**Федеральное государственное образовательное бюджетное учреждение**

**высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РФ»**

**Департамент анализа данных и машинного обучения**

**Отчет по практике №10**

по дисциплине «системы управления базами данных»

Студента группы ПМ23-1

Факультета информационных технологий и анализа больших данных

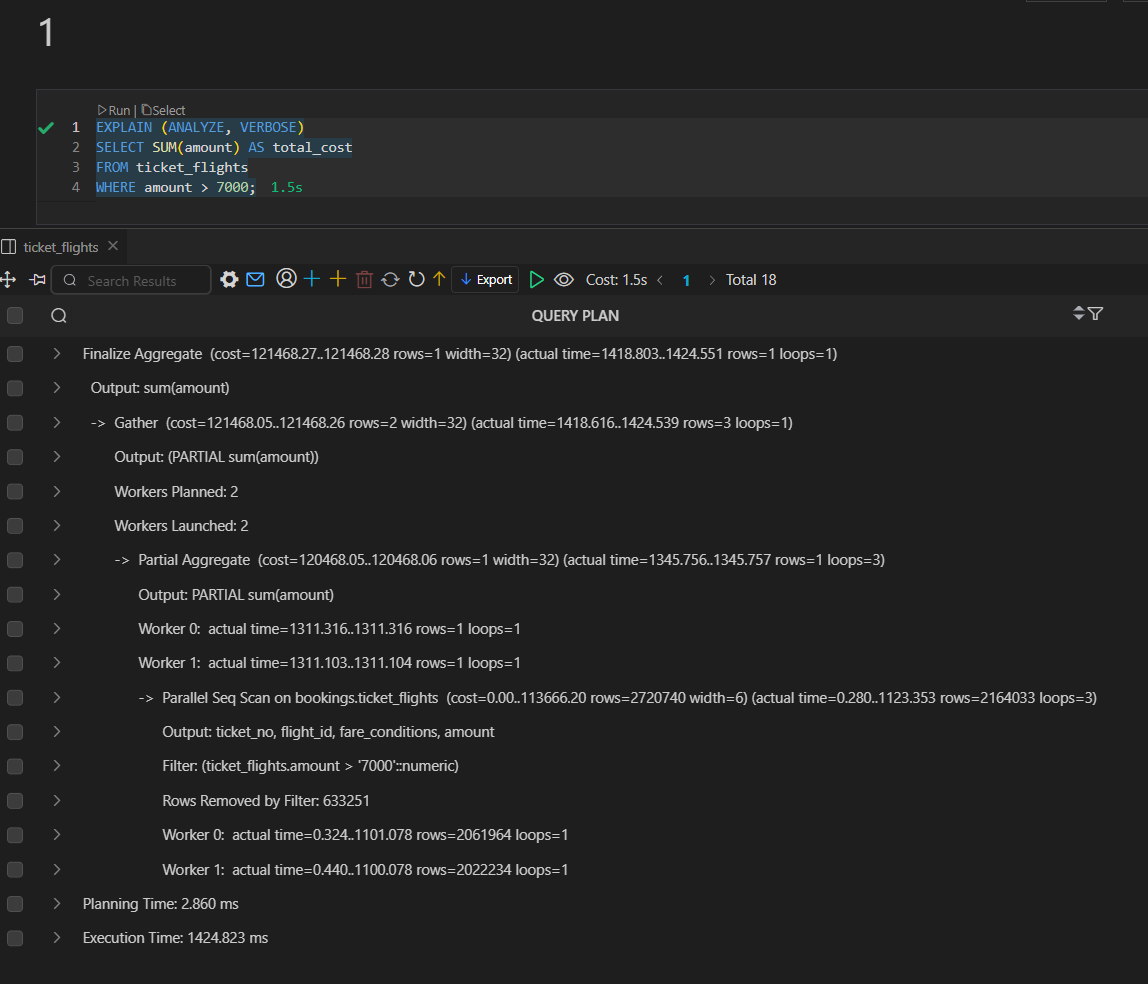
Тищенко И.С.

Преподаватель

Карамышев Е.П.

Москва 2025

# 1



Анализ элементов плана

- Finalize Aggregate:

- Завершающая агрегация результатов, собранных от параллельных рабочих процессов.

- Указывает, что агрегация распределена между несколькими процессами.

- Gather:

- Оператор сбора данных от параллельных рабочих процессов (Workers Launched: 2).

- Параметры Workers Planned и Workers Launched показывают запланированное и фактическое количество параллельных процессов.

- Partial Aggregate:

- Частичная агрегация данных на каждом рабочем процессе.

- Ускоряет обработку за счет распределения нагрузки.

- Parallel Seq Scan:

