|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 23 страниц, 3 рисунка, 5 источников, 1 приложение.

POSTGRESQL, УРОВЕНЬ ИЗОЛЯЦИИ, ТАНЗАКЦИИ, НЕПОВТОРЯЕМОЕ ЧТЕНИЕ, 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](#_Toc159577970)

1. [ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ 8](#_Toc159577971)

[1.1 Базовая работа с транзакциями 8](#_Toc159577972)

[1.2 Уровень изоляции Read Committed 13](#_Toc159577973)

[1.3 Уровень изоляции Repeatable Read 14](#_Toc159577974)

[1.3.1 Задание 1……………………………………………………………… 14](#_Toc159577975)

[1.3.2 Задание 2……………………………………………………………… 15](#_Toc159577976)

[ЗАКЛЮЧЕНИЕ 18](#_Toc159577977)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc159577978)

[ПРИЛОЖЕНИЕ А 20](#_Toc159577979)

**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.

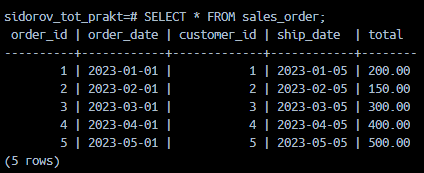


Рисунок 1 - Содержимое таблицы sales\_order до начала работы

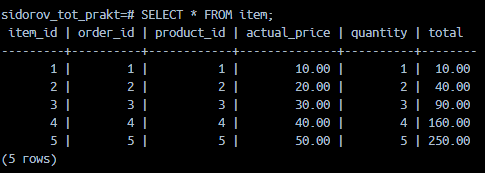
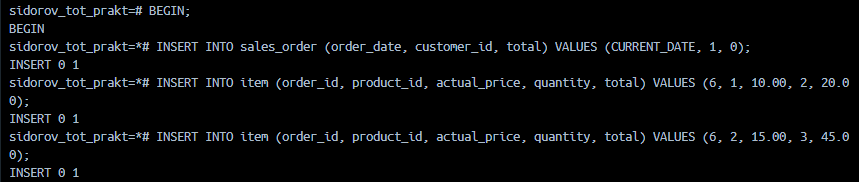


Рисунок 2 - Содержимое таблицы item до начала работы

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

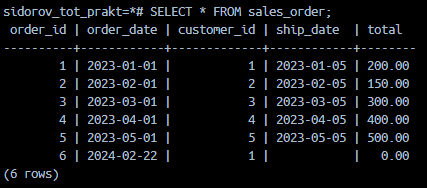


Рисунок 4 - Содержимое таблицы sales\_order внутри транзакции после изменения

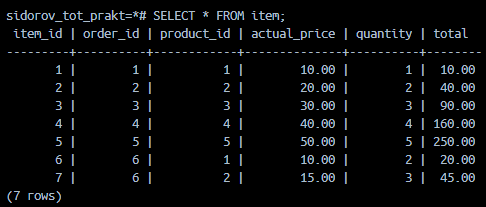


Рисунок 5 - Содержимое таблицы item внутри транзакции после изменения

Данные были добавлены внутри транзакции, после этого все изменения были отменены с помощью ключевого слова ROLLBACK, представленного на рисунке 6.



Рисунок 6 - Откат транзакции

Результат отката транзакции представлен на рисунках 7 - 8.

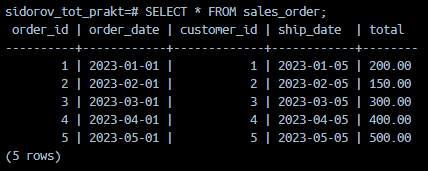


Рисунок 7 - Содержимое таблицы sales\_order после отката транзакции

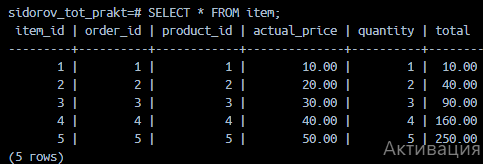


Рисунок 8 - Содержимое таблицы item после отката транзакции

Для выполнения дальнейшего задания база данных была приведена в изначальное состояние и была начата новая транзакция, в течении которой были добавлены строки в таблицу sales\_order и item с применением ключевого слова SAVEPOINT для возможности отката транзакции в определенное состояние, процесс обновления показан на рисунке 9, а результат обновления на рисунке 10.

