|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 Сидоров С. Д.

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**

Отчет 14 страниц, 3 рисунка, 5 источников, 1 приложение.

POSTGRESQL, РАЗРАБОТКА БАЗ ДАННЫХ, СОЗДАНИЕ ТАБЛИЦ, ЗАПОЛНЕНИЕ ТАБЛИЦ, SQL

Объектом разработки является база данных.

Цель работы – разработка и заполнение базы данных.

В процессе работы проводилось практическое использование базы данных в виде создания и заполнения различных таблиц.

Результатом является подготовленное рабочее место с установленным PostgreSQL и заполненная база данных.

**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 современной сфере баз данных, PostgreSQL занимает выдающуюся позицию, предоставляя мощный инструмент для управления информацией. Эта система управления базами данных предлагает механизм транзакций, который играет ключевую роль в обеспечении последовательности и консистентности изменений в базе данных.

Транзакции в PostgreSQL выполняют множество задач, включая поддержку атомарности, согласованности, изолированности и долговечности (ACID). Эти принципы являются фундаментальными для обеспечения надёжности и предсказуемости работы с данными. В данном контексте, изучение и понимание принципов транзакций в PostgreSQL становится необходимым для разработчиков и администраторов баз данных, стремящихся обеспечить эффективное управление информацией в сложных и динамичных проектах.

Целью данного проекта является практическое исследование аспектов работы с транзакциями в базах данных на примере системы управления PostgreSQL. Проект сфокусирован на сценариях отмены транзакций, рассматривается уровни изоляции и их влияние на консистентность данных. Работа предоставляет конкретные примеры, демонстрируя важность правильного управления транзакциями в реальных сценариях.

1. **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ**

## Базовая работа с транзакциями На рисунках 1 и 2 представлено состояние базы данных до начала работы.

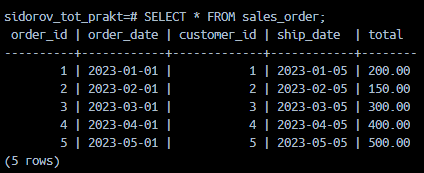


Рисунок 1 - Содержимое таблицы sales\_order до начала работы

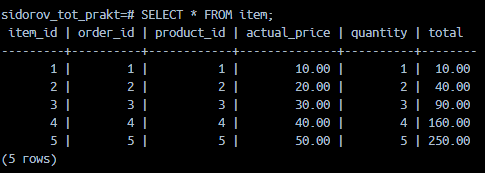
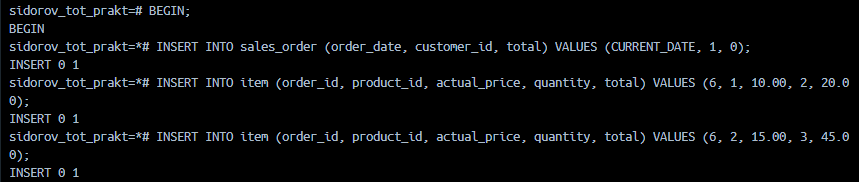


Рисунок 2 - Содержимое таблицы item до начала работы

Согласно заданию была начата транзакция с помощью ключевого слова BEGIN и добавлены данные в таблицы sales\_order и item, изображено на рисунке 3. На рисунках 4 - 5 изображён результат добавления данных в таблицы внутри транзакции.



Рисунках 3 - Создание транзакции и добавление данных в таблицы sales\_order и item

