|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий (ИТ)

Кафедра Инструментального и прикладного программного обеспечения

(ИиППО)

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №2**

**по дисциплине**

«Технологии обработки транзакций клиент-серверных приложений»

Выполнил студент группы ИКБО-20-21 Фомичев Р.А.

Принял Маличенко С.В.

Москва 2024

**ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ**

1. **Работа с Rollback**

Начните транзакцию (командой BEGIN) и создайте новый заказ в таблице sales\_order с сегодняшней датой. Добавьте два предмета в таблицу item, связанных с созданным заказом.

Представьте, что пользователь не подтвердил заказ и все введённые данные необходимо отменить. Выполните отмену транзакции и проверьте, что никакой добавленной вами информации действительно не осталось.

Теперь представьте сценарий, в котором нужно отменить не все данные, а только последний из добавленных предметов. Для этого повторите все действия из предыдущего упражнения, но перед добавлением каждого предмета создавайте точку сохранения (с одним и тем же именем). После ввода второго предмета выполните откат к точке сохранения. Проверьте, что заказ и первый предмет остались.

В рамках той же транзакции добавьте еще один предмет и зафиксируйте транзакцию. Обратите внимание на то, что после этой операции отменить внесённые транзакцией изменения будет уже невозможно.

1. **Уровень изоляции Read Committed**

Перед началом выполнения задания проверьте, что в таблице sales\_order нет заказов на сумму total 1 000 рублей.

* В первом сеансе начните транзакцию (командой BEGIN). Выполните обновление таблицы sales\_order: увеличьте total в два раза в тех строках, где сумма равна 1 000 рублей.
* Во втором сеансе (откройте новое окно psql). Начните транзакцию (командой BEGIN). Вставьте в таблицу sales\_order новый заказ на 1 000 рублей и зафиксируйте транзакцию.
* В первом сеансе повторите обновление таблицы sales\_order и зафиксируйте транзакцию.

Осталась ли сумма добавленного заказа равной 1 000 рублей? Почему это не так?

1. **Уровень изоляции Repeatable Read**

Повторите предыдущее упражнение, но начните транзакцию в первом сеансе с уровнем изоляции транзакций Repeatable Read. Объясните различие полученных результатов.

Выполните указанные действия в двух сеансах:

* В первом сеансе начните новую транзакцию с уровнем изоляции Repeatable Read. Вычислите количество заказов с суммой 20 000 рублей.
* Во втором сеансе начните новую транзакцию с уровнем изоляции Repeatable Read. Вычислите количество заказов с суммой 30 000 рублей.
* В первом сеансе добавьте новый заказ на 30 000 рублей и снова вычислите количество заказов с суммой 20 000 рублей.
* Во втором сеансе добавьте новый заказ на 20 000 рублей и снова вычислите количество заказов с суммой 30 000 рублей.
* Зафиксируйте транзакции в обоих сеансах.

Соответствует ли результат ожиданиями? Можно ли сериализовать эти транзакции (иными словами, можно ли представить такой порядок последовательного выполнения этих транзакций, при котором результат совпадает с тем, что получился при параллельном выполнении)?

**РЕФЕРАТ**

Отчёт 22 страницы, 22 рисунка, 5 источников.

POSTGRESQL, SQL, СУБД, РАЗРАБОТКА БАЗ ДАННЫХ, ТРАНЗАКЦИИ

Объектом разработки является база данных.

Цель работы – работа с транзациями уровня Repeatable Read и Read Commited.

В процессе работы проводилось использование транзакций различных уровней.

Результатом является набор данных о транзациях с различными уровнями доступа.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc7781)

[1. РАЗРАБОТКА БАЗЫ ДАННЫХ 8](#_Toc14719)

[ЗАКЛЮЧЕНИЕ 10](#_Toc11659)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11](#_Toc18253)

[ПРИЛОЖЕНИЕ А 12](#_Toc1669)

[ФРАГМЕНТЫ КОДА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ 12](#_Toc6355)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

БД – база данных;

СУБД – система управления базами данных.

ВВЕДЕНИЕ

В мире баз данных, эффективное управление транзакциями и обеспечение соответствующего уровня изоляции играют критическую роль в обеспечении целостности данных и предотвращении конфликтов при параллельном доступе. В контексте реляционных баз данных, таких как PostgreSQL, уровни изоляции определяют, как одна транзакция видит изменения, сделанные другими транзакциями во время своего выполнения. В данной работе рассмотрим ключевые аспекты уровней изоляции в PostgreSQL, сосредотачивая внимание на Read Committed и Repeatable Read, а также на практическое применение точек сохранения для обеспечения надежности и целостности данных.

