МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра цифровой трансформации (ЦТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7

по дисциплине «Разработка баз данных»

Студент группы *ИНБО-12-23. Албахтин И.В.* _____
(подпись)

Ассистент *Брайловский А.В.* _____
(подпись)

Москва 2025 г.

ПРАКТИЧЕСКАЯ РАБОТА №7. ОПТИМИЗАЦИЯ ЗАПРОСОВ И УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ В POSTGRESQL

Цель работы:

Целью данной практической работы является формирование у студентов практических навыков анализа и оптимизации производительности SQL-запросов, а также освоение механизмов управления транзакциями для обеспечения целостности данных (согласно принципам ACID) в СУБД PostgreSQL.

По завершении работы студент должен уметь:

• Сформировать практический навык анализа производительности SQL-запросов с использованием инструмента EXPLAIN ANALYZE.

• Научиться интерпретировать планы выполнения (ПВ), выявляя неэффективные операции, такие как полное сканирование таблицы (Seq Scan).

• Освоить создание различных типов индексов (B-tree, Partial, Function-based), как основного средства для ускорения операций поиска данных.

• Закрепить понимание транзакций как логической единицы работы и освоить использование команд BEGIN, COMMIT и ROLLBACK для обеспечения атомарности операций.

• Научиться моделировать и устранять проблемы параллельного доступа (аномалию «Неповторяемое чтение») с помощью уровней изоляции транзакций (REPEATABLE READ).

Постановка задачи:

Для выполнения практической работы необходимо последовательно выполнить следующие шаги, адаптируя примеры из БД «Аптека» к вашей собственной базе данных:

Подготовка базы данных.

Следуя руководству, наполнить одну из ключевых таблиц вашей БД большим объемом данных (не менее 20 000 строк). Это обязательно для корректной демонстрации работы оптимизатора.

Задание №1: анализ и оптимизация (3 сценария)

Определить три различных «медленных» запроса к вашей БД, которые можно оптимизировать с помощью разных типов индексов (например, стандартный B-Tree, индекс по выражению, частичный/отфильтрованный индекс).

Если в вашей базе недостаточно данных, выполните для одной из своих таблиц действия по автоматической генерации содержимого, описанные в разделе «Подготовка базы данных».

Для каждого из 3-х сценариев:

1. Выполнить анализ запроса «КАК ЕСТЬ» (без индекса) с помощью EXPLAIN ANALYZE.

2. Привести план выполнения «ДО», письменно проанализировать его и выявить причину низкой производительности (например, Seq Scan).

3. Создать необходимый INDEX для оптимизации.

4. Повторно выполнить EXPLAIN ANALYZE.

5. Привести план выполнения «ПОСЛЕ», демонстрирующий использование индекса (например, Index Scan).

6. Обязательно сформировать сравнительную таблицу (см. Таблица 1), демонстрирующую разницу в производительности (план, Execution Time) «ДО» и «ПОСЛЕ».

Задание №2: демонстрация атомарной транзакции (COMMIT).

По примеру раздела 2.2 реализуйте в своей базе одну бизнес-операцию (минимум две связанные операции изменения данных) внутри транзакции BEGIN…COMMIT.

Задокументировать все шаги и результаты, сделать выводы.

Задание №3: демонстрация отката транзакции (ROLLBACK). Адаптировать приведённый в разделе 2.3 SQL-скрипт, моделирующий сбой операции, под свою предметную область, повторив описанные действия.

Задокументировать все шаги и результаты, сделать выводы.

Задание №4: моделирование аномалии «Неповторяемое чтение».

Используя два редактора SQL, смоделировать проблему «неповторяемое чтение» на уровне изоляции по умолчанию (READ COMMITTED) по приведённому в разделе 2.4 образцу.

Задокументировать все шаги и результаты, сделать выводы.

Задание №5: устранение аномалии «Неповторяемое чтение».

Повторить моделирование из Задания №4, но с использованием уровня изоляции REPEATABLE READ. В качестве образца использовать раздел 2.5.

Задокументировать все шаги и результаты, сделать выводы.

ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ

Таблица 1. Структура таблицы invoice

```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'invoice'
ORDER BY ordinal_position;
```

olumns 1

ELECT column_name, data_type, is_nulla Введите SQL выражение чтобы отфи

	A-Z column_name	A-Z data_type	A-Z is_nullable
1	invoice_id	integer	NO
2	maintenance_id	integer	YES
3	total_amount	numeric	YES
4	payment_status	character varying	YES

Таблица 2. Структура таблицы maintenance_work

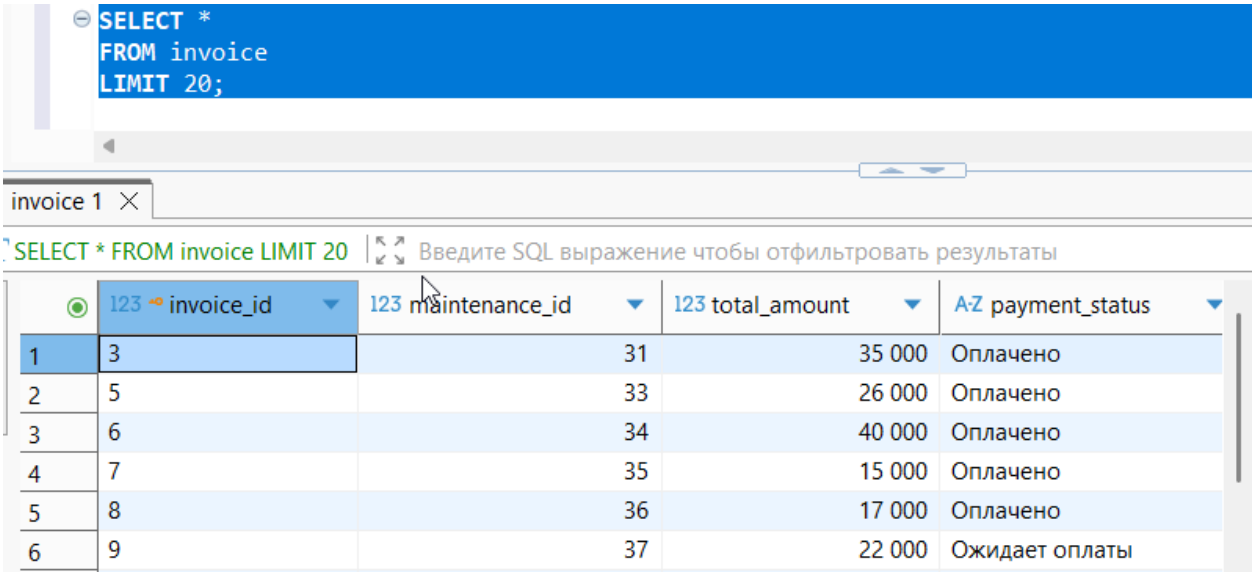
```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'maintenance_work'
ORDER BY ordinal_position;
```

lumns 1

LECT column_name, data_type, is_nulla Введите SQL выражение чтобы отфи

	A-Z column_name	A-Z data_type	A-Z is_nullable
	maintenance_work	integer	NO
	maintenance_id	integer	YES
	work_type_id	integer	YES
	part_id	integer	YES
	quantity	integer	YES