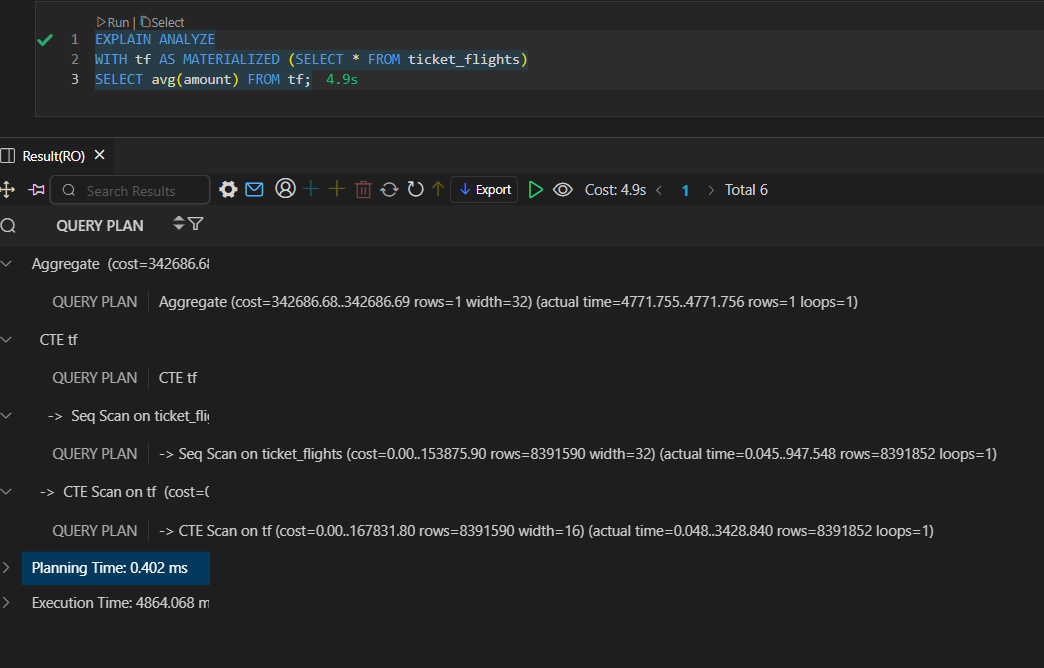
- Параллельное сканирование таблицы ticket\_flights.

- Фильтр Filter: (amount > 7000) применяется на уровне каждого рабочего процесса.

- Rows Removed by Filter:

- Количество строк, исключенных из обработки из-за условия amount > 7000.

# 2

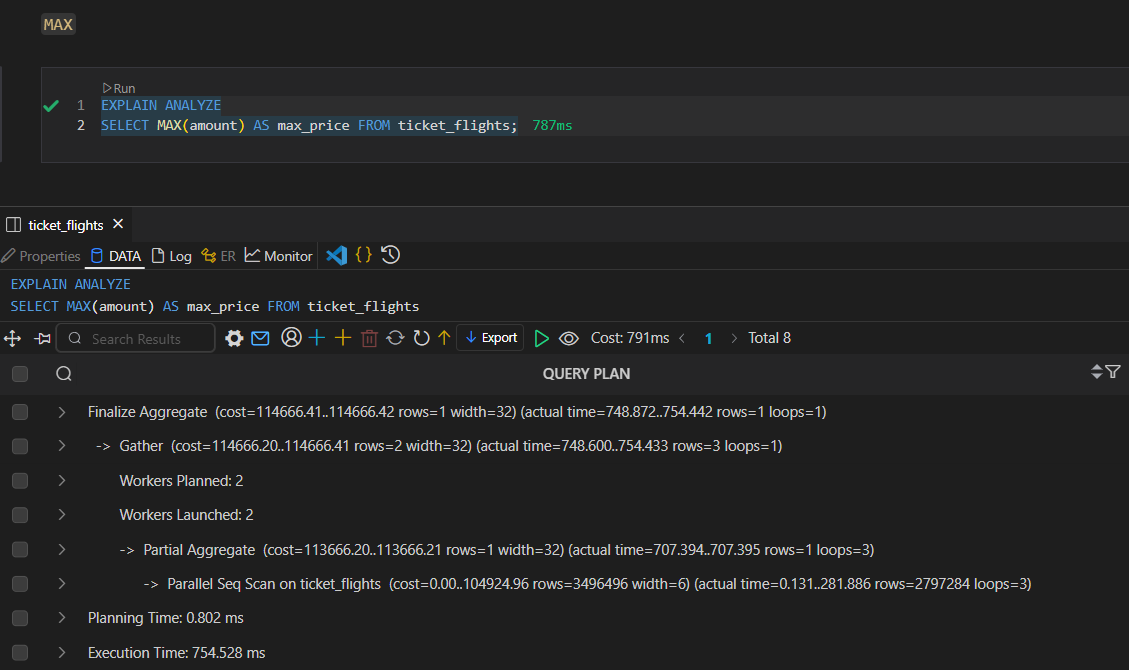


- Без force\_parallel\_mode: Запрос выполняется последовательно из-за материализации CTE. Данные из ticket\_flights читаются последовательно (Seq Scan), а агрегация происходит над готовым набором данных.

- С force\_parallel\_mode: Параллельность активируется, но только для этапа чтения данных из ticket\_flights (если настройки позволяют).

