# ЛАБОРАТОРНАЯ РАБОТА №9 «ОПТИМИЗАЦИЯ ЗАПРОСОВ SQL»

## Цель работы

Исследование и оптимизация производительности запросов к базе данных на примере демонстрационной базы данных «авиаперелёты» от PostgresPro.

## Постановка задачи

Выполните запросы без оптимизации и с оптимизацией. А также выполните запросы используя индексы B-tree и хеш-индексы, проведя собственное исследование. Каждый вариант содержит по два запроса SQL.

Запрос №9. Найти пассажиров, которые летали на рейсах с наибольшим количеством разных дней недели.

## Ход работы

Был рассмотрен неоптимизированный запрос для отображения пассажиров, которые летали на рейсах с наибольшим количеством разных моделей самолётов. Запрос представлен в листинге 9.1.

Неоптимизированный запрос

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT f.aircraft\_code)AS unique\_aircraft\_models

FROM tickets t

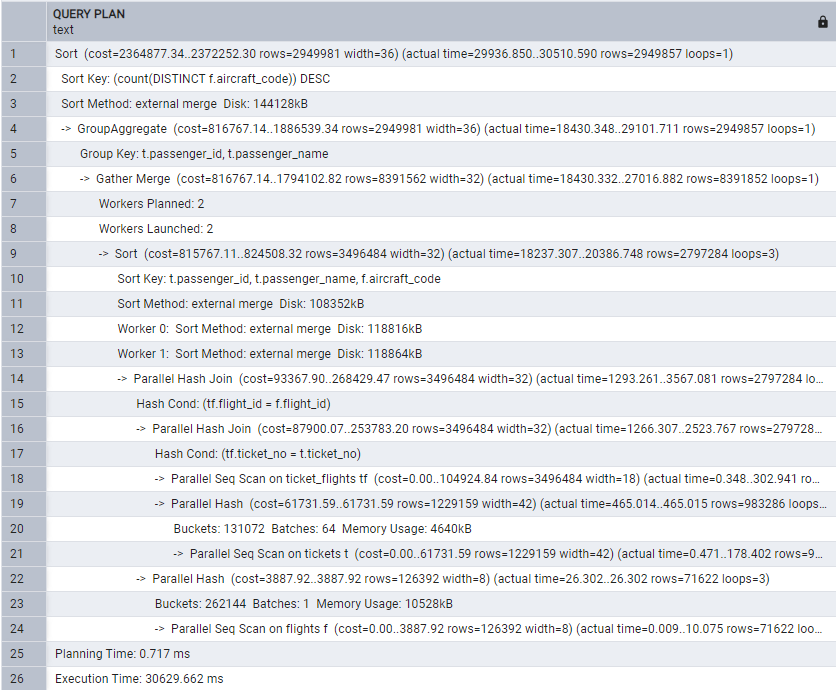
JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

JOIN flights f ON tf.flight\_id = f.flight\_id

GROUP BY t.passenger\_id, t.passenger\_name

ORDER BY unique\_aircraft\_models DESC;

На рисунке 9.1 представлен план выполнения неоптимизированного запроса.



* + - * 1. План выполнения неоптимизированного запроса

Как видно на рисунке 9.1, на выполнения запроса потребовалось ~30629.63мс. Попробуем оптимизировать запрос, чтобы сократить время выполнения.

В листинге 9.2. представлен запрос, решающий ту же задачу, но с использованием методов оптимизации.

Оптимизированный запрос

WITH passenger\_aircrafts AS (

SELECT t.passenger\_id, t.passenger\_name, f.aircraft\_code

FROM tickets t

JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

JOIN flights f ON tf.flight\_id = f.flight\_id

GROUP BY t.passenger\_id, t.passenger\_name, f.aircraft\_code)

SELECT passenger\_id, passenger\_name, COUNT(aircraft\_code)AS unique\_aircraft\_models

FROM passenger\_aircrafts

GROUP BY passenger\_id, passenger\_name

HAVING COUNT(departure\_airport) > 5

ORDER BY unique\_departure\_airports DESC;

На рисунке 9.2 представлен план выполнения оптимизированного запроса.



* + - * 1. План выполнения оптимизированного запроса

Как видно на рисунке 9.2, время, необходимое на выполнение запроса сократилось до ~17456.72мс.

Попробуем оптимизировать запрос используя для этого индексы. В листинге 9.3 представлен оптимизированный с помощью индексов запрос.