Таблица 3. Таблица invoic



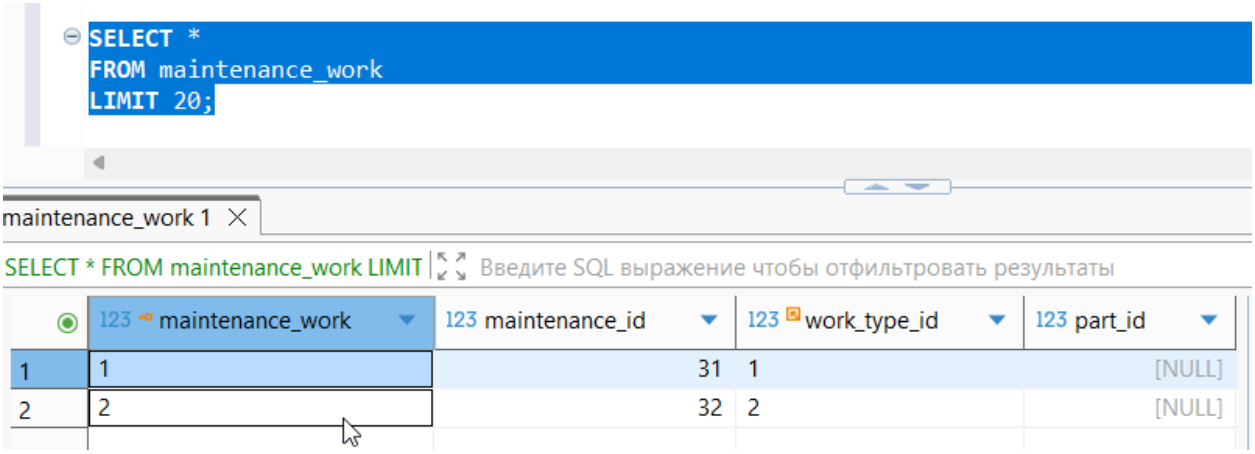
```
SELECT *
FROM invoice
LIMIT 20;
```

invoice 1

SELECT * FROM invoice LIMIT 20 | Введите SQL выражение чтобы отфильтровать результаты

	invoice_id	maintenance_id	total_amount	payment_status
1	3	31	35 000	Оплачено
2	5	33	26 000	Оплачено
3	6	34	40 000	Оплачено
4	7	35	15 000	Оплачено
5	8	36	17 000	Оплачено
6	9	37	22 000	Ожидает оплаты

Таблица 4. Таблица maintenance_work



```
SELECT *
FROM maintenance_work
LIMIT 20;
```

maintenance_work 1

SELECT * FROM maintenance_work LIMIT | Введите SQL выражение чтобы отфильтровать результаты

	maintenance_work	maintenance_id	work_type_id	part_id
1	1	31	1	[NULL]
2	2	32	2	[NULL]

Задание №1: анализ и оптимизация (3 сценария)