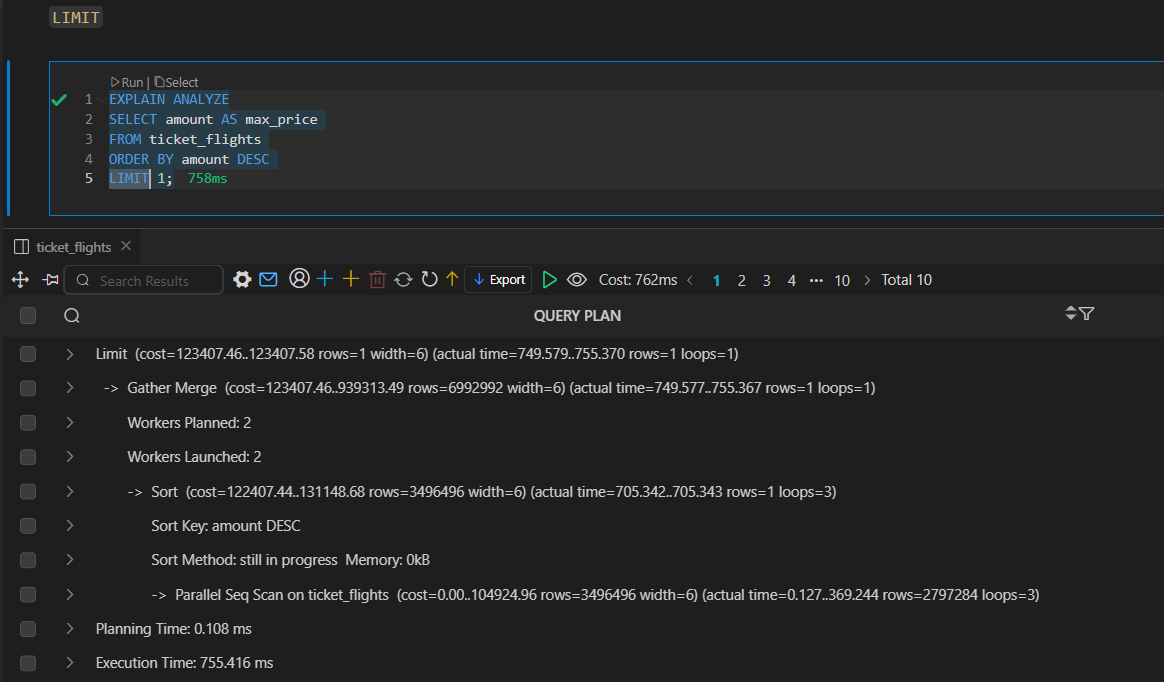
Это демонстрирует, что использование CTE с MATERIALIZED может ограничивать возможности параллелизма, так как требует промежуточного хранения данных. Для критических сценариев рекомендуется тестировать настройки и анализировать планы с помощью EXPLAIN ANALYZE.

# 3



Метод доступа: Seq Scan (последовательное сканирование таблицы).

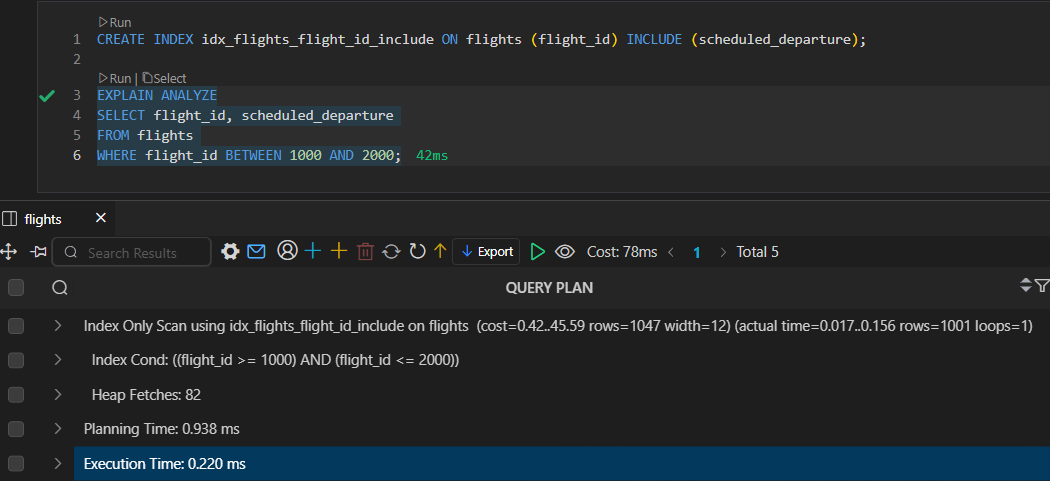
Эффективность: Низкая, так как требуется прочитать все строки таблицы для поиска максимального значения.

Метод доступа: Seq Scan + Sort.

Эффективность: менее эффективен, чем MAX(), так как требует сортировки всех строк.

**На практике разница меньше одной милисекунды**

# 4

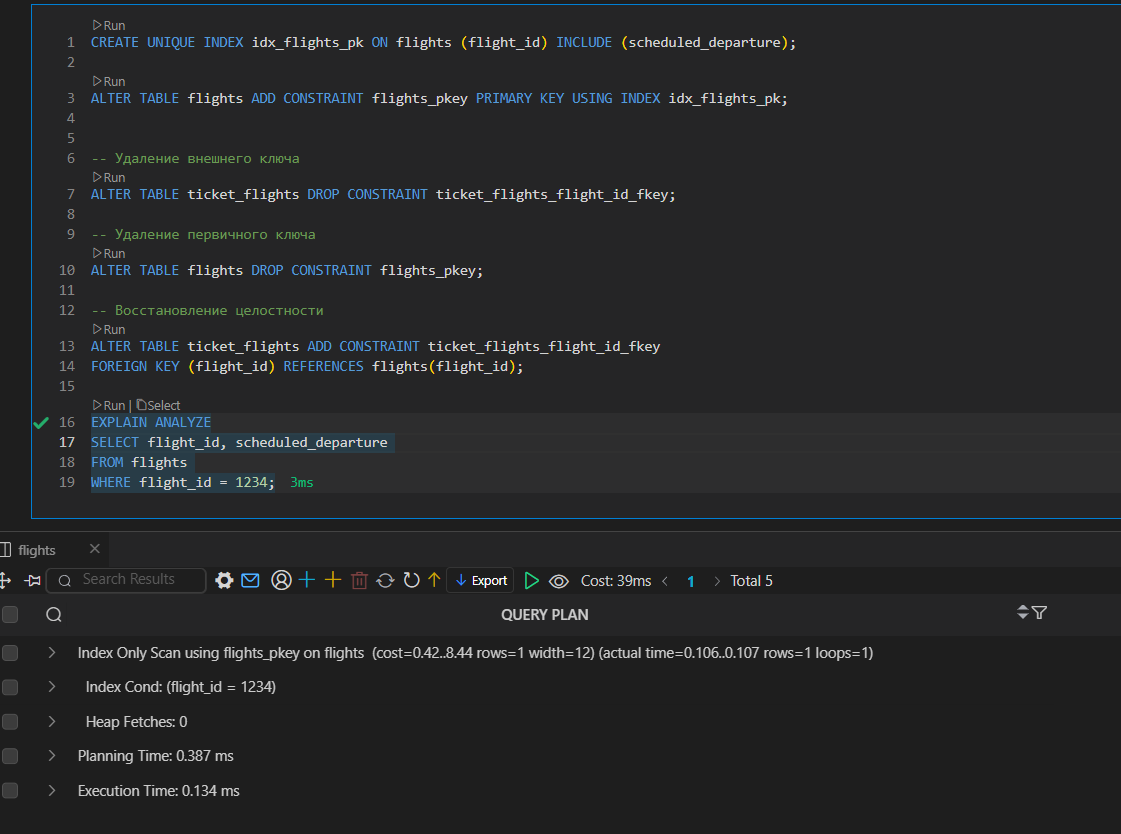
Объяснение элементов плана:

Index Scan: используется индекс idx\_flights\_flight\_id\_include.

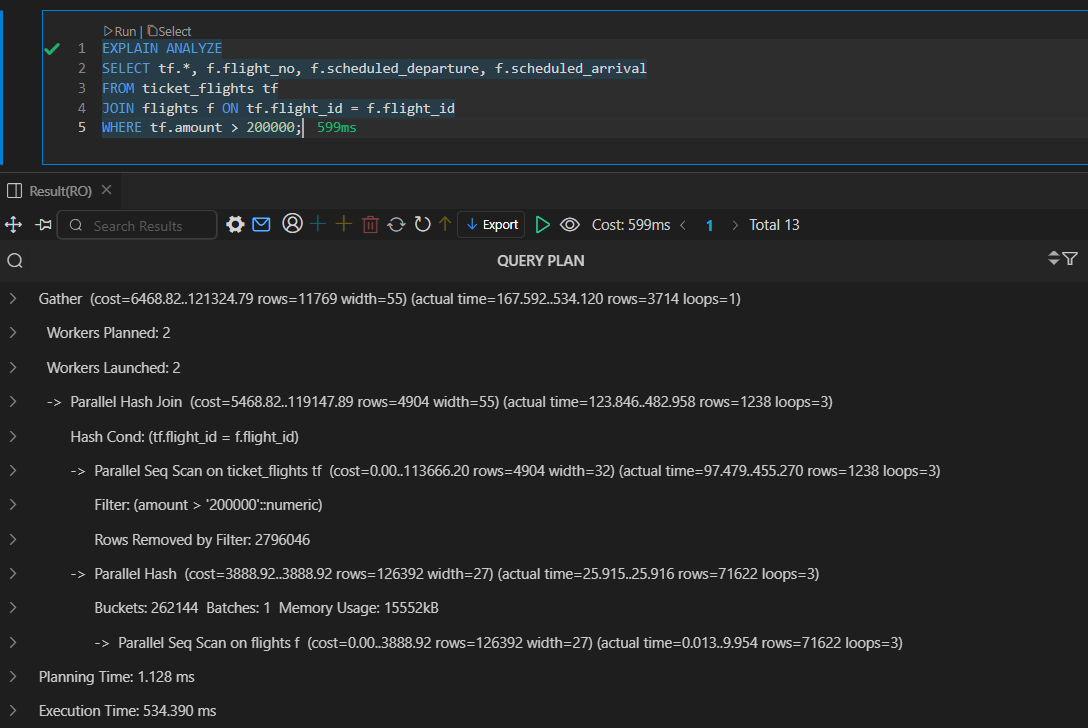
Index Cond: Условие фильтрации по диапазону flight\_id.

width=12: размер данных на строку (8 байт для flight\_id (integer) + 4 байта для scheduled\_departure (timestamp без часового пояса))

rows=1001: Оценка количества строк.



# 5



Анализ элементов плана:

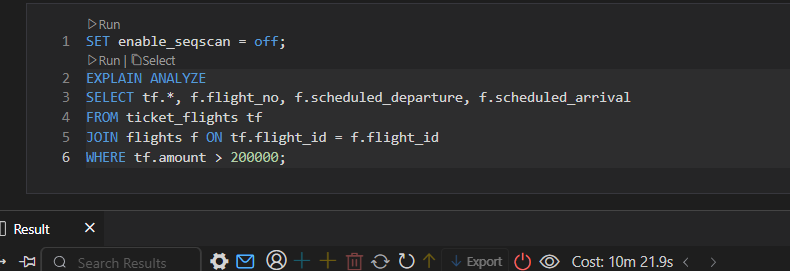
Seq Scan on ticket\_flights: используется последовательное сканирование таблицы ticket\_flights, так как отсутствует индекс на столбце amount.

Index Scan on flights: используется индекс flights\_pkey для соединения с таблицей flights.

Rows Removed by Filter: условие amount > 200000 исключает 2 796 046 строк.

Nested Loop: Соединение таблиц через цикл, что неэффективно при больших объемах данных.