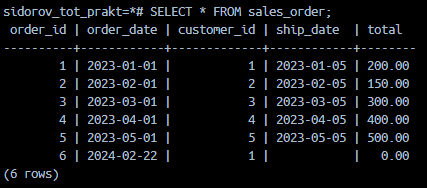


Рисунок 4 - Содержимое таблицы sales\_order внутри транзакции после изменения

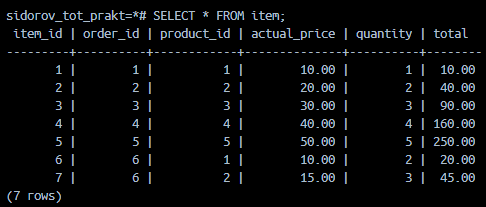


Рисунок 5 - Содержимое таблицы item внутри транзакции после изменения



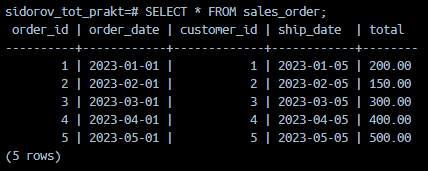
Рисунок 6 - Откат транзакции  


Рисунок 7 - Содержимое таблицы sales\_order после отката транзакции

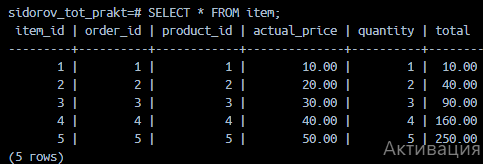


Рисунок 8 - Содержимое таблицы item после отката транзакции

Б1З23

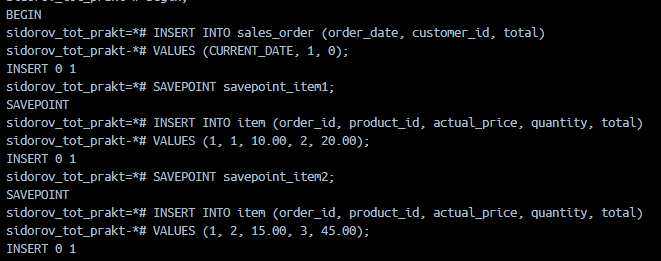


Рисунок 9 - Создание транзакции с заполнением таблиц sales\_order и item данными и точками сохранения

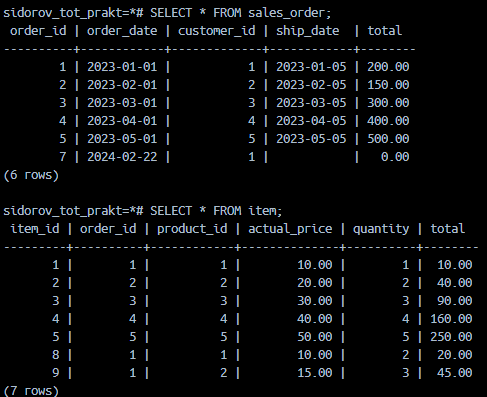


Рисунок 10 - Содержимое таблиц sales\_order и item после заполнения внутри транзакции

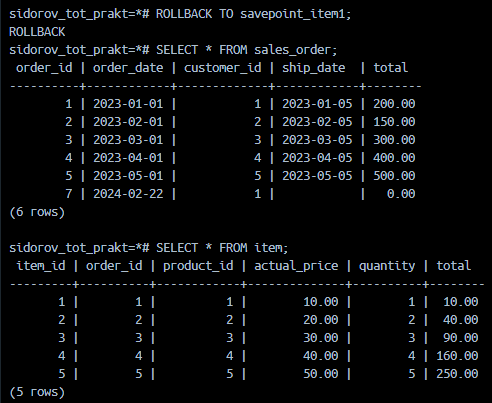


Рисунок 11 - Откат к первой точке сохранения и содержимое таблиц sales\_order и item после отката

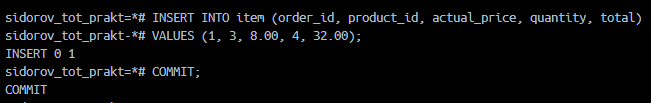


Рисунок 12 - Добавление данных в таблицу item и фиксация транзакции



Рисунок 13 - Результат попытки отката к точке сохранения после фиксации транзакции