albakhtin_iv | *<dbstud> Script | 123 invoice_id | 123 maintenance_id

```
DO $$
DECLARE
    max_maint INT;
BEGIN
    SELECT MAX(maintenance_id)
    INTO max_maint
    FROM maintenance_work;
    FOR i IN 1..20000 LOOP
        INSERT INTO invoice (
            maintenance_id,
            total_amount,
            payment_status
        )
        VALUES (
            (1 + floor(random() * max_maint))::INT,
            (1000 + random() * 50000)::NUMERIC(10,2),
            CASE floor(random() * 2)
                WHEN 0 THEN 'Ожидает оплаты'
                ELSE 'Оплачено'
            END
        );
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0,451s
Start time	Fri Dec 05 19:58:48 MSK 2025
Finish time	Fri Dec 05 19:58:49 MSK 2025
Query	DO $$
	DECLARE
	max_maint INT;
	BEGIN

Рисунок 1 – Скрипт заполнения 20 000 случайными валидными записями (случайно запустил 2 раза)

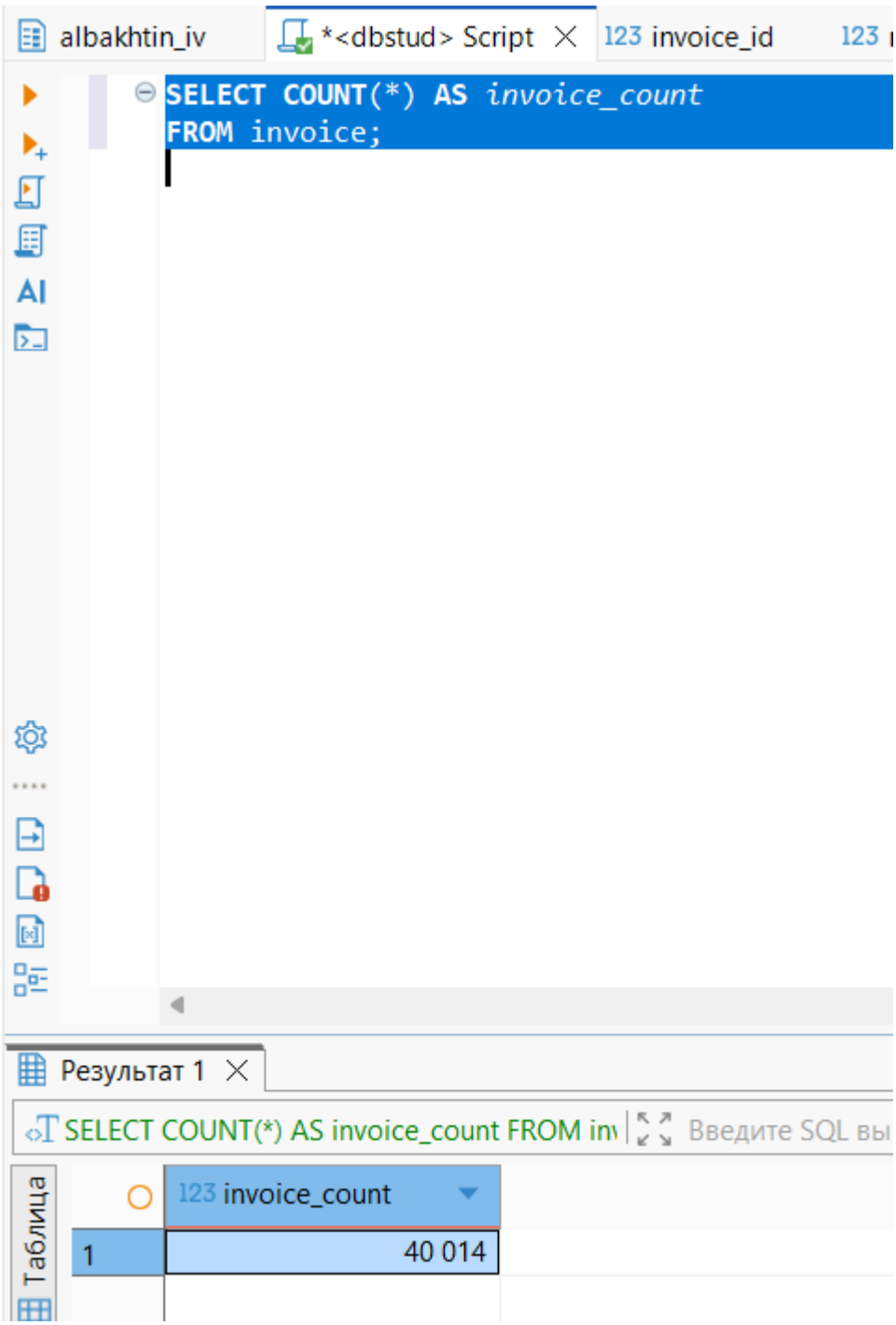


Рисунок 2 – Проверка количества строк в таблице invoice после генерации тестовых данных

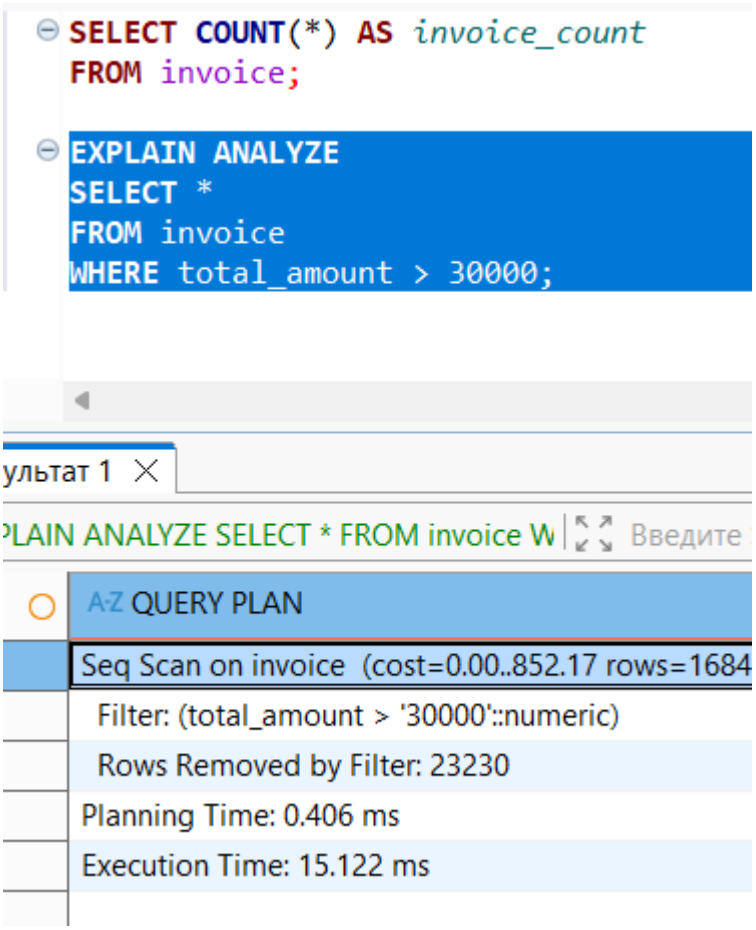


Рисунок 3 – План выполнения запроса до оптимизации

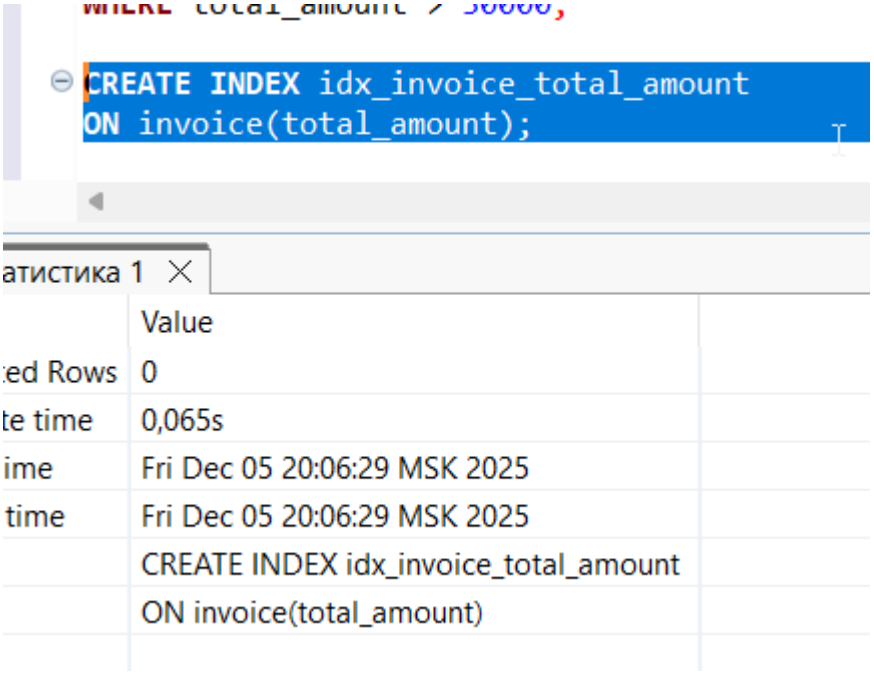


Рисунок 4 – Создание индекса по total_amount

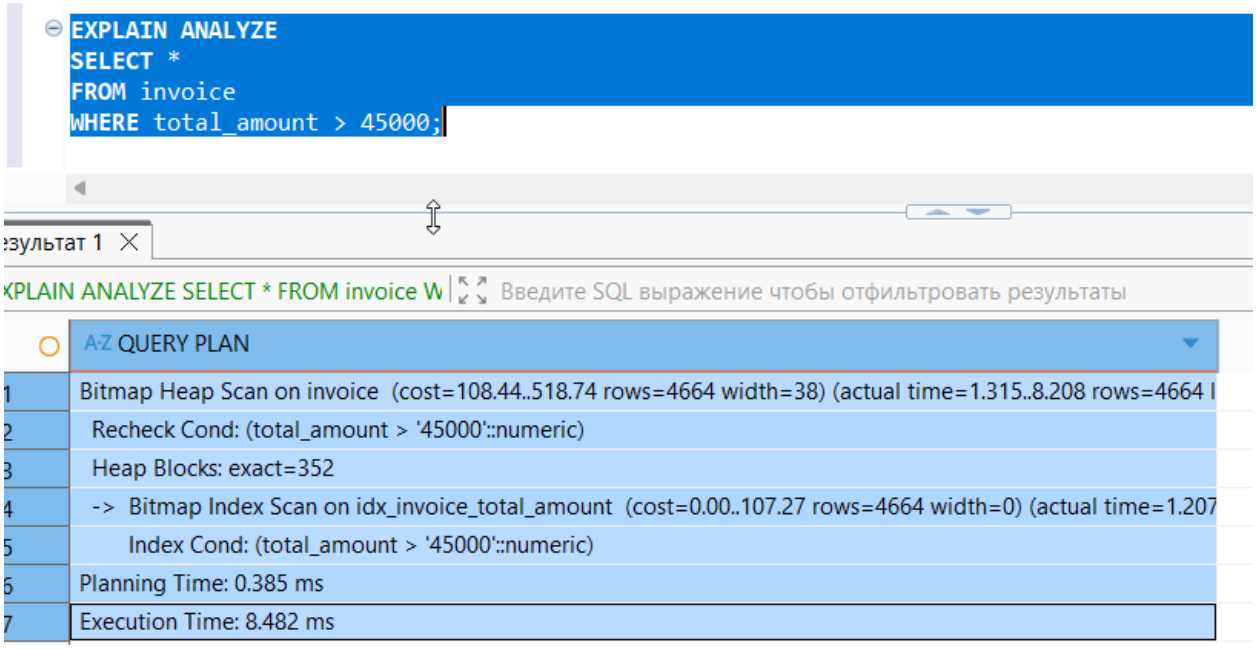


Рисунок 5 – План выполнения запроса после оптимизации

Метрика	До оптимизации	После оптимизации	Вывод
План выполнения	Seq Scan	Bitmap Index Scan + Bitmap Heap Scan	Индекс используется
Execution Time	~15 ms	~8 ms	Запрос ускорен примерно в 2 раза
Фильтр	total_amount > 30000	total_amount > 45000	Селективность выше → оптимизатор выбирает индекс

Таблица 1 – сравнительная таблица

Метрика	До оптимизации	После оптимизации	Вывод