1. **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ**

## Базовая работа с транзакциями

## На рисунке 1 представлено состояние базы данных до начала работы.

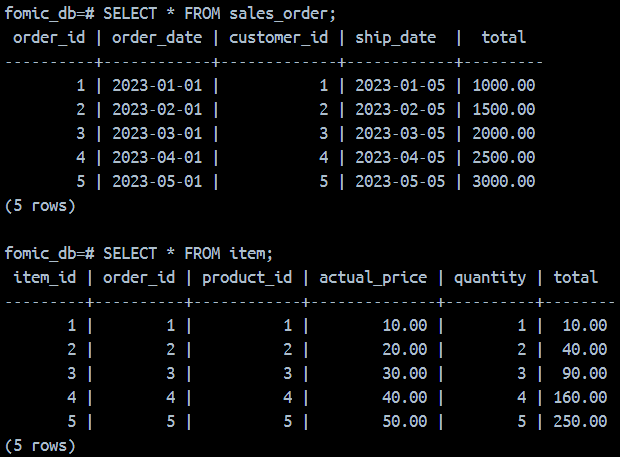
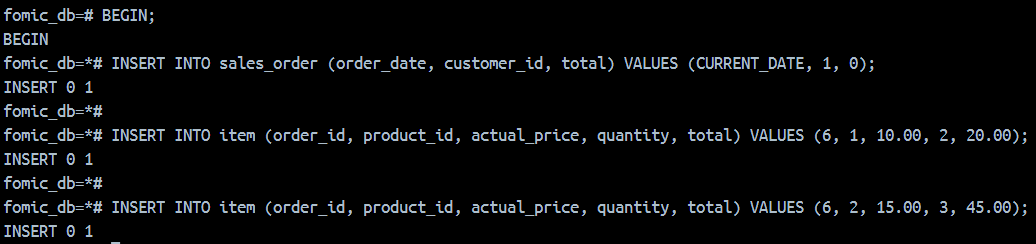


Рисунок 1 – Содержимое таблиц до начала работы

Согласно заданию была начата транзакция с помощью ключевого слова BEGIN и добавлены данные в таблицы sales\_order и item, изображено на рисунке 2.



Рисунках 2 – Создание транзакции и добавление данных в таблицы

Результат добавления данных в таблицы внутри транзакции представлен на рисунке 3.

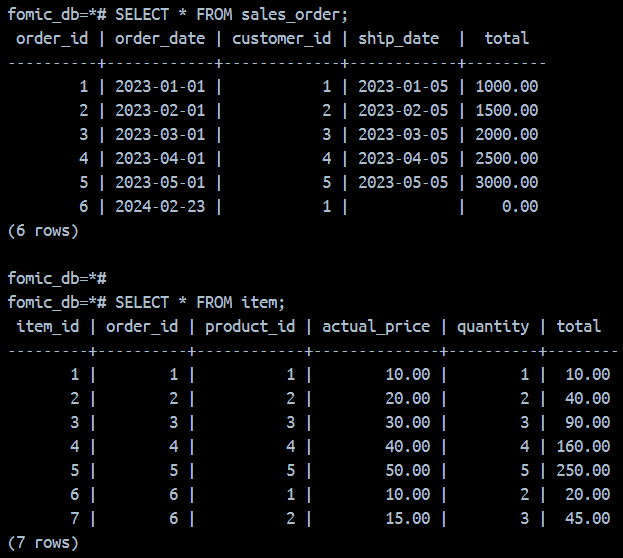


Рисунок 3 – Содержимое таблиц внутри транзакции после изменения

Откат транзакции представлен на рисунке 4.



Рисунок 4 – Откат транзакции

Содержимое таблиц после отката транзакции представлено на рисунке 5.