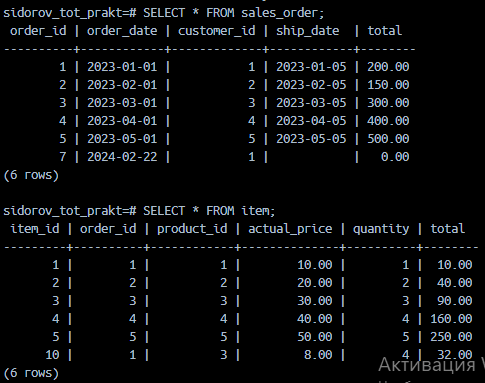
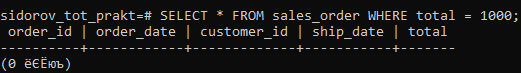
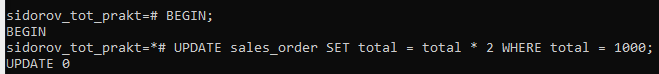
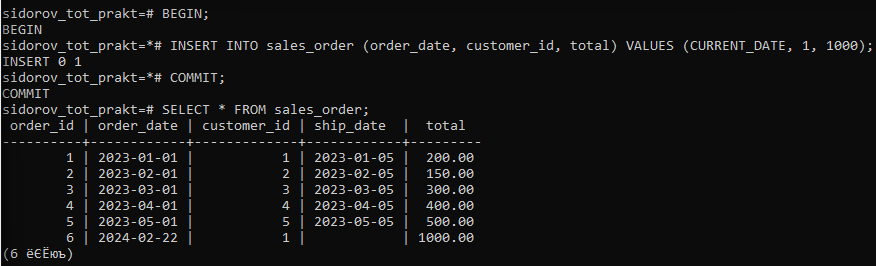


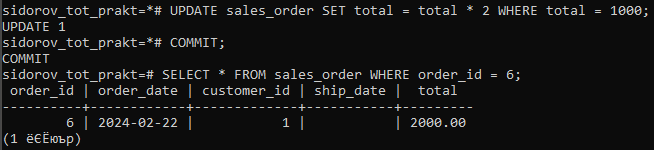
Рисунок 14 - Содержимое таблиц sales\_order и item после фиксации транзакции