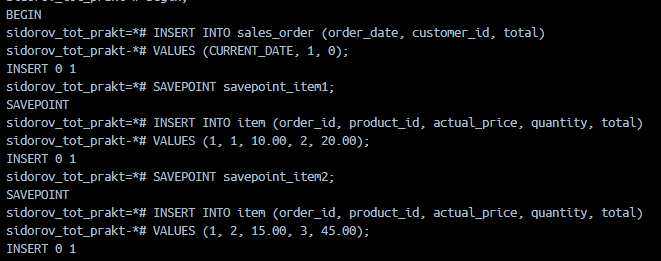


Рисунок 9 - Создание транзакции с заполнением таблиц sales\_order и item данными и точками сохранения

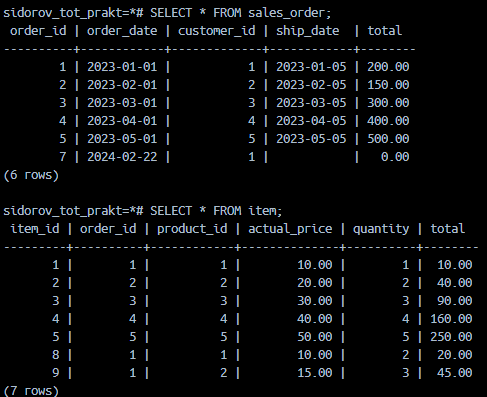


Рисунок 10 - Содержимое таблиц sales\_order и item после заполнения внутри транзакции

После проверки, что данные находятся в таблице с точки зрения транзакции, бал произведен откат до первой точки сохранения с помощью ключевого слова ROLLBACK TO и проверен результат отката. Процесс отката и результат представлены на рисунке 11.

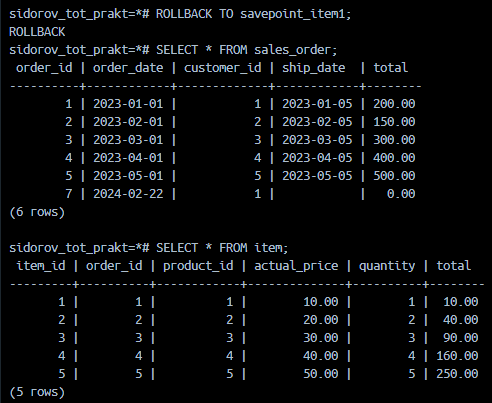


Рисунок 11 - Откат к первой точке сохранения и содержимое таблиц sales\_order и item после отката

После отката в таблицу item была добавлена еще одна строка и изменения были зафиксированы, как представлено на рисунке 12.

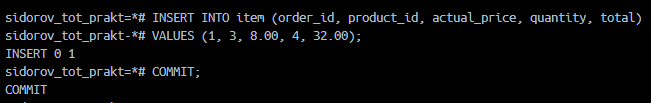


Рисунок 12 - Добавление данных в таблицу item и фиксация транзакции

После фиксации откатить изменения на точку сохранения не удается, попытка представлена на рисунке 13.



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

После фиксации была произведена проверка наличия вставленных в транзакции и зафиксированных данных, результат которой представлен на рисунке 14.

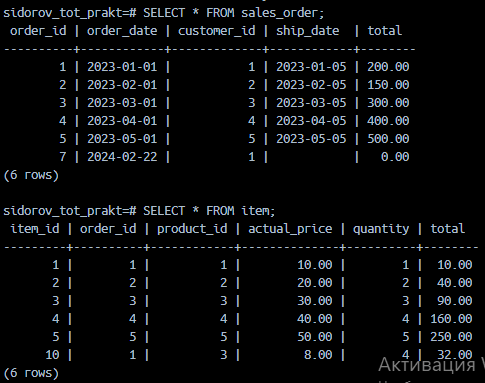


Рисунок 14 - Содержимое таблиц sales\_order и item после фиксации транзакции

## Уровень изоляции Read Committed

Переда началом выполнения задания была проверена таблицы sales\_order на наличие записей с total = 1000. Таких записей в таблице не оказалось, что продемонстрировано на рисунке 15.