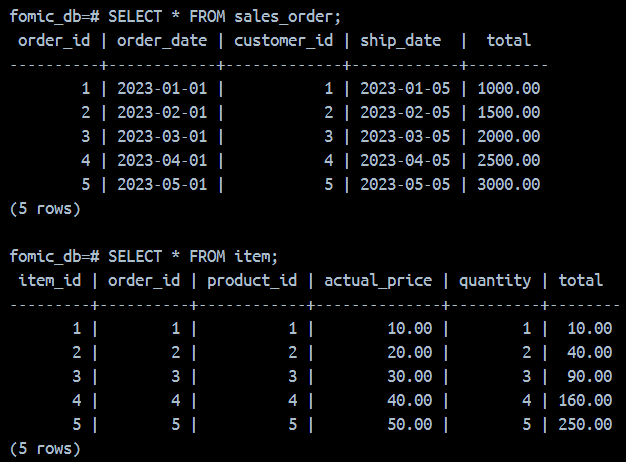


Рисунок 5 – Cодержимое таблиц после отката транзакции

Создание транзакции с заполнением таблиц данными и точками сохранения представлено на рисунке 6.

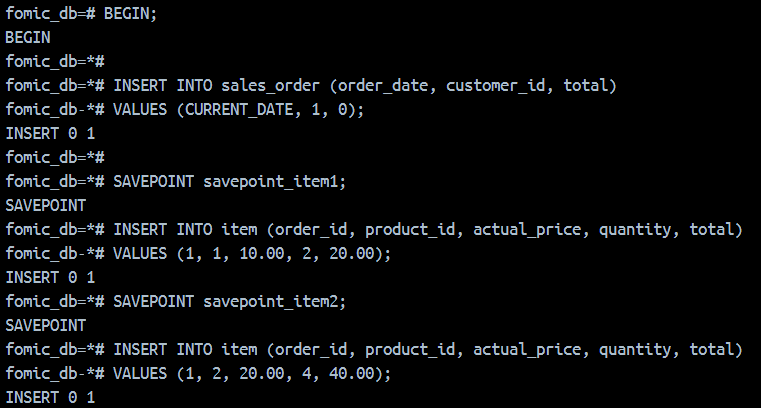


Рисунок 6 – Cоздание транзакции с заполнением таблиц данными и точками сохранения

Содержимое таблиц после заполнения внутри транзакций представлено на рисунке 7.

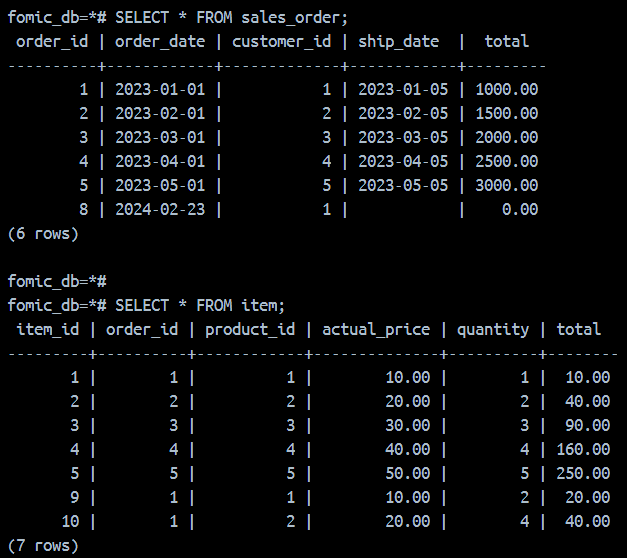


Рисунок 7 – Содержимое таблиц после заполнения внутри транзакции

Содержимое таблиц после отката к первой точке сохранения представлено на рисунке 8.

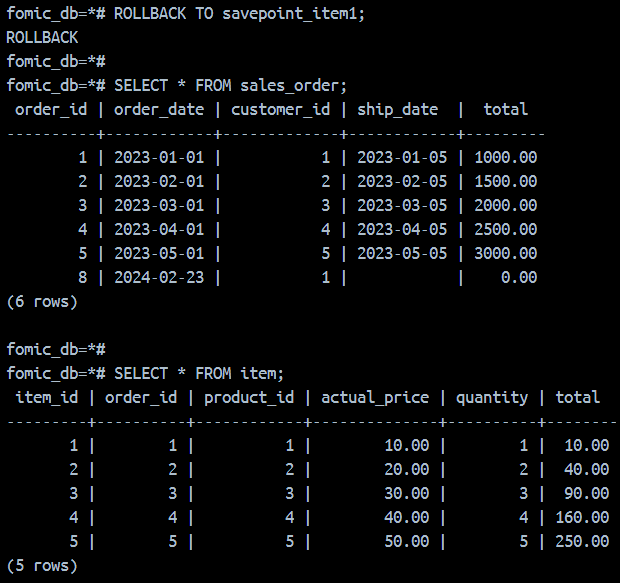


Рисунок 8 – Содержимое таблиц после отката к первой точке сохранения

Добавление данных в таблицу item и фиксация транзакции представлено на рисунке 9.