Создание индексов и запрос с ними

-- B-tree:

CREATE INDEX idx\_tickets\_passenger\_id ON tickets USING btree(passenger\_id);

CREATE INDEX idx\_ticket\_flights\_ticket\_no ON ticket\_flights USING btree(ticket\_no);

CREATE INDEX idx\_flights\_flight\_id ON flights USING btree (flight\_id);

CREATE INDEX idx\_flights\_aircraft\_code ON flights USING btree(aircraft\_code);

SET enable\_seqscan = OFF;

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT f.aircraft\_code)AS unique\_aircraft\_models

FROM tickets t

JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

JOIN flights f ON tf.flight\_id = f.flight\_id

WHERE f.aircraft\_code IS NOT NULL

GROUP BY t.passenger\_id, t.passenger\_name

ORDER BY unique\_aircraft\_models DESC;

В данном запросе создаются четыре индекса на основе B-деревьев. Запрос выполняет ту же задачу, что и два предыдущих.

План выполнения запроса представлен на рисунке 9.3.



* + - * 1. План выполнения запроса с индексами на основе B-деревьев

В листинге 9.4. представлен схожий запрос, но с использованием индексов на основе хэш-таблиц.

Создание хэш индексов и запрос сними

-- Хеш-таблицы:

CREATE INDEX idx\_tickets\_passenger\_id\_hash ON tickets USING hash(passenger\_id);

CREATE INDEX idx\_ticket\_flights\_ticket\_no\_hash ON ticket\_flights USING hash (ticket\_no);

CREATE INDEX idx\_flights\_flight\_id\_hash ON flights USING hash(flight\_id);

CREATE INDEX idx\_flights\_aircraft\_code\_hash ON flights USING hash(aircraft\_code);

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT f.aircraft\_code)AS unique\_aircraft\_models

FROM tickets t

JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

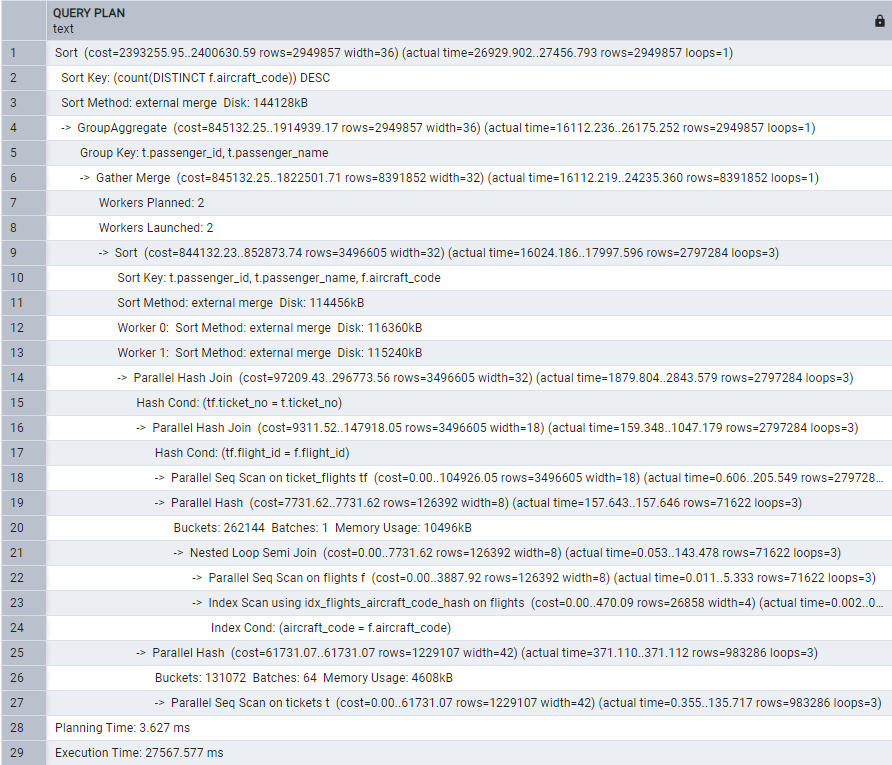
JOIN flights f ON tf.flight\_id = f.flight\_id

WHERE f.aircraft\_code = ANY(SELECT aircraft\_code FROM flights)

GROUP BY t.passenger\_id, t.passenger\_name

ORDER BY unique\_aircraft\_models DESC;

План выполнения запроса представлен на рисунке 9.4.



* + - * 1. План выполнения запроса с использованием хэш индексов

Рассмотрим другой запрос. В листинге 9.5. представлен неоптимизированный запрос, который анализирует в пассажиров, которые летали на рейсах с наибольшим количеством разных временных зон.