Б2

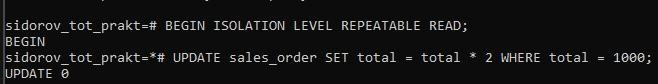


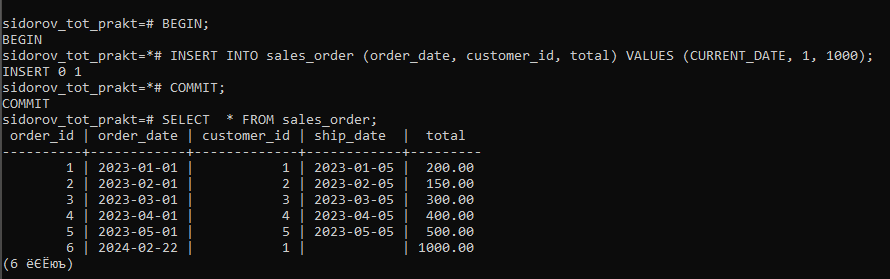


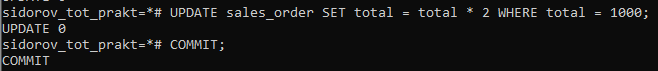


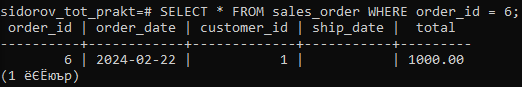


Б3З1



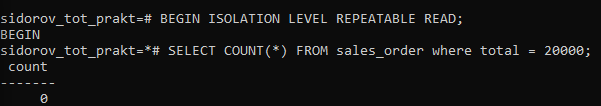




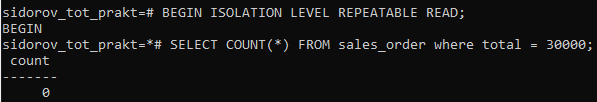


Б3З2

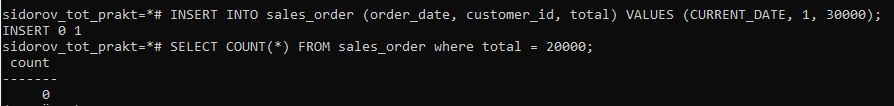
C1



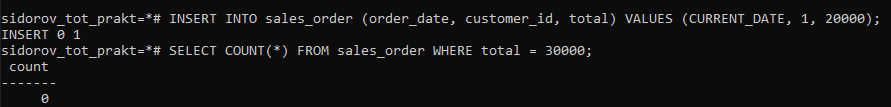
C2



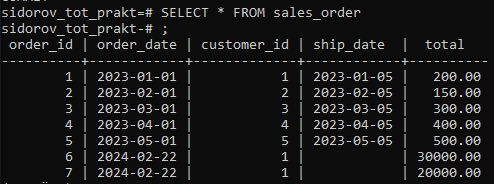
C1



C2



Tot



# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы было осуществлено детальное изучение процессов транзакций в системе управления базами данных PostgreSQL. Анализировалась работа с Rollback, где исследовалась возможность отмены транзакций с полным откатом и частичной отменой введённых данных с использованием точек сохранения. Этот опыт позволил получить понимание о гибкости управления изменениями средствами транзакций и точек сохранения.

Переход к рассмотрению уровня изоляции Read Committed выявил особенности воздействия параллельного выполнения транзакций на результаты обновления данных в рамках этого уровня. Это подчеркнуло значимость выбора соответствующего уровня изоляции в зависимости от требований к целостности данных.

Уровень изоляции Repeatable Read добавил новые аспекты в анализ. Сравнение результатов выявило, как этот уровень изоляции влияет на возможность сериализации транзакций и предотвращает сценарии "dirty read".

В ходе выполнения этих заданий были приобретены глубокие знания о методах эффективного управления транзакциями в PostgreSQL. Эти навыки предоставляют возможность более обоснованного и осознанного взаимодействия с данными в контексте баз данных, повышая их целостность и надёжность.

# 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 блока 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);

ROLLBACK;

SELECT \* FROM sales\_order;

SELECT \* FROM item;

Листинг 2 - SQL код для выполнения задания 2 блока 1

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 (6, 1, 10.00, 2, 20.00);

SAVEPOINT savepoint\_item2;

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

VALUES (6, 2, 15.00, 3, 45.00);

ROLLBACK TO savepoint\_item1;

SELECT \* FROM sales\_order;

SELECT \* FROM item;

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

VALUES (@order\_id, 3, 8.00, 4, 32.00);

COMMIT;

ROLLBACK TO savepoint\_item1;

SELECT \* FROM sales\_order;

SELECT \* FROM item;

Листинг 3 - SQL код для выполнения задания блока 2

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

-- Сессия 1 --

BEGIN;

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

-- Сессия 2 --

BEGIN;

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

COMMIT;

-- Сессия 1 --

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

COMMIT;

SELECT \* FROM sales\_order WHERE order\_id = @new\_order\_id;

Листинг 4 - SQL код для выполнения задания 1 блока 3

BEGIN ISOLATION LEVEL REPEATABLE READ;

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

-- Сессия 2 --

BEGIN;

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

COMMIT;

-- Сессия 1 --

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

COMMIT;

SELECT \* FROM sales\_order WHERE order\_id = @new\_order\_id;

Листинг 5 - SQL код для выполнения задания 2 блока 3

--Сессия 1--

BEGIN ISOLATION LEVEL REPEATABLE READ;

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

--Сессия 2--

BEGIN ISOLATION LEVEL REPEATABLE READ;

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

--Сессия 1--

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

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

--Сессия 2--

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

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

--Сессия 1--

COMMIT;

--Сессия 2--

COMMIT;