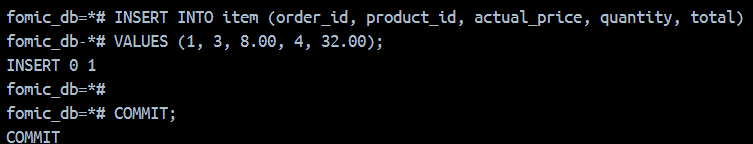


Рисунок 9 – Добавление данных в таблицу item и фиксация транзакции

Результат попытки отката к точке сохранения 1 представлен на рисунке 10.



Рисунок 10 – Результат попытки отката к точке сохранения после фиксации транзакции

Содержимое таблиц после фиксации транзакции представлено на рисунке 11.

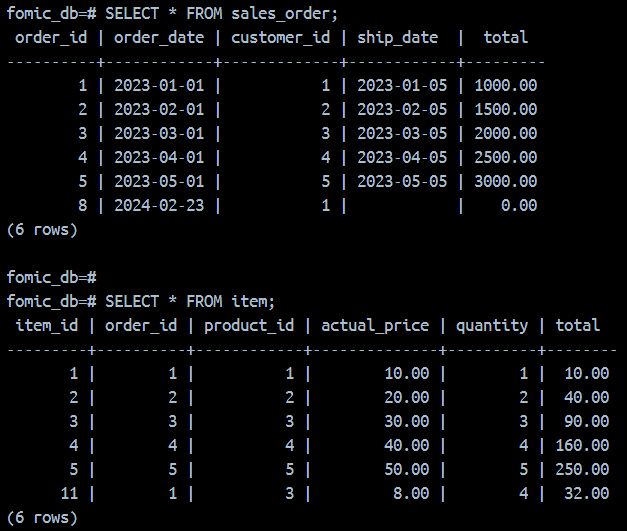


Рисунок 11 – Содержимое таблиц после фиксации транзакции

Был начат первый сеанс. Результат обновления таблицы sales\_order представлен на рисунке 12.

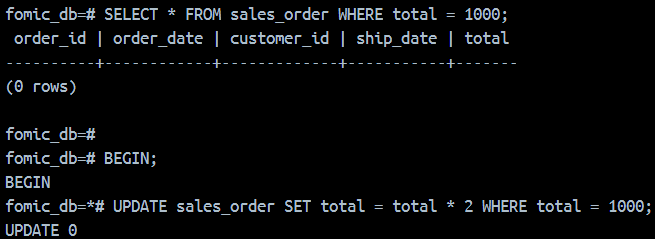


Рисунок 12 – Результат обновления таблицы sales\_order в первом сеансе

Был начат второй сеанс. Результат вставки записи в таблицу sales\_order представлен на рисунке 13.

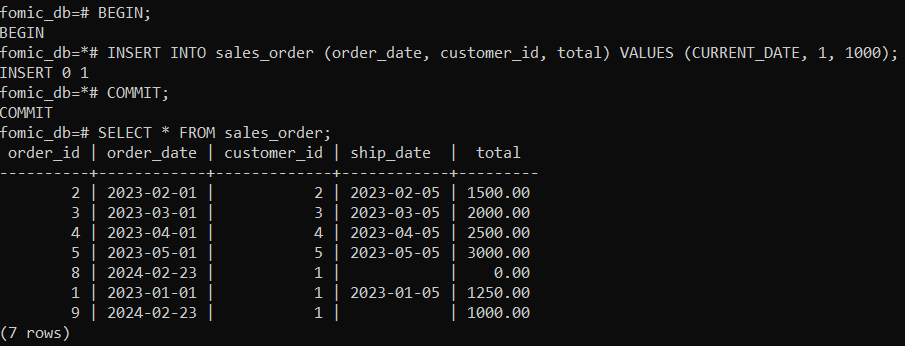


Рисунок 13 – Результат вставки записи в таблицу во втором сеансе

Результат повторного обновления страницы в первом сеансе представлен на рисунке 14.