Неоптимизированный запрос

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT a.timezone)AS unique\_timezones

FROM tickets t

JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

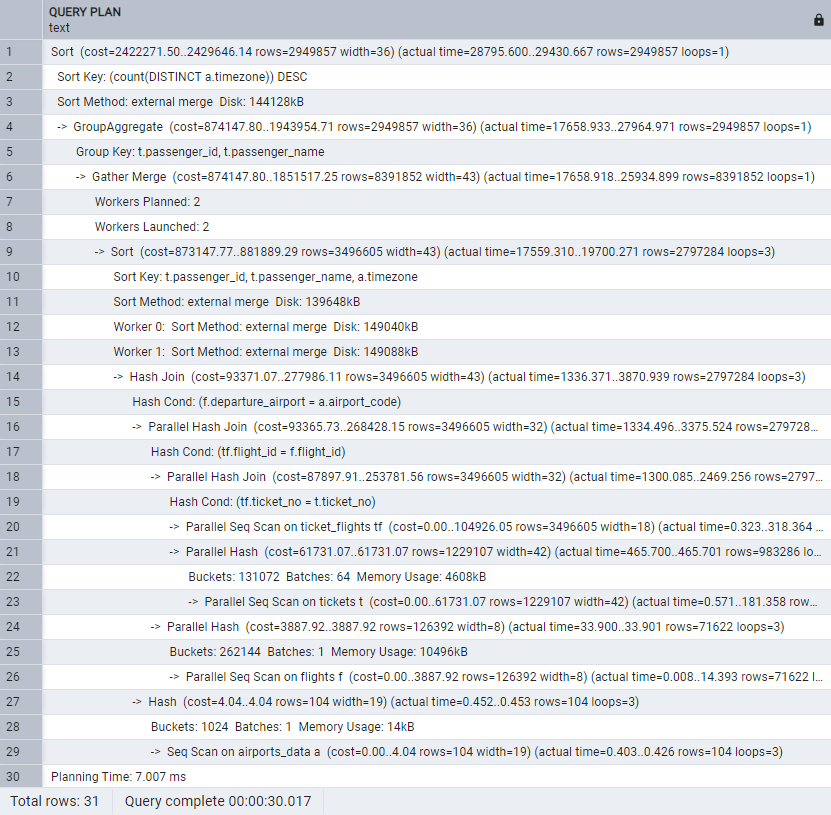
JOIN flights f ON tf.flight\_id = f.flight\_id

JOIN airports\_data a ON f.departure\_airport = a.airport\_code G

ROUP BY t.passenger\_id, t.passenger\_name

ORDER BY unique\_timezones DESC;

План выполнения запроса представлен на рисунке 9.5.



* + - * 1. План выполнения неоптимизированного запроса

Как видно на рисунке 9.5, на выполнения такого запроса потребовалось ~ 29538.88мс. Оптимизируем запрос и сравним время.

В листинге 9.6. представлен оптимизированный запрос, решающий эту задачу.

Оптимизированный запрос

EXPLAIN ANALYZE

WITH passenger\_timezones AS (

SELECT t.passenger\_id, t.passenger\_name, a.timezone

FROM tickets t JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

JOIN flights f ON tf.flight\_id = f.flight\_id

JOIN airports\_data a ON f.departure\_airport = a.airport\_code

GROUP BY t.passenger\_id, t.passenger\_name, a.timezone)

SELECT passenger\_id, passenger\_name, COUNT(timezone) AS unique\_timezones

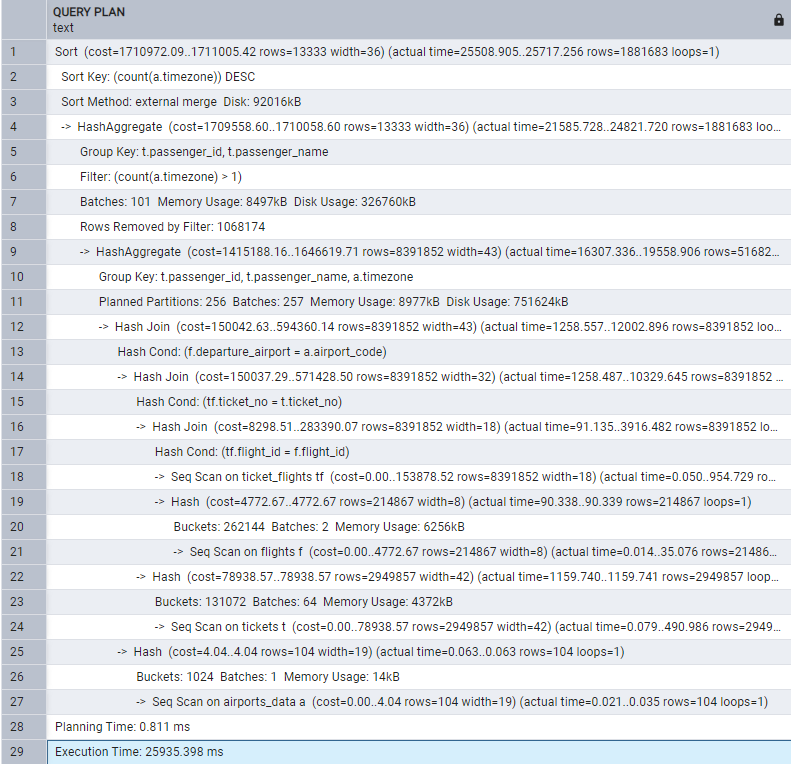
FROM passenger\_timezones

GROUP BY passenger\_id, passenger\_name

HAVING COUNT(timezone) > 1

ORDER BY unique\_timezones DESC;

План выполнения запроса представлен на рисунке 9.6.