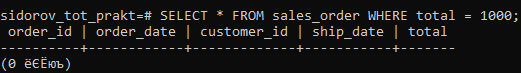


Рисунок 15 – Содержимое таблицы sales\_order со значением total = 1000

После проверки в первом сеансе была создана транзакция и выполнено обновление таблицы sales\_order, поле total было удвоено у строк с total = 1000. Было обновлено 0 строк, что показано на рисунке 16.

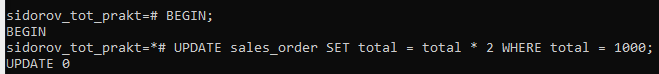


Рисунок 16 – Начало транзакции и удвоение значения поля total у всех строк со значением total = 1000

В тоже время во втором сеансе была начата новая транзакция с добавлением новой строки с total равным 1000 в таблицу sales\_order. Изменения были зафиксированы, результат действий представлен на рисунке 17.

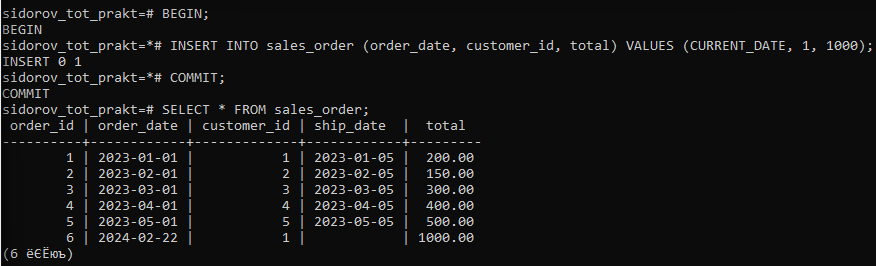


Рисунок 17 – Начало транзакции в сессии 2 с добавлением данных в таблицу sales\_order

После обновления таблицы во втором сеансе, была произведена повторная попытка обновления таблицы sales\_order, поле total было удвоено у строк с total = 1000. В данном случае была обновлена 1 строка, что продемонстрировано на рисунке 18. Это произошло потому, что уровень изоляции Read Committed допускает аномалию неповторяющегося чтения, при котором транзакция видит изменения, зафиксированные другими транзакциями.

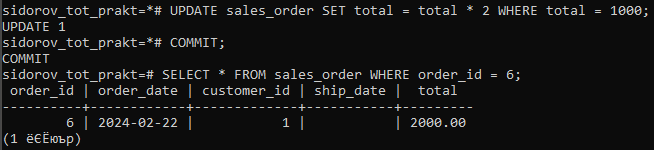


Рисунок 18 – Повторная попытка удвоения поля total, фиксация транзакций и проверка результата

## Уровень изоляции Repeatable Read

### Задание 1

После выполнения предыдущего задания система была приведена в изначальное состояние, и в первом сеансе была начата транзакция с уровнем изоляции Repeatable Read и выполнено обновление таблицы sales\_order, поле total было удвоено у строк с total = 1000. Было обновлено 0 строк, что показано на рисунке 19.

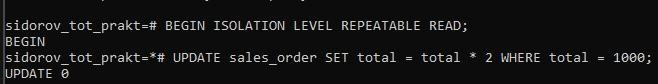


Рисунок 19 – Начало транзакции с уровнем изоляции Repeatable Read и удвоение поля total в некоторых строках

Во втором сеансе была начата транзакция без указания уровня изоляции с добавлением новой строки с total равным 1000 в таблицу sales\_order. Изменения были зафиксированы, результат действий представлен на рисунке 20.

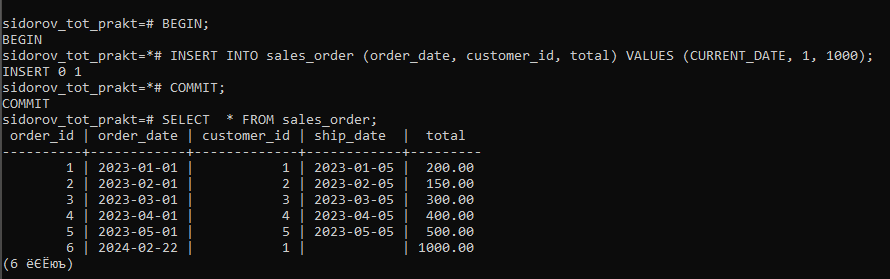


Рисунок 20 – Начало транзакции в сессии 2 и добавление новой записи в sales\_order с фиксацией изменений

После фиксации изменений во втором сеансе, в первом сеансе была произведена повторная попытка обновления таблицы sales\_order, поле total было удвоено у строк с total = 1000, что продемонстрировано на рисунке 21.