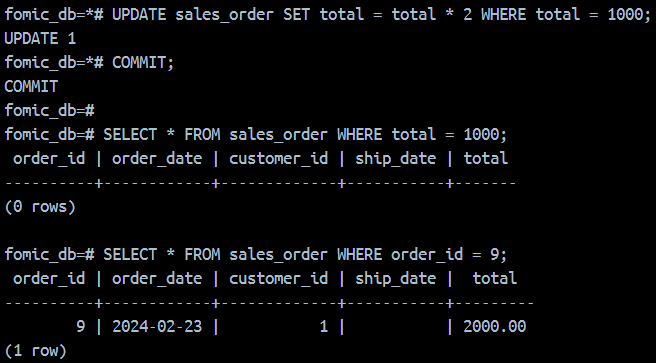


Рисунок 14 – Повторное обновление страницы в первом сеансе

Была начата транзакция с уровнем Repeatable Read в первом сеансе. Результат обновления таблицы sales\_order представлен на рисунке 15.

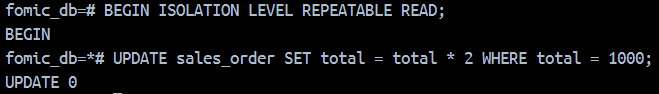


Рисунок 15 – Результат обновления таблицы в первом сеансе

Результат вставки новой записи в таблицу sales\_order во втором сеансе представлен на рисунке 16.

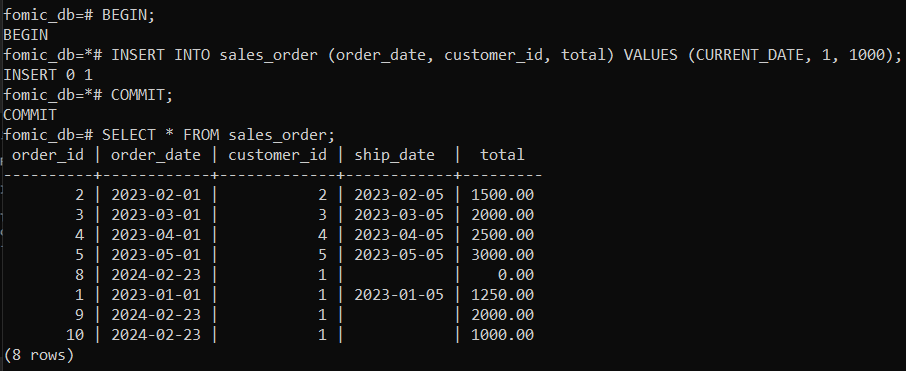


Рисунок 16 – Результат вставки записи в таблицу во втором сеансе

Результат повторного обновления таблицы в первом сеансе представлен на рисунке 17.

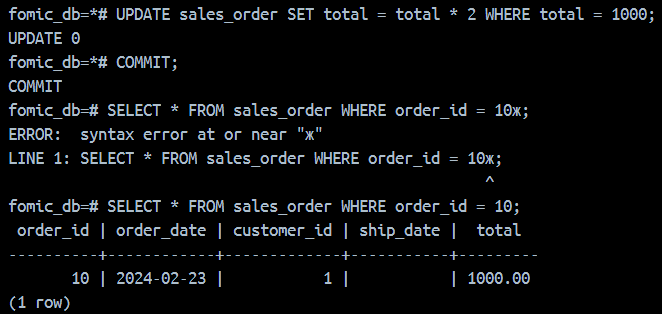


Рисунок 17 – Результат повторного обновления страницы в первом сеансе

В первом сеансе была начата транзакция с уровнем изоляции Repeatable Read. Результат вычисления заказов с суммой 20 000 рублей представлен на рисунке 18.

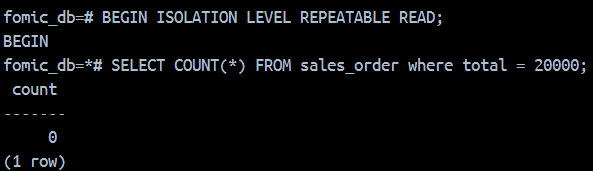


Рисунок 18 – Результат вычисления в первом сеансе

Во втором сеансе была начата транзакция с уровнем изоляции Repeatable Read. Результат вычисления заказов с суммой 30 000 рублей представлен на рисунке 19.

