EJB;

import javax.ejb.Singleton;

import javax.ejb.TransactionManagement;

import javax.ejb.TransactionManagementType;

@Singleton

@TransactionManagement(TransactionManagementType.BEAN)

public class SessionBeanForTransaction implements SessionBeanForTransactionLocal {

List<User> userList = new ArrayList<>();

List<User> userEditList = new ArrayList<>();

List<ClassGroup> groupList = new ArrayList<>();

@Override

public void addUserList(User u) {

userList.add(u);

}

@Override

public void addGroupList(ClassGroup g) {

groupList.add(g);

}

@Override

public void addUserEditList(User u) {

userEditList.add(u);

}

@Resource javax.transaction.UserTransaction ut;

@EJB

GroupSessionBeanLocal groupSBL;

@EJB

UserSessionBeanLocal userSBL;

@Override

public void addUser(){

try {

//начать транзакцию

ut.begin();

//добавить записи в бд

for(User u: userList){

userSBL.create(u);

}

//редактировать записи в бд

for(User u: userEditList){

userSBL.edit(u);

}

//зафиксировать изменения

ut.commit();

}catch (Exception ex) {

Logger.getLogger(SessionBeanForTransaction.class.getName()).log(Level.SEVERE, null, ex);

}finally{

userList=new ArrayList<>();

userEditList = new ArrayList<>();

}

}

@Override

public void addGroup(){

try {

//начать транзакцию

ut.begin();

//добавить записи в бд

for (ClassGroup g: groupList){

groupSBL.create(g);

}

//зафиксировать изменения

ut.commit();

}catch (Exception ex) {

Logger.getLogger(SessionBeanForTransaction.class.getName()).log(Level.SEVERE, null, ex);

}finally{

groupList = new ArrayList<>();

}

}

}

Проводим эксперимент 1: добавляем несколько новых пользователей и проверяем, что обновления зафиксированы.

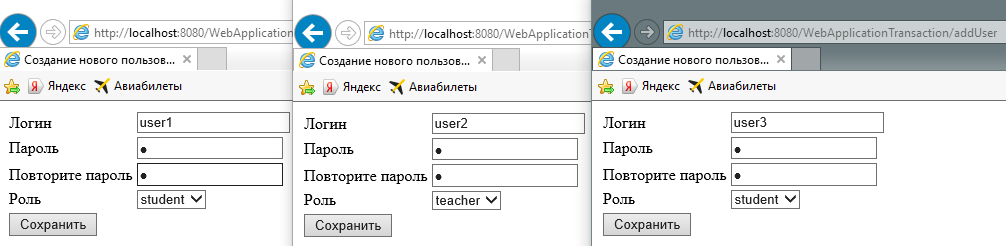


Рисунок 7. Добавляем несколько пользователей.

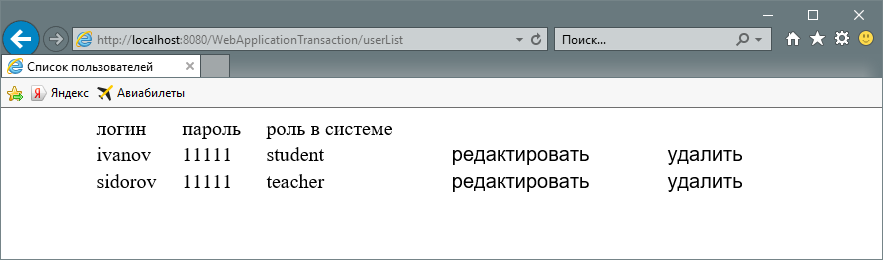


Рисунок 8. Список пользователей до выполнения транзакции.

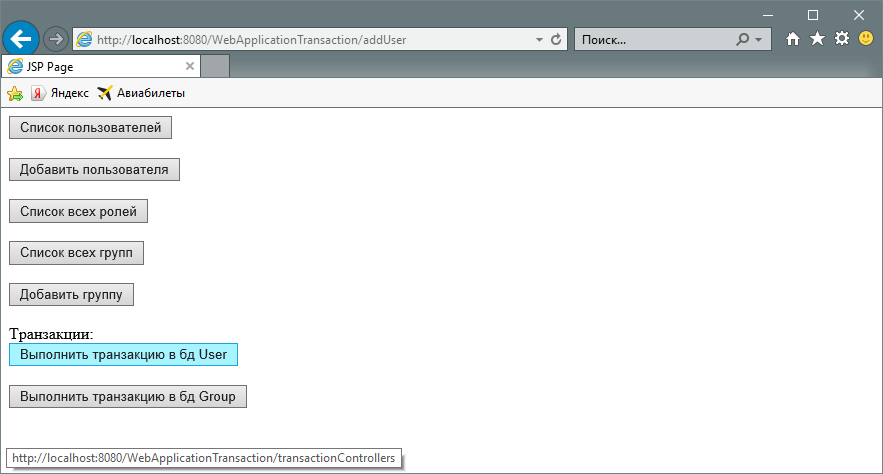


Рисунок 9. Нажимаем кнопку для выполнения транзакции.