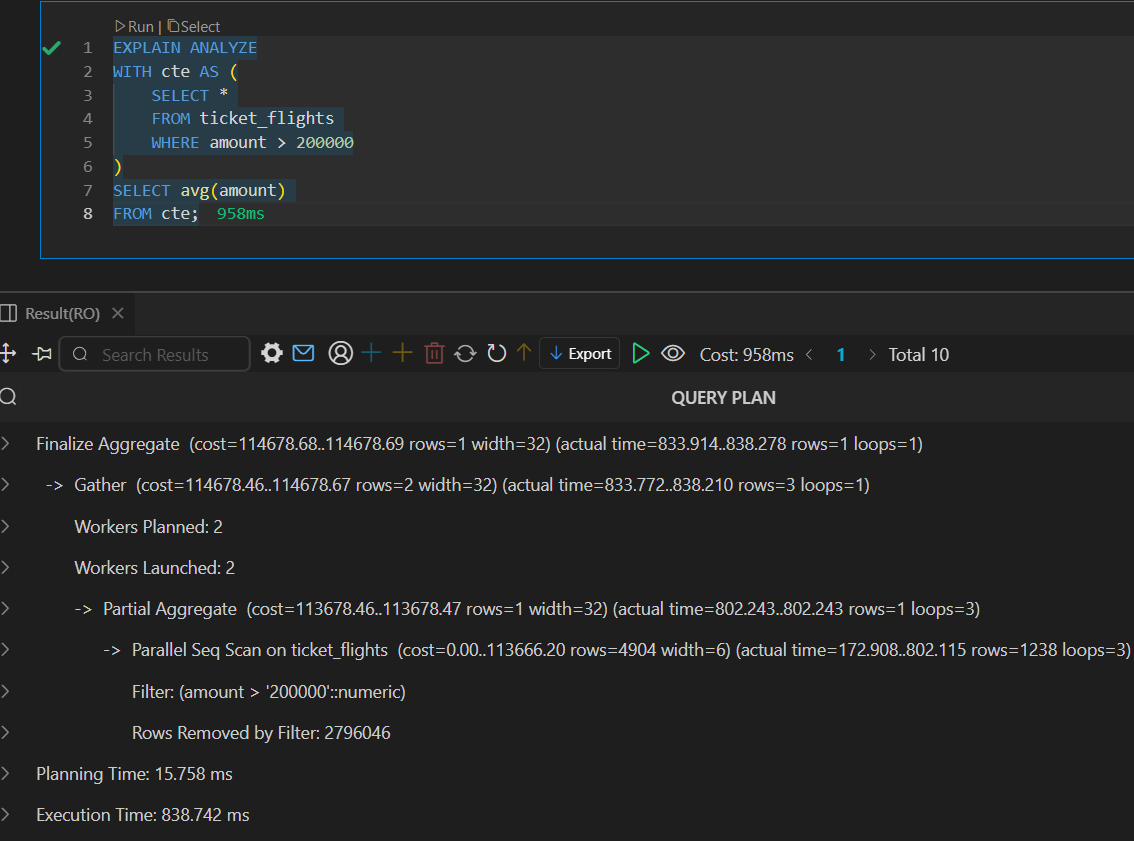
Метод доступа: Seq Scan для ticket\_flights.



С выключенным Seq Scan запрос выполняется уже более 10 минут

# 6

## 3

CTE материализуется в отдельный этап плана (CTE Scan), что позволяет избежать повторных вычислений.

## 4

Анализ плана выполнения запроса

Объяснение узлов и параметров

1. Узел Seq Scan on bookings

- Описание: Последовательное сканирование всей таблицы bookings.

- Параметры:

- cost=0.00..4301.88:

- 0.00 — стартовые затраты (время до начала сканирования).

- 4301.88 — общие затраты на сканирование всей таблицы.

- rows=262788: Оценка количества строк, возвращаемых из таблицы.

- width=6: Средний размер одной строки в байтах (тип numeric(10,2) занимает 6 байт).

2. Узел Sort

- Описание: Сортировка всех строк по убыванию total\_amount.

- Параметры:

- cost=8666.69..9323.66:

- 8666.69 — затраты на подготовку сортировки (включая сканирование таблицы).

- 9323.66 — общие затраты на сортировку всех 262,788 строк.

- rows=262788: Количество строк, участвующих в сортировке.

- Sort Key: total\_amount DESC: Ключ сортировки — столбец total\_amount в порядке убывания.

3. Узел Limit

- Описание: Ограничение результата до 5 строк.

- Параметры:

- cost=8666.69..8666.71:

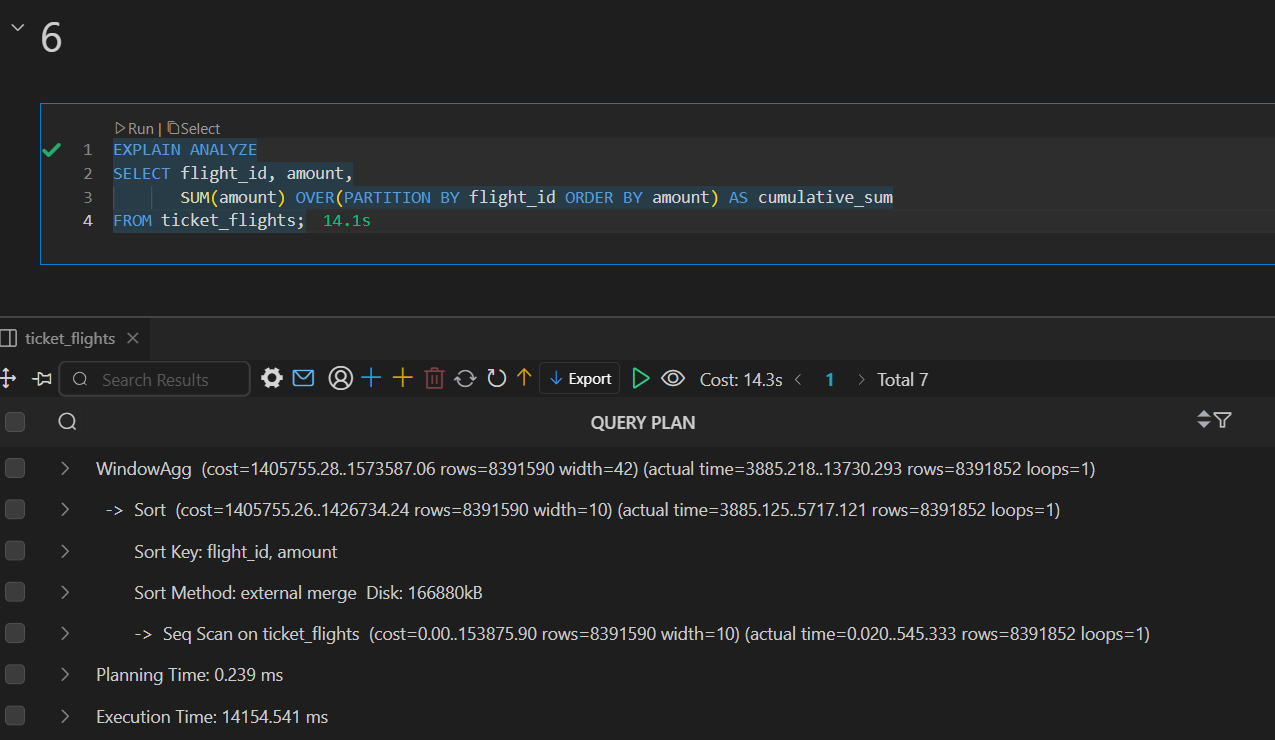
- 8666.69 — затраты до начала операции Limit.

- 8666.71 — общие затраты на выполнение запроса.

- rows=5: Ожидаемое количество строк в результате.

- width=6: Размер строки в байтах (аналогично предыдущим узлам).

## 6



Почему WindowAgg находится на этом уровне?

1. Последовательность обработки:

- Оконные функции требуют предварительной подготовки данных: группировки (PARTITION BY) и сортировки (ORDER BY).

- В данном случае:

- PARTITION BY flight\_id: Группировка по flight\_id.

- ORDER BY amount: Сортировка внутри каждой группы.

- Эти этапы (Seq Scan → Sort) готовят данные для WindowAgg.

2. Роль сортировки:

- Оконные функции, использующие ORDER BY, требуют упорядоченных данных для вычисления агрегатов (например, накопительной суммы).

- Sort гарантирует, что строки в каждом разделе flight\_id упорядочены по amount.

3. Иерархия узлов:

- Seq Scan → Sort → WindowAgg:

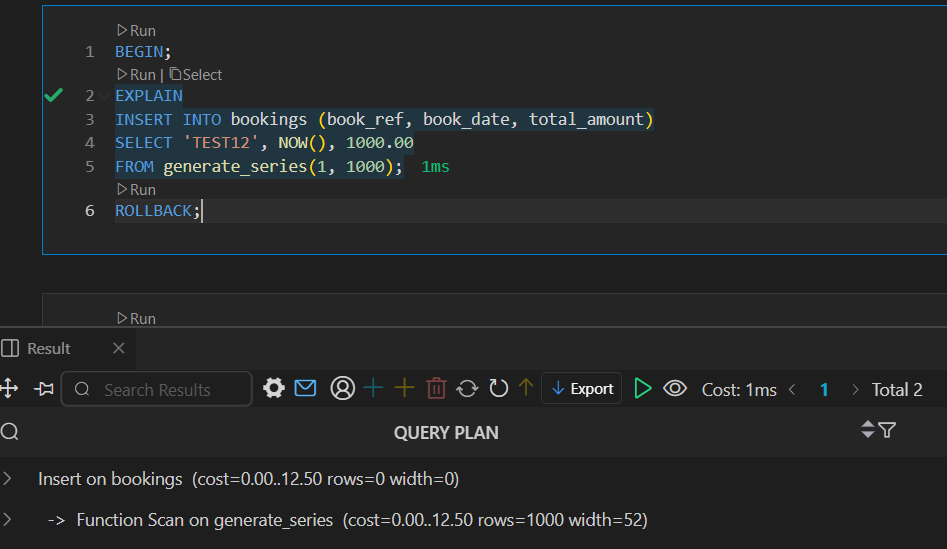
- Сначала данные читаются из таблицы (Seq Scan).

- Затем они группируются и сортируются (Sort).