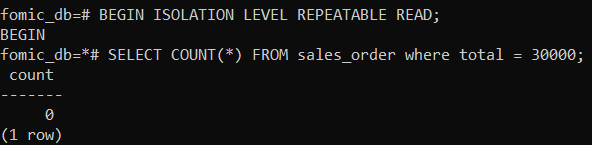


Рисунок 19 – Результат вычисления во втором сеансе

В первом сеансе был добавлен заказ стоимостью 30 000 рублей. Результат выполнения вычисления заказов стоимостью 20 000 рублей представлен на рисунке 20.

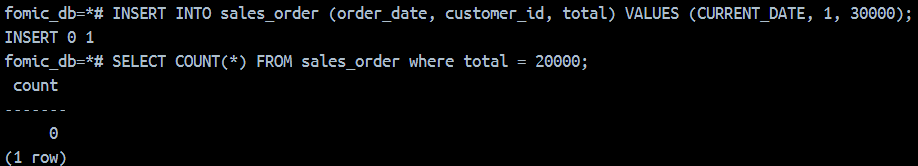


Рисунок 20 – Результат вычисления в первом сеансе

Во втором сеансе был добавлен заказ стоимостью 20 000 рублей. Результат выполнения вычисления заказов стоимостью 30 000 рублей представлен на рисунке 21.

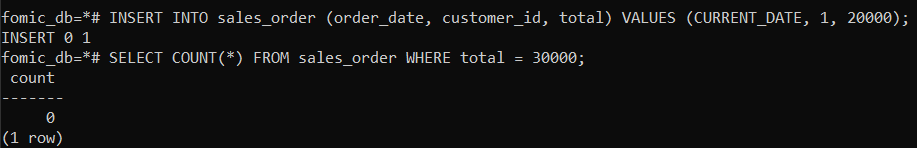


Рисунок 21 – Результат вычисления во втором сеансе

В обоих сеансах были зафиксированы изменения. Вывод таблицы sales\_order после фиксации изменений представлен на рисунке 22.

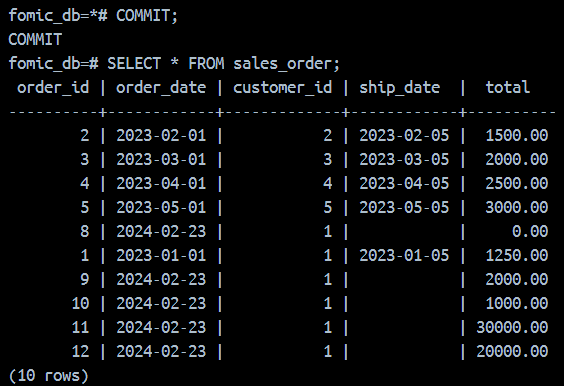


Рисунок 22 – Таблицы sales\_order после фиксации изменений

# ЗАКЛЮЧЕНИЕ

В процессе выполнения практической работы были исследованы различные аспекты управления транзакциями в PostgreSQL. Основное внимание уделялось процессам Rollback и уровням изоляции транзакций, таким как Read Committed и Repeatable Read.

Подробно рассмотрены возможности отмены транзакций с помощью оператора Rollback, включая как полный откат, так и частичную отмену с использованием точек сохранения. Этот опыт позволил лучше понять гибкость управления изменениями и внесением корректив в базу данных.

Изучение уровня изоляции Read Committed выявило важные аспекты параллельного выполнения транзакций и его воздействия на целостность данных. Это подчеркнуло значимость выбора подходящего уровня изоляции в зависимости от требований к конкретной базе данных.

Анализ уровня изоляции Repeatable Read расширил понимание о том, как этот уровень изоляции обеспечивает консистентность данных и предотвращает сценарии "dirty read". Это позволяет уверенно взаимодействовать с данными, обеспечивая их целостность и надёжность.

В результате выполнения этих заданий были приобретены ценные знания о методах эффективного управления транзакциями в PostgreSQL, что способствует более осознанному и обоснованному взаимодействию с данными в контексте баз данных.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. PostgreSQL: Официальная документация [Электронный ресурс] – URL: https://www.postgresql.org/docs/ (дата обращения: 14.02.2024).
2. PostgreSQL Tutorial for Beginners [Электронный ресурс] – URL: https://www.tutorialspoint.com/postgresql/index.htm (дата обращения: 14.02.2024).
3. PostgreSQL: Википедия [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/PostgreSQL (дата обращения: 14.02.2024).
4. PostgreSQL: Primary Key Constraint. [Электронный ресурс] – URL: https://www.tutorialsteacher.com/postgresql/foreign-key (дата обращения: 14.02.2024).
5. PostgreSQL Foreign Key Constraint. [Электронный ресурс] – URL: https://www.tutorialsteacher.com/postgresql/foreign-key (дата обращения: 17.12.2023)

