Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет компьютерных технологий и управления

Факультет программной инженерии и компьютерной техники

Лабораторная работа

Рефакторинг баз данных и приложений

Группа: P34151

Студент: Дау Конг Туан Ань

Преподаватель: Логинов Иван Павлович

г. Санкт-Петербург, 2024

Оглавление

1. Задачи

В качестве исходного проекта может быть использован курсовой проект по курсу «Базы данных» или иной сопоставимый по объему программный проект

2. Выполнение

- ✓ **Замените транзакции запросов функциями транзакций ORM.**

```
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        @Override
        public Long checkAccount(String username, String password) {
-           begin();
-           List<Account> ret = (List<Account>) entityManager.createQuery("SELECT a FROM Account a WHERE a.username = ?1 AND a.password= ?2")
-                   .setParameter(1, username)
-                   .setParameter(2, password)
-                   .getResultList();
-           commit();
-           if(ret.isEmpty()) return null;
-           return ret.get(0).getId();
+           Account account = entityManager.find(Account.class, username);
+           if(account.getPassword().equals(password)) {
+               return account.getId();
+           }
+           return null;
        }
```

```
-           begin();
-           entityManager.createNativeQuery("insert into account (username, password)  values ( ?, ?)")
-
-                   .setParameter(1, account.getUsername())
-                   .setParameter(2, account.getPassword())
-                   .executeUpdate();
-           commit();
-           List<Account> temp = entityManager.createNativeQuery("select id from account limit 1", Account.class).getResultList();
-           commit();
-           return temp.isEmpty()?null: temp.get(0).getId();
+           entityManager.persist(account);
+           return entityManager.find(Account.class, account.getUsername()).getId();
```

```
    @Override
    public long updatePassword(Account account, String newPassword) {
        begin();
        long ret = entityManager.createQuery("update Account a set a.password = ?1 where a.username = ?2 and a.password = ?3")
                .setParameter(1, newPassword)
                .setParameter(2, account.getUsername())
                .setParameter(3, account.getPassword())
                .executeUpdate();
        commit();

        return ret;
        Account account1 = entityManager.find(account.getClass(), account.getId());
        account1.setPassword(newPassword);
        return entityManager.merge(account1).getId();
    }
```

```java
@Override
public long deleteAccount(Account account) {
    begin();
    long ret = entityManager.createQuery("delete from Account a where a.username = ?1 and a.password = ?2")
            .setParameter(1, account.getUsername())
            .setParameter(2, account.getPassword())
            .executeUpdate();
    commit();

    return ret;
    entityManager.remove(account);
    return 0L;
}
```

Ошибка: Нет ничего плохого в использовании запросов для транзакций, как раньше, но использование встроенных функций ORM помогает оптимизировать его использование, а также минимизировать ошибки в процессе транзакции, делая транзакции более синхронизированными.

- ✓ **Используйте TransactionManager из пула компонентов**

```
15      - @Singleton
    15  + @Stateless
16  16    public class AccountTable implements AccountTableRemote{
17      -     EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("tad");
18      -     EntityManager entityManager = entityManagerFactory.createEntityManager();
    17  +     @PersistenceContext
    18  +     private EntityManager entityManager;
19  19        @Override
20  20        public Long checkAccount(String username, String password) {
21  21            Account account = entityManager.find(Account.class, username);
```

Ошибка: Создание дополнительных TransactionManager каждый раз при создании нового класса приводит к созданию более 18 TransactionManager во время использования приложения, что делает использование памяти более оптимальным. Здесь мы используем @Stateless для разделения отдельных транзакций, делая функции, использующие более одного коммита, более согласованными.

Код изменений

Всю проделанную работу(все изменения в коде) с кратким описанием изменений можно найти в PR на гитхабе проекта: refactoring_app_and_db