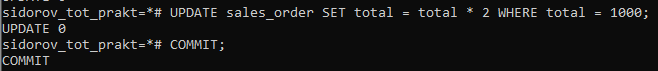


Рисунок 21 – Попытка удвоения поля total в сессии 1 и фиксация транзакции

В результате строки не подверглись изменению, что видно на рисунке 22, так как уровень изоляции Repeatable Read предотвращает аномалию неповторяемого чтения.

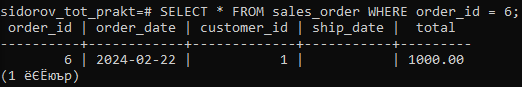


Рисунок 22 – Проверка удвоения поля total

### Задание 2

После выполнения предыдущего задания система была приведена в изначальное состояние. В первом сеансе была начата новая транзакция с уровнем изоляции Repeatable Read, а также вычислено количество заказов с суммой равной 20000, которых в таблице не оказалось, что представлено на рисунке 23.

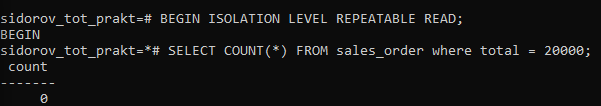


Рисунок 23 – Начало транзакции с уровнем изоляции Repeatable Read и просмотр данных таблицы sales\_order с total = 20000

Во втором сеансе была начата новая транзакция с уровнем изоляции Repeatable Read, а также вычислено количество заказов с суммой равной 30000, которых в таблице также не оказалось, что представлено на рисунке 24.

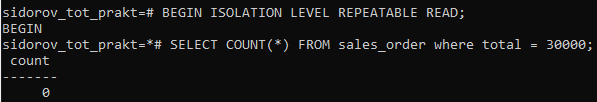


Рисунок 24 – Начало транзакции в сессии 2 с уровнем изоляции Repeatable Read и просмотр данных таблицы sales\_order с total = 30000

Вернувшись в первый сеанс в таблицу был добавлен новый заказ на сумму 30000, и снова вычислено количество заказов на 20000, которых также не оказалось, что представлено на рисунке 25.

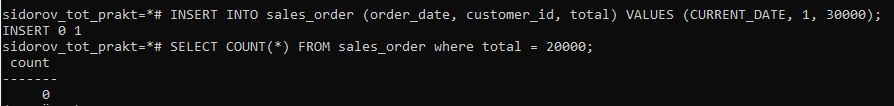


Рисунок 25 – Обновление данных в таблице sales\_order и проверка на наличие строк с total = 20000

После обновление данных в первом сеансе, во втором сеансе в таблицу был добавлен новый заказ на сумму 20000, и снова вычислено количество заказов на 30000, которых также не оказалось, что представлено на рисунке 26.

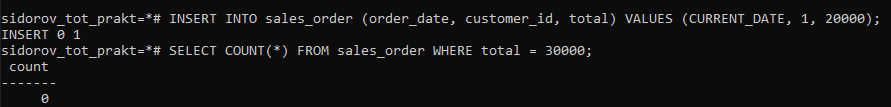


Рисунок 26 – Обновление данных в таблице sales\_order в сессии 2 и проверка на наличие строк с total = 30000

После фиксации изменений была проведена проверка, что строки действительно были вставлены в таблицу sales\_order, что показано на рисунке 27.

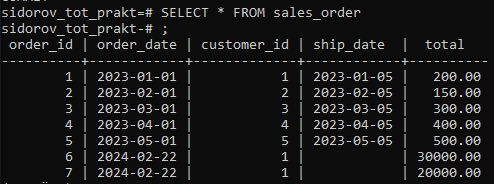


Рисунок 27 – Проверка наличия введенных в транзакциях после фиксации

Результат проведенных действия был ожидаемым. Так как обе транзакции имеют уровень изоляции Repeatable Read, то они обе не видят данные, обновляемые другими транзакциями, поэтому при повторных проверках не было видно добавленных строк.  
 Сериализовать данные транзакции не представляется возможным, так как в случае последовательного выполнения с фиксацией изменений, транзакция выполняемая второй, будет видеть добавленные строки первой, что не соответствует полученному результату.

# 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;