План выполнения	Seq Scan	Bitmap Index Scan + Bitmap Heap Scan	Индекс используется
Execution Time	~15 ms	~8 ms	Запрос ускорен примерно в 2 раза
Фильтр	total_amount > 30000	total_amount > 45000	Селективность выше → оптимизатор выбирает индекс

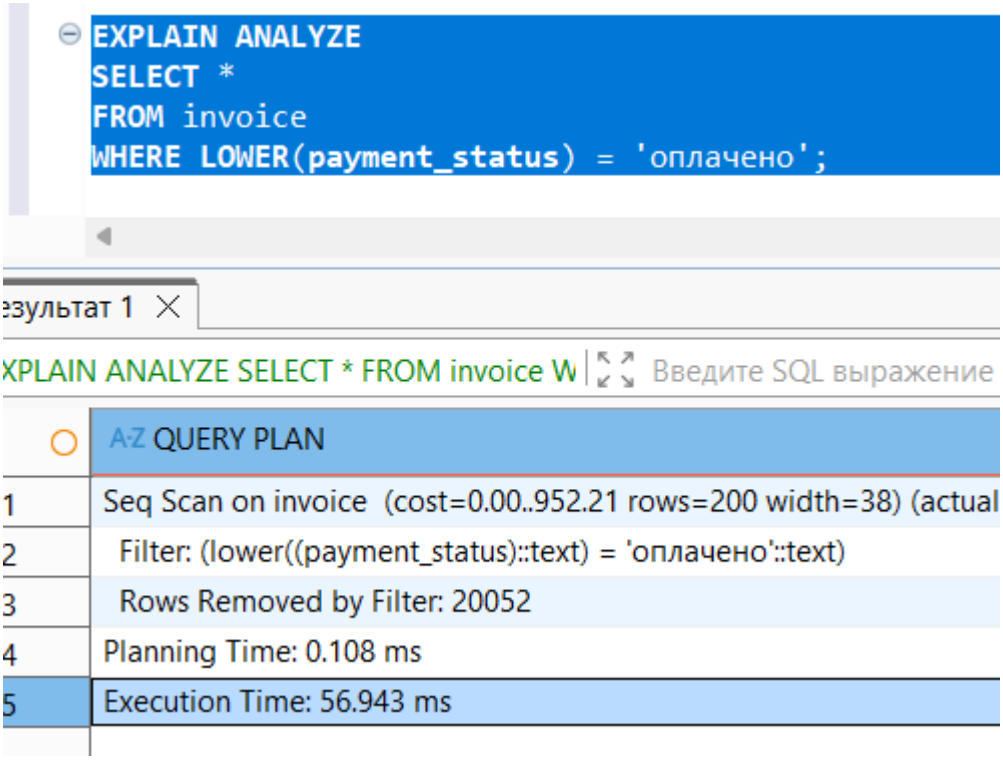


Рисунок 6 – План выполнения до оптимизации

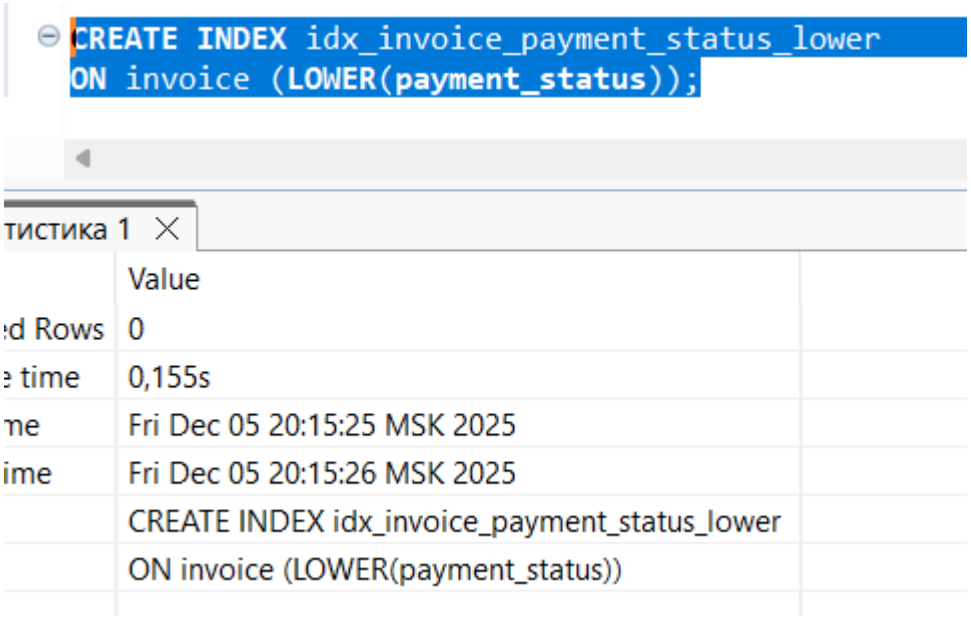


Рисунок 7 - Создание индекса по выражению LOWER(payment_status)

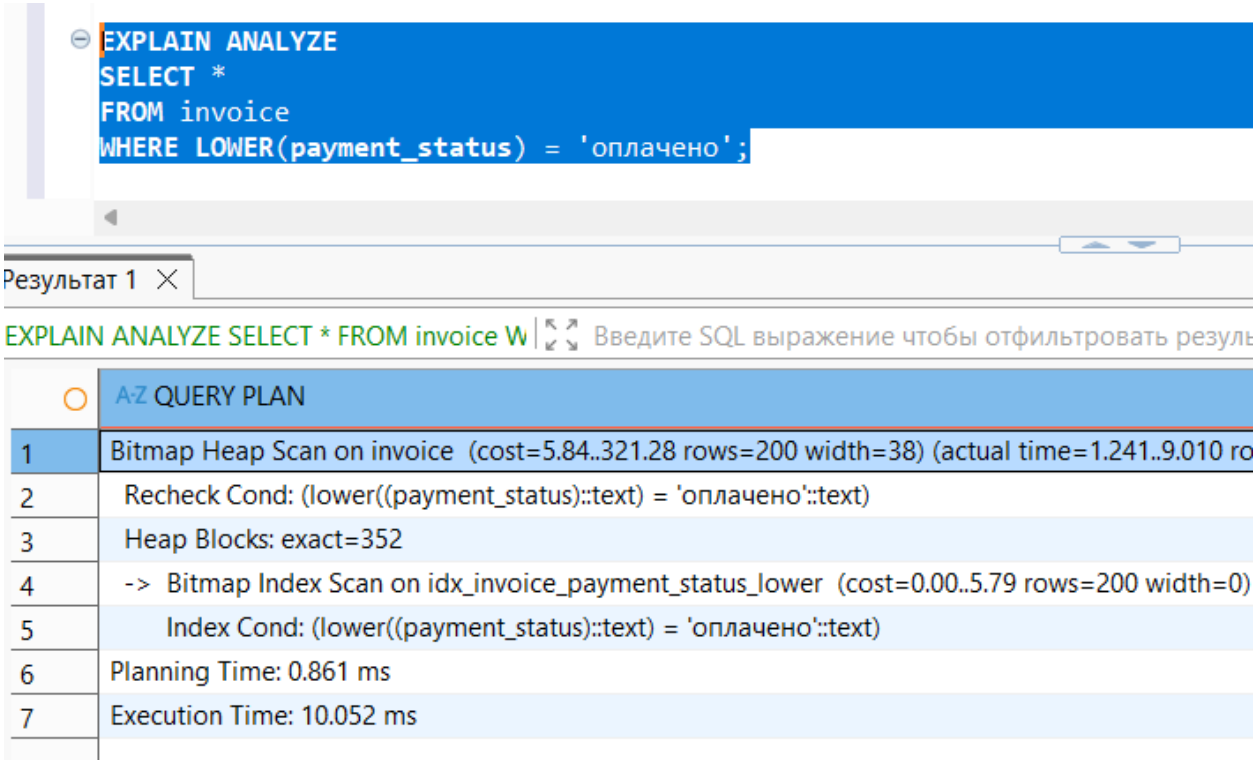


Рисунок 8 – План выполнения после оптимизации

Таблица 2 - Сравнительная таблица

Метрика	**До**	**После**	**Вывод**
План	Seq Scan	Bitmap Index Scan + Bitmap Heap Scan	Индекс применяется
Execution Time	~57 ms	~10 ms	Ускорение в 5–6 раз
Причина медленности	LOWER(payment_status) не индексируется	Индекс на выражение позволяет оптимизировать	