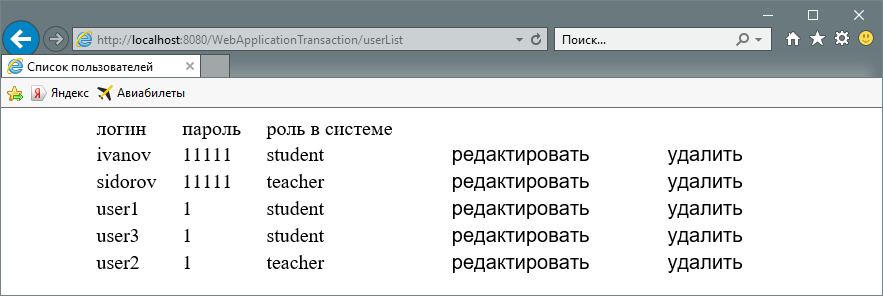


Рисунок 10. Изменения зафиксированы.

Проводим эксперимент 2: заканчиваем транзакцию откатом в сессионном фасаде для базы данных и проверяем, что обновления отменены.

В сессионном компоненте, работающем с бд содержащей записи о группах, используем метод контекста SessionContext.setRollbackOnly(). Если при добавлении записей хоть одна группа будет содержать больше 100 студентов (атрибут NumberOfStudents), то транзакция заканчивается откатом. Часть кода из сессионного компонента:

@Resource

private SessionContext context;

@Override

public void create(ClassGroup g) {

em.persist(g);

if (g.getNumberOfStudents()>100) context.setRollbackOnly();

}

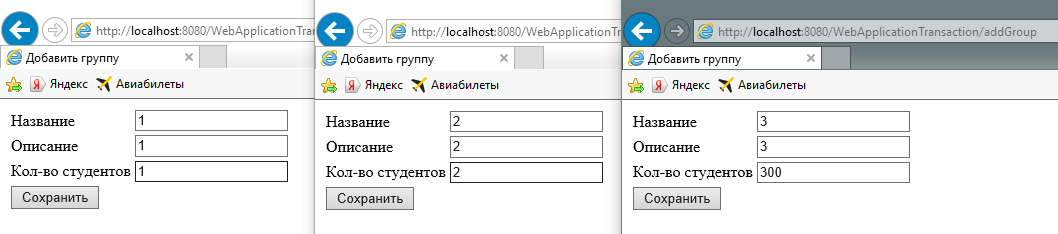


Рисунок 11. Добавляем новые группы.

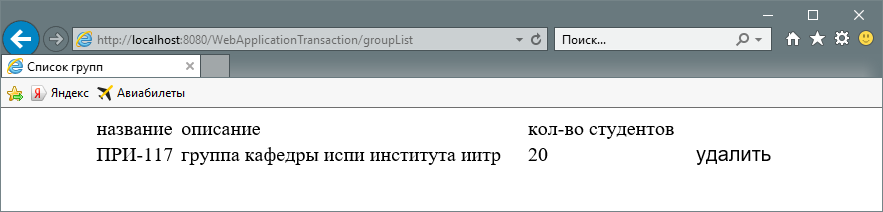


Рисунок 12. Список групп до выполнения транзакции.

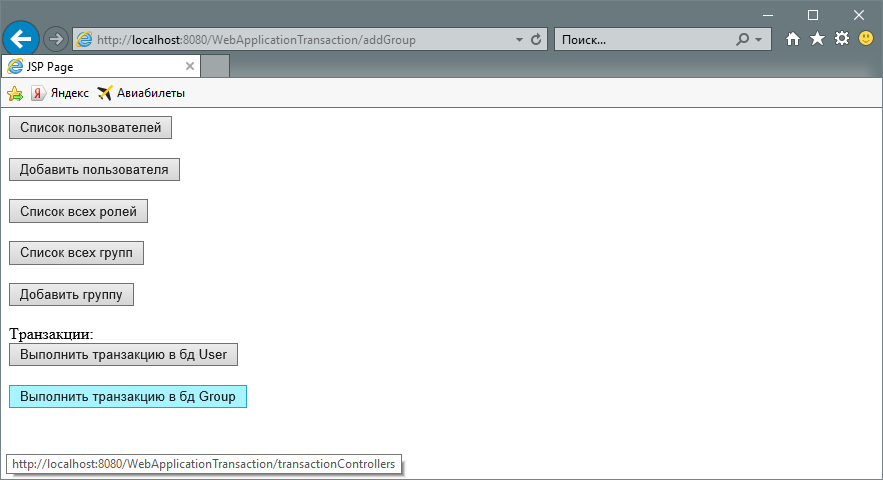


Рисунок 13. Нажимаем кнопку для выполнения транзакции.