# ПРИЛОЖЕНИЕ А

**ФРАГМЕНТЫ КОДА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

Листинг 1 - SQL код для выполнения задания 1

--Task 1 --

SELECT \* FROM sales\_order;

SELECT \* FROM item;

BEGIN;

INSERT INTO sales\_order (order\_date, customer\_id, total) VALUES (CURRENT\_DATE, 1, 0);

INSERT INTO item (order\_id, product\_id, actual\_price, quantity, total) VALUES (6, 1, 10.00, 2, 20.00);

INSERT INTO item (order\_id, product\_id, actual\_price, quantity, total) VALUES (6, 2, 15.00, 3, 45.00);

SELECT \* FROM sales\_order;

SELECT \* FROM item;

ROLLBACK;

SELECT \* FROM sales\_order;

SELECT \* FROM item;

Листинг 2 – SQL код для выполнения задания 2

-- Task 2 --

BEGIN;

INSERT INTO sales\_order (order\_date, customer\_id, total)

VALUES (CURRENT\_DATE, 1, 0);

SAVEPOINT savepoint\_item1;

INSERT INTO item (order\_id, product\_id, actual\_price, quantity, total)

VALUES (1, 1, 10.00, 2, 20.00);

SAVEPOINT savepoint\_item2;

INSERT INTO item (order\_id, product\_id, actual\_price, quantity, total)

VALUES (1, 2, 20.00, 4, 40.00);

SELECT \* FROM sales\_order;

Продолжение листинга 2

SELECT \* FROM item;

ROLLBACK TO savepoint\_item1;

SELECT \* FROM sales\_order;

SELECT \* FROM item;

Листинг 3 – SQL код для выполнения задания 3

-- Task 3 --

INSERT INTO item (order\_id, product\_id, actual\_price, quantity, total)

VALUES (1, 3, 8.00, 4, 32.00);

COMMIT;

ROLLBACK TO savepoint\_item1;

SELECT \* FROM sales\_order;

SELECT \* FROM item;

Листинг 4 – SQL код для выполнения задания 4

-- Task 4 --

UPDATE sales\_order SET total = 1250 WHERE order\_id = 1;

SELECT \* FROM sales\_order WHERE total = 1000;

-- Session 1 --

BEGIN;

UPDATE sales\_order SET total = total \* 2 WHERE total = 1000;

-- Session 2 --

BEGIN;

INSERT INTO sales\_order (order\_date, customer\_id, total) VALUES (CURRENT\_DATE, 1, 1000);

COMMIT;

-- Session 1 --

UPDATE sales\_order SET total = total \* 2 WHERE total = 1000;

COMMIT;

SELECT \* FROM sales\_order WHERE order\_id = 9;

Листинг 5 – SQL код для выполнения задания 5

-- Task 5 --

-- Session 1 --

BEGIN ISOLATION LEVEL REPEATABLE READ;

UPDATE sales\_order SET total = total \* 2 WHERE total = 1000;

-- Session 2 --

BEGIN;

INSERT INTO sales\_order (order\_date, customer\_id, total) VALUES (CURRENT\_DATE, 1, 1000);

COMMIT;

-- Session 1 --

UPDATE sales\_order SET total = total \* 2 WHERE total = 1000;

COMMIT;

SELECT \* FROM sales\_order WHERE order\_id = 10;

Листинг 6 – SQL код для выполнения задания 6

-- Task 6 --

-- Session 1--

BEGIN ISOLATION LEVEL REPEATABLE READ;

SELECT COUNT(\*) FROM sales\_order where total = 20000;

-- Session 2--

BEGIN ISOLATION LEVEL REPEATABLE READ;

SELECT COUNT(\*) FROM sales\_order where total = 30000;

-- Session 1--

INSERT INTO sales\_order (order\_date, customer\_id, total) VALUES (CURRENT\_DATE, 1, 30000);

SELECT COUNT(\*) FROM sales\_order where total = 20000;

-- Session 2--

INSERT INTO sales\_order (order\_date, customer\_id, total) VALUES (CURRENT\_DATE, 1, 20000);

SELECT COUNT(\*) FROM sales\_order WHERE total = 30000;

-- Session 1--

COMMIT;

-- Session 2—-

COMMIT;