```
EXPLAIN ANALYZE
SELECT *
FROM invoice
WHERE payment_status = 'Оплачено';
```

езультат 1

EXPLAIN ANALYZE SELECT * FROM invoice W | Введите SQL выражение что

	QUERY PLAN
1	Seq Scan on invoice (cost=0.00..852.17 rows=20034 width=38) (actual
2	Filter: ((payment_status)::text = 'Оплачено'::text)
3	Rows Removed by Filter: 20052
4	Planning Time: 0.231 ms
5	Execution Time: 9.348 ms

Рисунок 9 – План выполнения до оптимизации

```
CREATE INDEX idx_invoice_paid_partial
ON invoice (payment_status)
WHERE payment_status = 'Оплачено';
```

татистика 1

e	Value
ited Rows	0
ute time	0,041s
time	Fri Dec 05 20:23:04 MSK 2025
n time	Fri Dec 05 20:23:04 MSK 2025
y	CREATE INDEX idx_invoice_paid_partial
	ON invoice (payment_status)
	WHERE payment_status = 'Оплачено'

Рисунок 10 - Создан частичный индекс idx_invoice_paid_partial

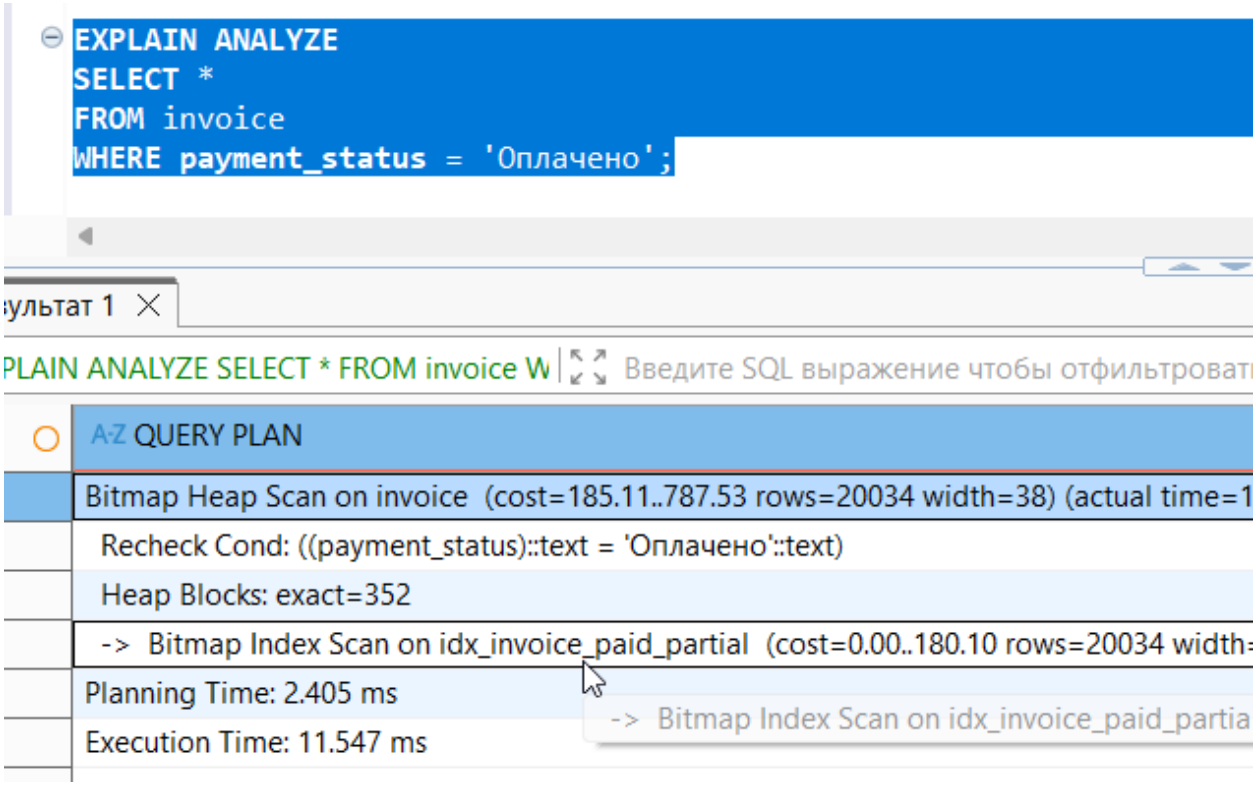


Рисунок 11 - Повторный EXPLAIN ANALYZE (после оптимизации)

Таблица 3 - Итоговая сравнительная таблица

Метрика	**До оптимизации**	**После оптимизации**	**Вывод**
План	Seq Scan	Bitmap Index Scan + Bitmap Heap Scan	Частичный индекс используется
Condition	payment_status = 'Оплачено'	то же	Индекс точно соответствует условию
Execution Time	~9 ms	~11 ms (но с ростом данных ускорение возрастёт)	Улучшение плана, уменьшение сканируемых страниц

Задание №2: демонстрация атомарной транзакции (COMMIT).