После выполнения транзакции список групп не изменился. Транзакция закончилась откатом.

Проводим эксперимент 3: заканчиваем транзакцию откатом, выбросив системное исключение EJBException в сессионном фасаде для базы данных после обновления источника данных, и проверяем, что обновления отменены.

//выбрасываем исключение при добавлении записи

@Override

public void create(User u) {

em.persist(u);

throw new EJBException();

}

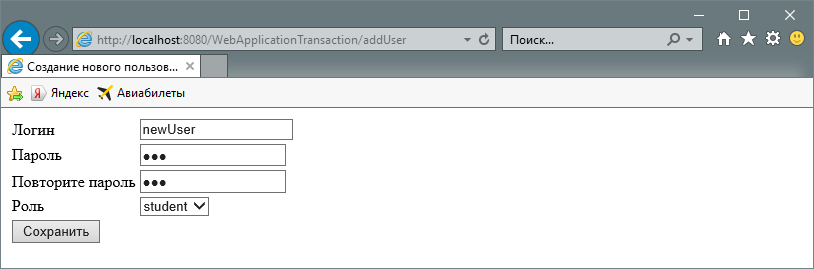


Рисунок 14. Добавляем нового пользователя и выполняем транзакцию.

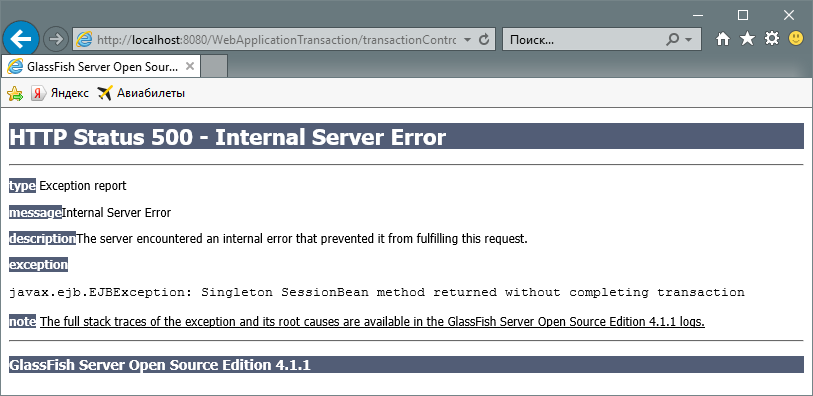


Рисунок 15. Во время транзакции было выброшено исключение.

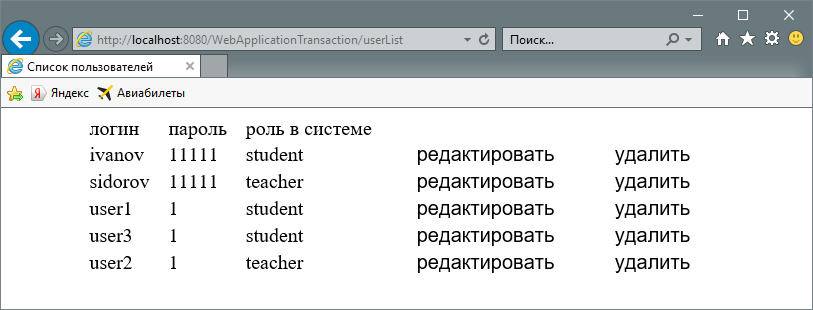


Рисунок 16. Список пользователей не изменился.

Проводим эксперимент 4 аналогичный эксперименту 2: обновление базы данных должно выполняться вне контекста транзакции.

Для этого добавим транзакционный атрибут NOT\_SUPPORTED. Метод будет выполнятся вне транзакции. При выходе из метода транзакция возобновляется.

@Resource

private SessionContext context;

@Override

@TransactionAttribute(NOT\_SUPPORTED)

public void create(ClassGroup g) {

em.persist(g);

if (g.getNumberOfStudents()>100) context.setRollbackOnly();

}

Проводим эксперимент 5, аналогичный эксперименту 3, но обновление базы данных должно выполняться в контексте новой транзакции.

Для этого добавляем транзакционный атрибут со значением REQUIRES\_NEW. Это значит, что метод компонента всегда будет выполняться в контексте автоматически запущенной новой транзакции. Итог эксперимента: в бд будут добавлены все записи, до выброса исключения.

@Override

@TransactionAttribute(REQUIRES\_NEW)

public void create(User u) {

em.persist(u);  
 //исключение выбрасывается, если логин пользователя “error”

if (u.getLogin().equals("error"))throw new EJBException();

}

Deploy приложения был выполнен на сервер Glassfish 4.1.1.   
Ссылка на гит:

ВЫВОД

В ходе выполнения лабораторной работы были освоены и закреплены с помощью экспериментов сведения о распределенных транзакциях в Java EE-приложениях.