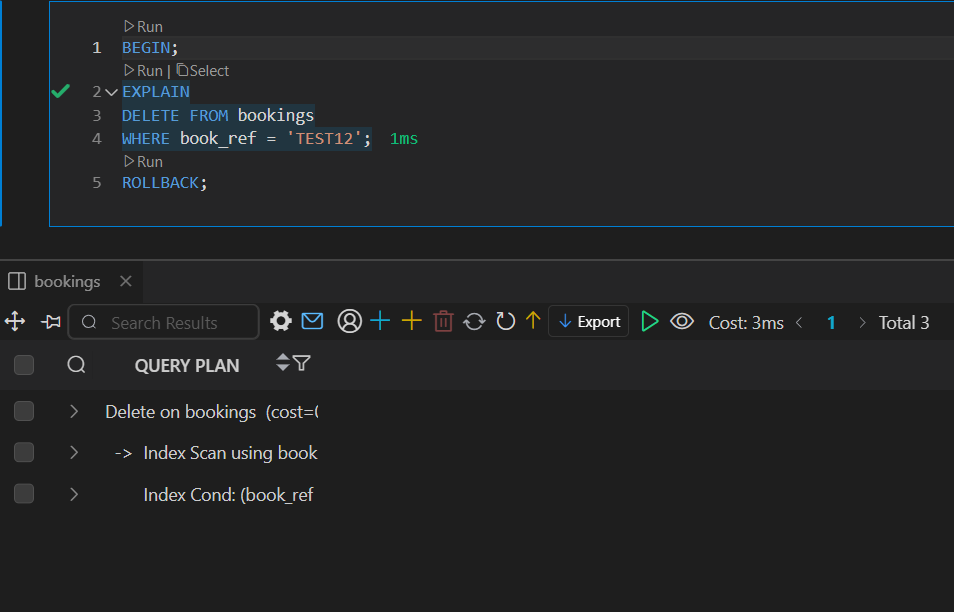
- Наконец, к отсортированным данным применяется оконная функция (WindowAgg).

## 7

**insert**



**Delete**



## 14

### Задание 1

Индекс nulls\_ind был создан с ASC (по умолчанию). При запросе ORDER BY num DESC NULLS FIRST:

1. DESC — требует обратного сканирования индекса (ключевое слово Backward в плане).

2. NULLS FIRST — совпадает с поведением по умолчанию для DESC, так как NULL считается "больше" любых значений. Поэтому индекс может использоваться.

Ожидаемый план:

Index Scan Backward using nulls\_ind on nulls (cost=0.42..5852.42 rows=200000 width=14)

- Index Scan Backward: Указывает на использование индекса в обратном порядке (для DESC).

- NULLS FIRST: Не требует дополнительной сортировки, так как NULL уже обрабатывается как "максимальное" значение в DESC.

- Вывод: Индекс используется. Ключевое слово Backward подтверждает обратное сканирование индекса для DESC. Так как NULLS FIRST совпадает с поведением по умолчанию для DESC, дополнительная сортировка не требуется .

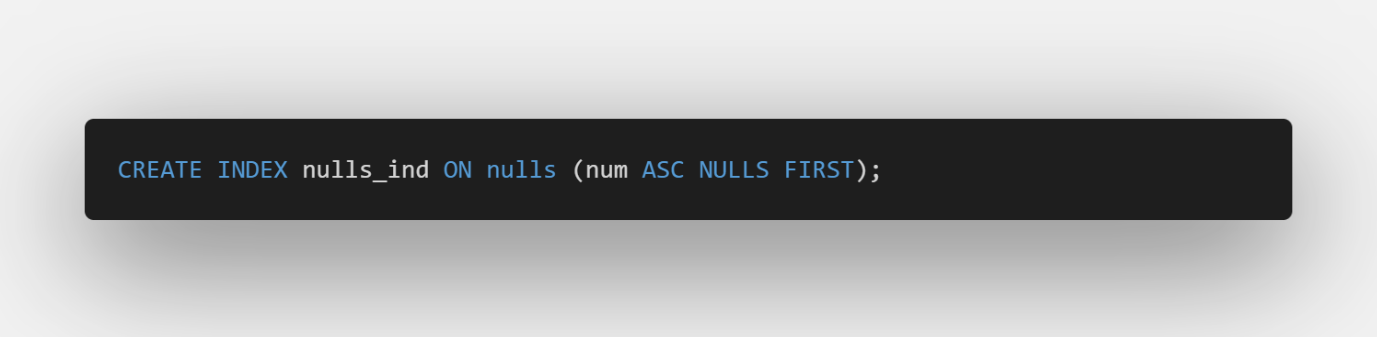
### Задание 2

Проблема

По умолчанию индекс nulls\_ind создан как ASC, где NULL обрабатывается как "больше" любых значений. Для ORDER BY num NULLS FIRST требуется, чтобы NULL шли первыми, но в ASC это противоречит стандартному поведению PostgreSQL.

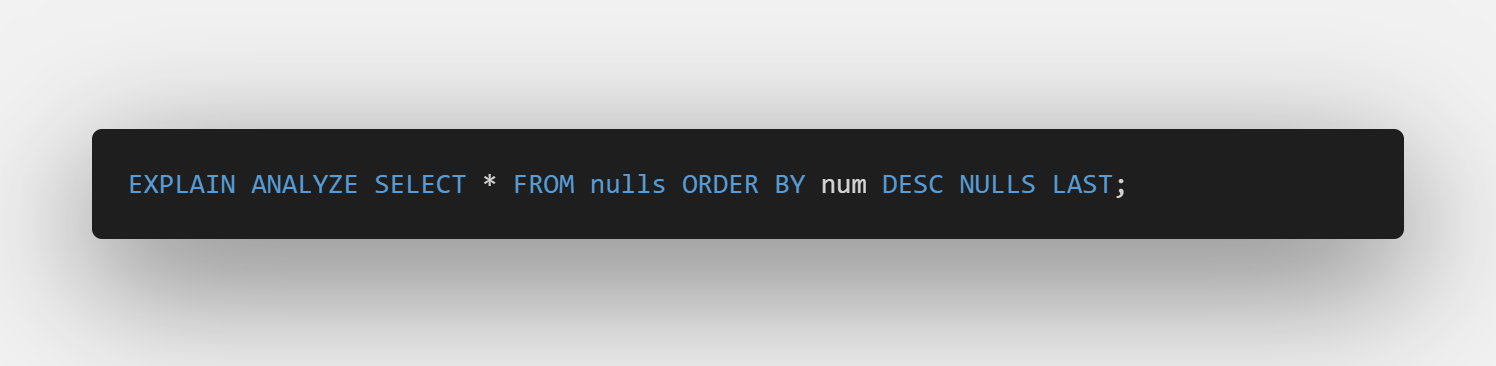
Решение

Создадим индекс с явным указанием ASC NULLS FIRST:



Вывод: Индекс nulls\_ind теперь поддерживает сортировку ASC NULLS FIRST, и планировщик использует его для запроса ORDER BY num NULLS FIRST .