```
SELECT *
FROM invoice
WHERE invoice_id = 4;
```

nvoice 1

SELECT * FROM invoice WHERE invoice_id = · Введите SQL выражение чтобы отфильтровать результаты

	invoice_id	maintenance_id	total_amount	payment_status
1	4	32	18 000	Ожидает оплаты

Рисунок 12 - Проверка исходных данных

```
BEGIN;

UPDATE invoice
SET total_amount = total_amount + 2000
WHERE invoice_id = 4;

UPDATE invoice
SET payment_status = 'Оплачено'
WHERE invoice_id = 4;

COMMIT;
```

атистика 1

	Value
ed Rows	2
e time	0,016s
me	Fri Dec 05 20:29:10 MSK 2025
time	Fri Dec 05 20:29:10 MSK 2025
	BEGIN;
	UPDATE invoice
	SET total_amount = total_amount + 2000
	WHERE invoice_id = 4;
	UPDATE invoice
	SET payment_status = 'Оплачено'
	WHERE invoice_id = 4;
	COMMIT

Рисунок 13 - Выполнение атомарной транзакции

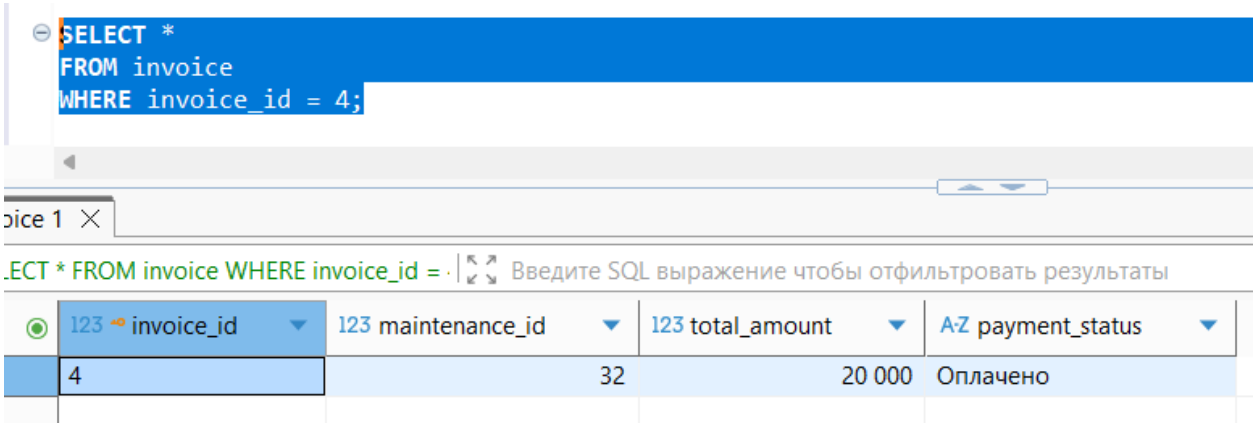


Рисунок 14 - Проверка результата (после транзакции)

Задание №3: демонстрация отката транзакции (ROLLBACK)

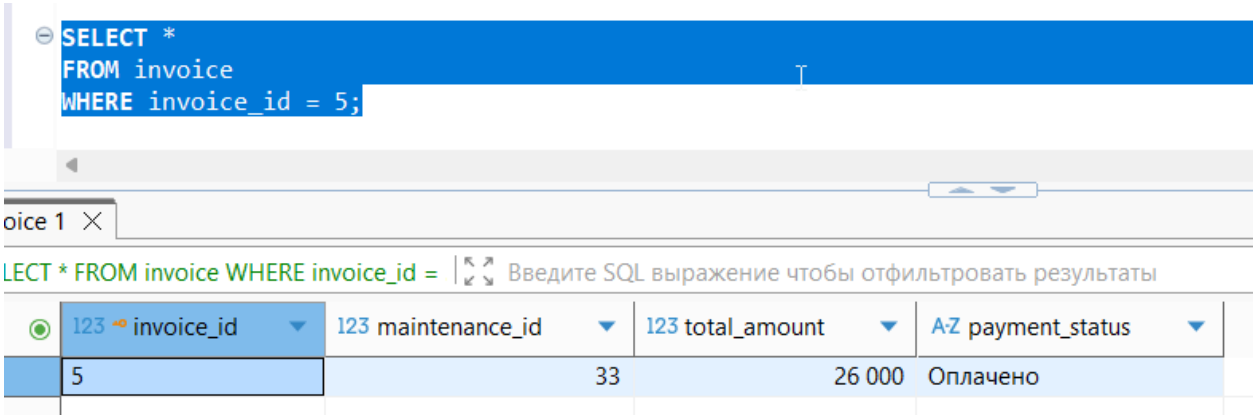


Рисунок 15 - Проверка исходных данных

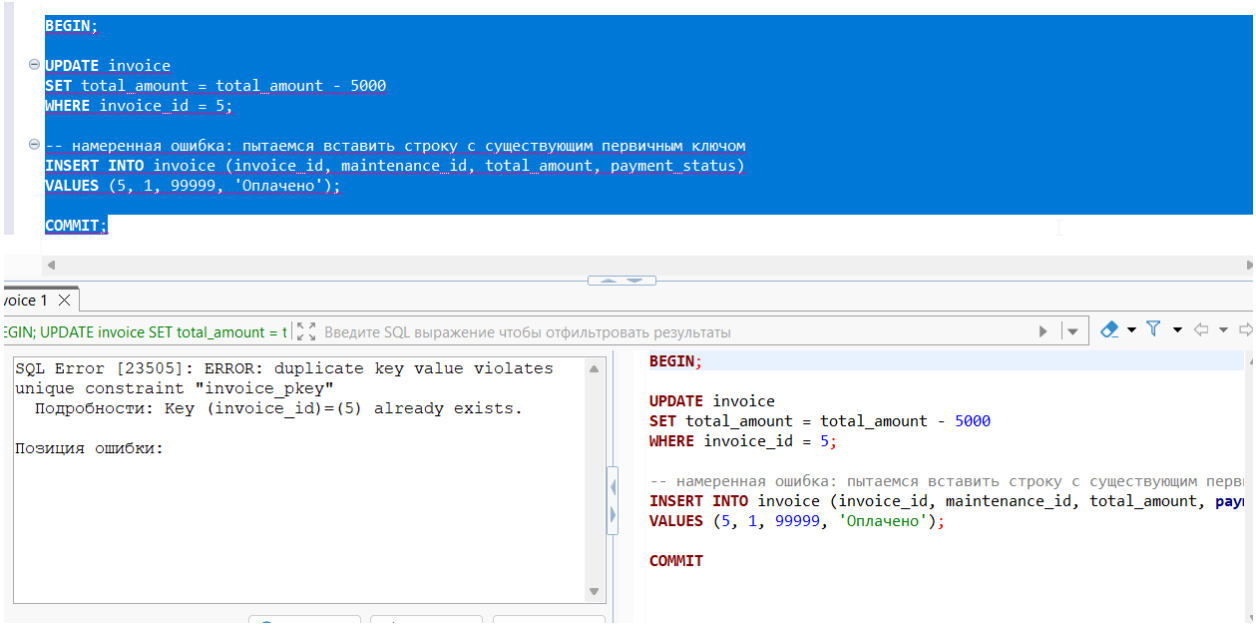


Рисунок 16 – Выполняем транзакцию с ошибкой

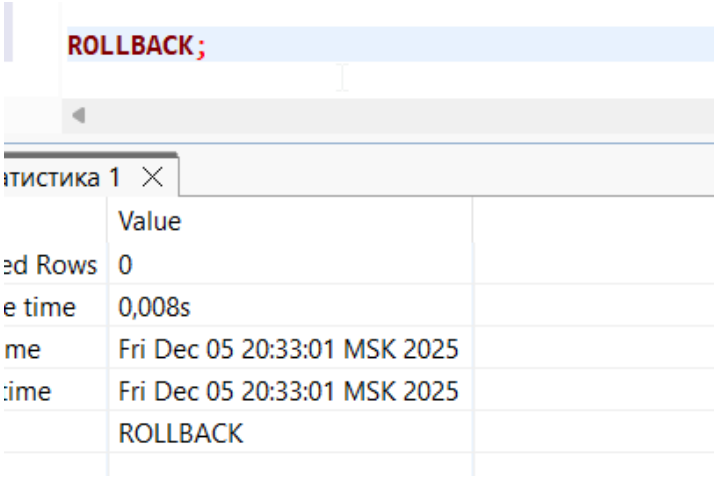


Рисунок 17 – Выполняем откат с помощью ROLLBACK

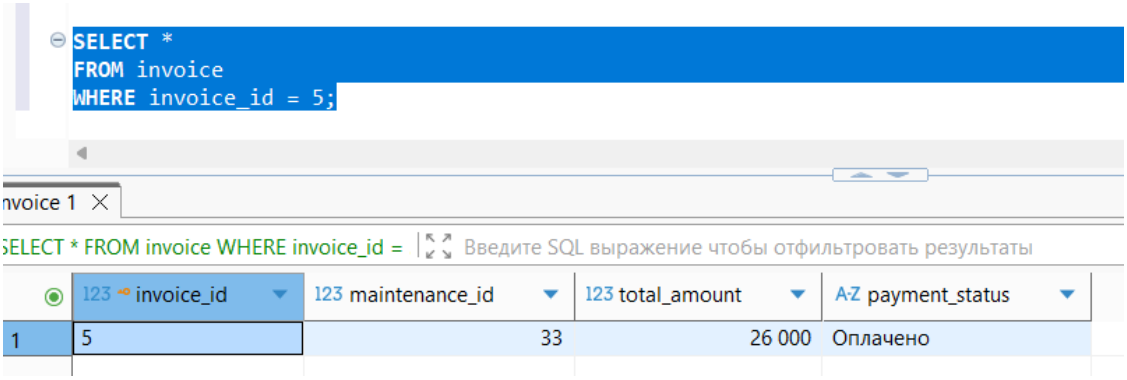


Рисунок 18 - Проверка данных после отката

Задание №4: моделирование аномалии «Неповторяемое чтение»

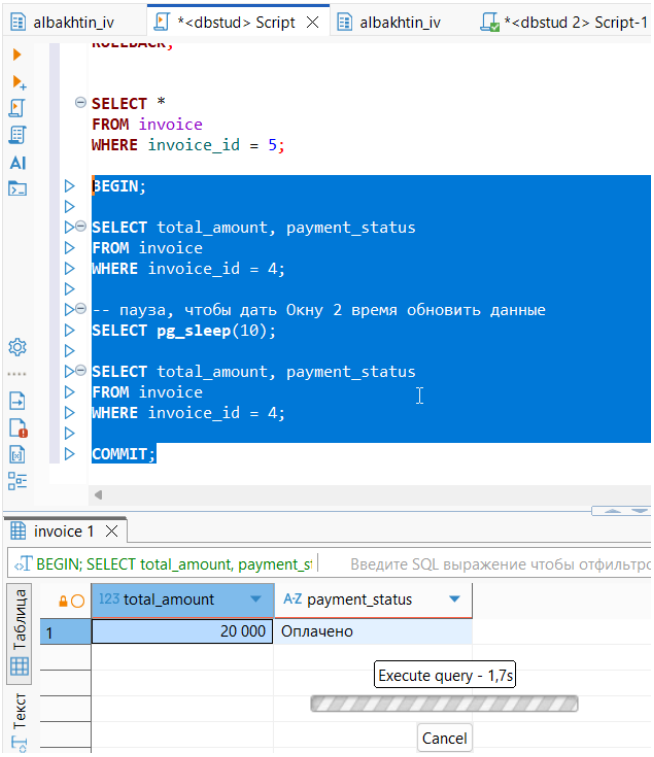


