Федеральное государственное образовательное бюджетное

учреждение высшего образования

**«Финансовый университет при Правительстве Российской Федерации»**

**(Финансовый университет)**

Факультет «Информационных технологий и анализа больших данных»

**ОТЧЕТ**

**по практической работе № 3**

**по теме «Изучение транзакций»**

**по дисциплине Б.1.1.3.4 Системы управления базами данных**

**Вариант 10**

Выполнил: студент группы ИД23-1

Маслов Александр Николаевич

Проверил преподаватель: Цветков И.Е.

**Москва – 2025**

**Задание №1.**

1. Модифицируйте сценарий выполнения транзакций: в первой транзакции вместо фиксации изменений выполните их отмену с помощью команды ROLLBACK и посмотрите, будет ли удалена строка и какая конкретно.

**Запрос:**

**Первая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range < 2000;

UPDATE aircrafts SET range = 2100 WHERE aircraft\_code = 'CN1';

UPDATE aircrafts SET range = 1900 WHERE aircraft\_code = 'CR2';

ROLLBACK;

**Вторая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range < 2000;

DELETE FROM aircrafts WHERE range < 2000;

COMMIT;

SELECT \* FROM aircrafts;

**Удалится строка:**

****

1. Можно ли говорить, что в такой ситуации имеет место потерянное обновление? Если оно имеет место, то что можно предпринять для его недопущения? Обоснуйте ваш ответ.

Да, в описанной ситуации имеет место потерянное обновление. Это происходит, когда две транзакции одновременно изменяют одни и те же данные, и одно из изменений перезаписывает другое, не учитывая предыдущего. В результате сохраняется только последнее изменение, а первое теряется.

Можно использовать SELECT FOR UPDATE для блокировки строки перед изменением.

1. Модифицируйте этот эксперимент: вместо операции UPDATE используйте операцию INSERT.

**Запрос:**

**Первая транзакция:**

BEGIN;

SELECT \*

FROM aircrafts\_tmp

WHERE range > 6000;

**Вторая транзакция:**

BEGIN;

INSERT INTO aircrafts\_tmp (aircraft\_code, model, range) VALUES ('321', 'Airbus A321-200', 6500);

COMMIT;

1. Была рассмотрена команда SELECT ... FOR UPDATE, выполняющая блокировку на уровне отдельных строк. Организуйте две параллельные транзакции с уровнем изоляции Read Committed и выполните с ними ряд экспериментов. В первой транзакции заблокируйте некоторое множество строк, отбираемых с помощью условия WHERE. А во второй транзакции изменяйте условие выборки таким образом, чтобы выбираемое множество строк:

– являлось подмножеством множества строк, выбираемых в первой транзакции;

– являлось надмножеством множества строк, выбираемых в первой транзакции;

– пересекалось с множеством строк, выбираемых в первой транзакции;

– не пересекалось с множеством строк, выбираемых в первой транзакции.

Наблюдайте за поведением команд выборки в каждой транзакции. Попробуйте обобщить ваши наблюдения.

**Первая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range > 7000 FOR UPDATE;

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Первый эксперимент:**

**Вторая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range > 9000 FOR UPDATE;

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**После COMMIT первой транзакции:**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Второй эксперимент:**

**Вторая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range > 6000 FOR UPDATE;

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**После COMMIT первой транзакции:**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Третий эксперимент:**

**Вторая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range > 7500 FOR UPDATE;

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**После COMMIT первой транзакции:**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Четвертый эксперимент:**

**Вторая транзакция:**

BEGIN;

SELECT \* FROM aircrafts WHERE range < 7000 FOR UPDATE;

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Вывод:**

SELECT ... FOR UPDATE блокирует отдельные строки, а не всю таблицу, но пересекающиеся запросы ожидают освобождения заблокированных строк.

1. В тексте главы была рассмотрена транзакция для выполнения бронирования билетов. Для нее был выбран уровень изоляции Read Committed. Как вы думаете, если одновременно будут производиться несколько операций бронирования, то, может быть, имеет смысл «ужесточить» уровень изоляции до Serializable? Или нет необходимости это делать? Обдумайте и вариант с использованием явных блокировок. Обоснуйте ваш ответ.

Имеет смысл поднять уровень до Serializable, так как Read Committed не гарантирует корректность бронирования в условиях высокой конкуренции.

Serializable же гарантирует, что транзакции выполняются так, как если бы они шли строго последовательно, а не параллельно.

Альтернативным вариантом является использование SELECT ... FOR UPDATE.

1. Попытайтесь объяснить, почему транзакции не удалось сериализовать. Что можно сделать, чтобы удалось зафиксировать обе транзакции? Одно из возможных решений — понизить уровень изоляции. Другим решением может быть создание индекса по столбцу flight\_id для таблицы ticket\_flights. Почему создание индекса может помочь?

Ошибка **"**не удалось сериализовать доступ из-за зависимостей чтения/записи**"** означает, что PostgreSQL обнаружил ситуацию, в которой две транзакции зависели друг от друга, но их нельзя было безопасно выполнить параллельно без риска аномалий.

Если конкурентное бронирование билетов допускает небольшие расхождения в данных, можно использовать Repeatable Read или Read Committed. Они менее строгие и не приводят к конфликтам сериализации.

Второй вариант — это создать индекс по flight\_id в ticket\_flights.

Оно помогает, потому что:

- PostgreSQL использует индексные блокировки, а не блокировки всей таблицы.

- Снижает вероятность пересечения блокировок и делает выборку SELECT \* FROM ticket\_flights WHERE flight\_id = ? более эффективной.

1. В первой транзакции условие в команде SELECT такое: ... WHERE flight\_id = 13881. В команде вставки в таблицу ticket\_flights значение поля flight\_id также равно 13881. Во второй транзакции в этих же командах используется значение 5572. Поменяйте местами значения в командах SELECT и повторите эксперименты, выполнив транзакции параллельно с уровнем изоляции Serializable. Почему сейчас наличие индекса не помогает зафиксировать обе транзакции? Вспомните, что аномалия сериализации — это ситуация, когда параллельное выполнение транзакций приводит к результату, невозможному ни при каком из вариантов упорядочения этих же транзакций при их последовательном выполнении.

Почему индекс больше не помогает?

* Индекс помогает ускорить SELECT, но не решает проблему записи (INSERT).
* Транзакции поочередно читают одни flight\_id, а записывают другие, что приводит к перекрестным зависимостям.
* PostgreSQL не может найти корректный последовательный порядок выполнения транзакций.

Почему это аномалия сериализации?

Если представить транзакции в последовательном порядке, невозможно воспроизвести тот же эффект, который получается при их параллельном выполнении.