* + - * 1. План выполнения оптимизированного запроса

Как видно на рисунке 9.6. время выполнения запроса сократилось до ~25935.40мс, время планирования также сократилось.

Создадим индексы на основе B-деревьев и оптимизируем запрос с их помощью. Запрос с использованием индексов на основе B-деревьев представлен в листинге 9.7.

Запрос с использованием индексов на основе B-tree

SET enable\_seqscan = OFF;

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT a.timezone)AS unique\_timezones

FROM tickets t

JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

JOIN flights f ON tf.flight\_id = f.flight\_id

JOIN airports\_data a ON f.departure\_airport = a.airport\_code

WHERE a.timezone IS NOT NULL

GROUP BY t.passenger\_id, t.passenger\_name

ORDER BY unique\_timezones DESC;

План выполнения запроса представлен на рисунке 9.7.



* + - * 1. План выполнения запроса с использованием индексов

Для сравнения в похожем запросе используем хэш-индексы. Запрос с использованием индексов на основе хэш-таблиц представлен в листинге 9.8.

Запрос с использованием хэш-индексов

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT a.timezone)AS unique\_timezones

FROM tickets t

JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

JOIN flights f ON tf.flight\_id = f.flight\_id

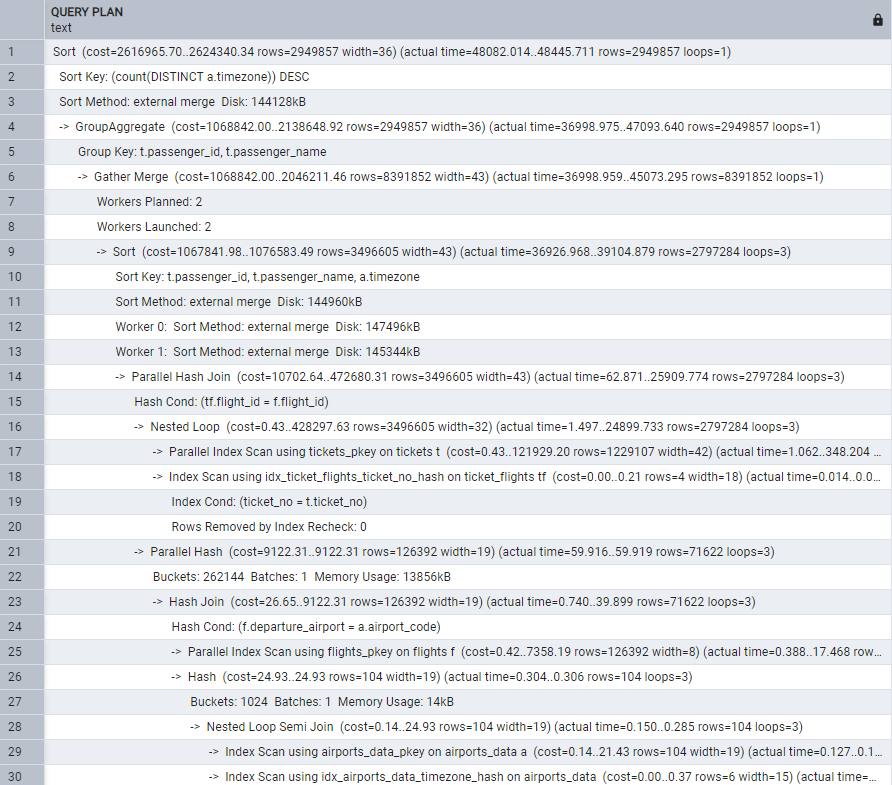
JOIN airports\_data a ON f.departure\_airport = a.airport\_code

WHERE a.timezone = ANY(SELECT timezone FROM airports\_data)

GROUP BY t.passenger\_id, t.passenger\_name

ORDER BY unique\_timezones DESC;

План выполнения запроса представлен на рисунке 9.8.



* + - * 1. План выполнения запроса с использованием хэш-индексов

## Вывод

В ходе выполнения работы были изучены методы оптимизации SQL запросов. В работе использовали СУБД PostgreSQL. В рамках работы сначала выполняли неоптимизированный запрос, затем несколько запросов с разными методами оптимизации. В каждом запросе использовали запрос EXPLAIN ANALYZE, для того чтобы проанализировать выполнение запроса.