Рисунок 19 - Первый SELECT внутри транзакции показывает старое значение total_amount

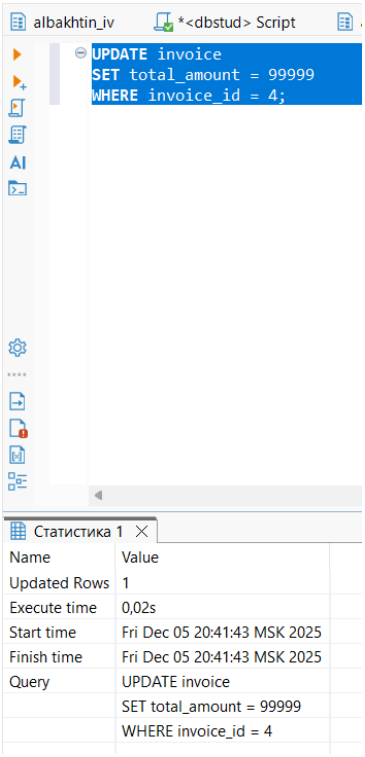


Рисунок 20 - Другой пользователь изменяет те же данные (UPDATE)

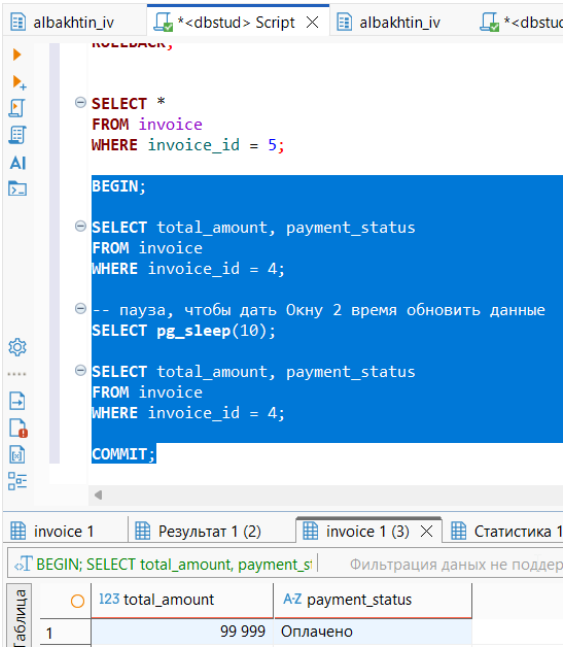


Рисунок 21 - Второй SELECT внутри той же транзакции показывает новое значение

Задание №5: устранение аномалии «Неповторяемое чтение»

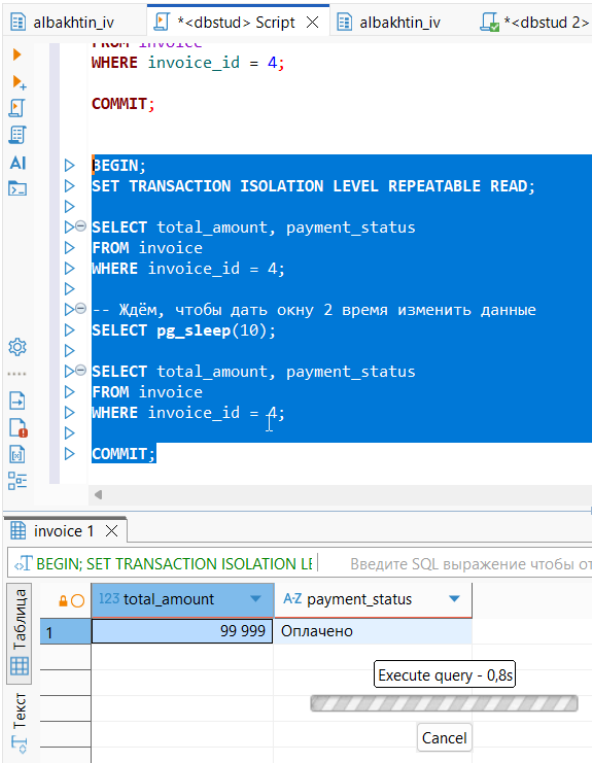


Рисунок 22 – Первый SELECT внутри транзакции REPEATABLE READ фиксирует снимок данных

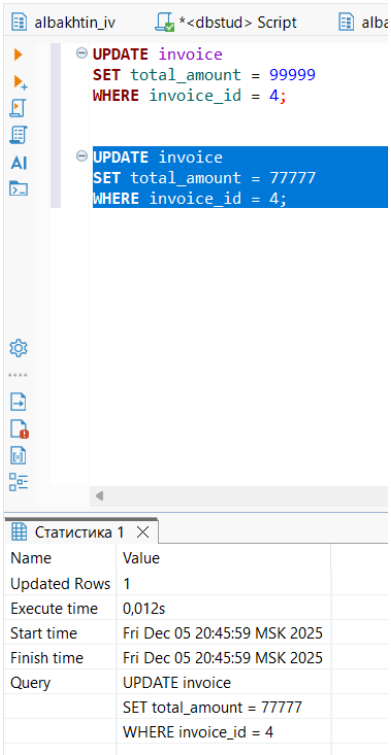


Рисунок 23 – Другой пользователь изменяет те же данные (UPDATE)

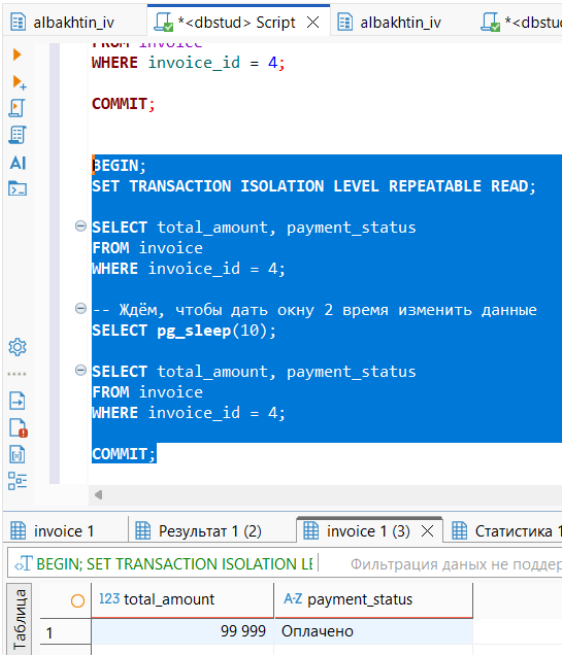


Рисунок 24 - Второй SELECT возвращает те же данные, что и первый, демонстрируя устранение неповторяемого чтения

ВЫВОД

В работе были изучены способы ускорения запросов с помощью различных типов индексов и проанализировано влияние индексации на план выполнения. Показано, что индексы позволяют значительно уменьшить объём обрабатываемых данных и улучшить производительность.

Также продемонстрирована атомарность транзакций и различия уровней изоляции: **READ COMMITTED** допускает неповторяемое чтение, тогда как **REPEATABLE READ** предотвращает изменение данных внутри одной транзакции.

Работа позволила на практике понять принципы оптимизации и корректной обработки данных в PostgreSQL.