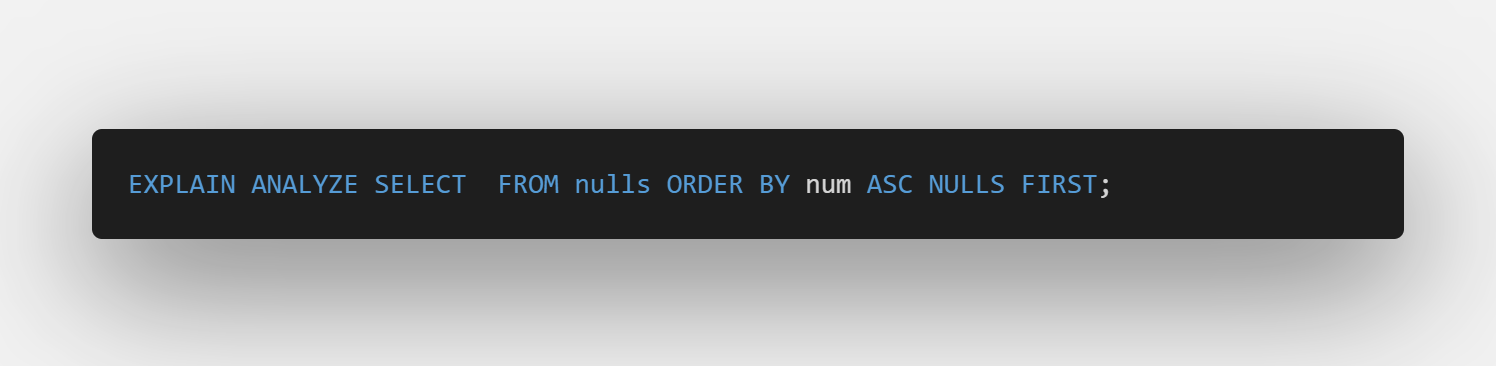
### Задание 3

Эксперимент 1: ORDER BY num DESC NULLS LAST

- Объяснение: NULLS LAST совпадает с поведением по умолчанию для DESC, поэтому индекс используется без дополнительной сортировки .

- Время выполнения: Минимальное, так как данные читаются из индекса.

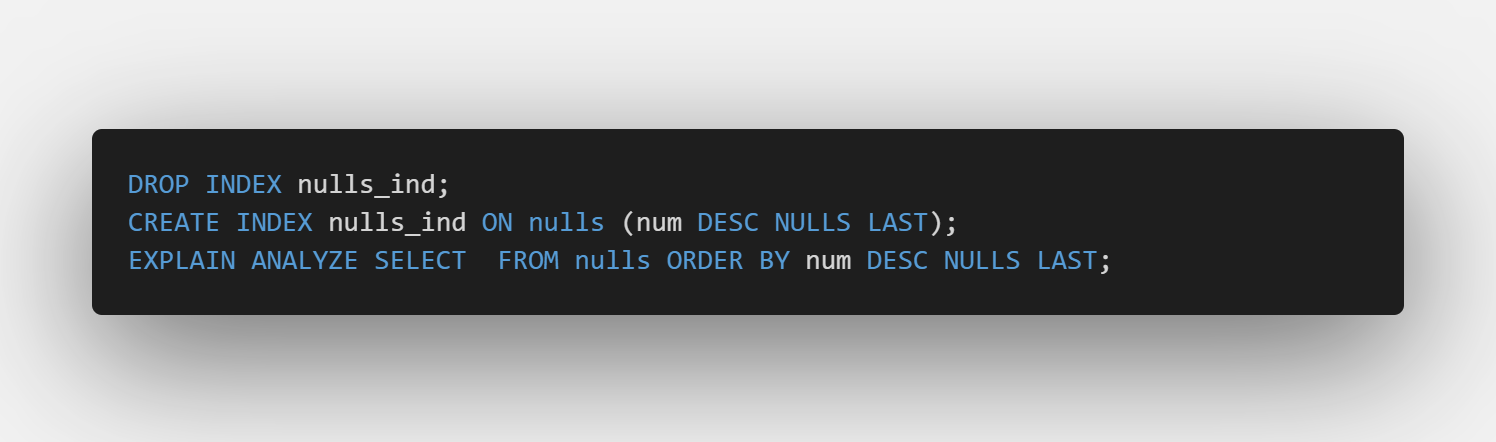
Эксперимент 2: ORDER BY num ASC NULLS FIRST



- Объяснение: Без индекса ASC NULLS FIRST требует сортировки, так как стандартный индекс ASC не поддерживает такое расположение NULL .

- Время выполнения: Высокое из-за Sort.

Эксперимент 3: Индекс с DESC NULLS LAST



- Вывод: Индекс DESC NULLS LAST полностью совпадает с запросом, поэтому используется без сортировки.