# 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

Рассмотрим неоптимизированный запрос для отображения пассажиров и количества дней, в которые они совершали полёты. Запрос представлен в листинге 9.1.

Неоптимизированный запрос

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT EXTRACT(DOW FROM f.scheduled\_departure)) AS unique\_days\_of\_week

FROM tickets t

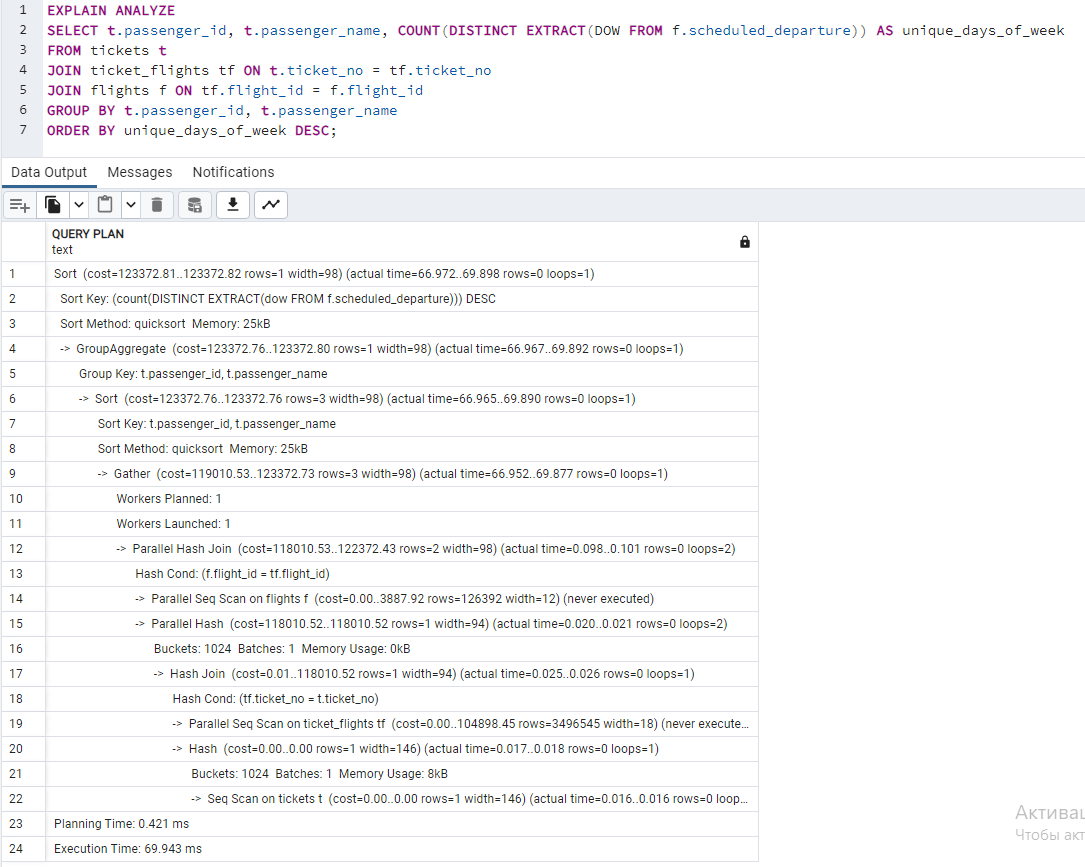
JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

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

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

ORDER BY unique\_days\_of\_week 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 ~70мс. Попробуем оптимизировать запрос, чтобы сократить время выполнения.

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

Оптимизированный запрос

EXPLAIN ANALYZE

WITH passenger\_days AS (SELECT t.passenger\_id, t.passenger\_name,EXTRACT(DOW FROM f.scheduled\_departure) AS day\_of\_week

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, day\_of\_week)

SELECT passenger\_id, passenger\_name, COUNT(day\_of\_week) AS unique\_days\_of\_week

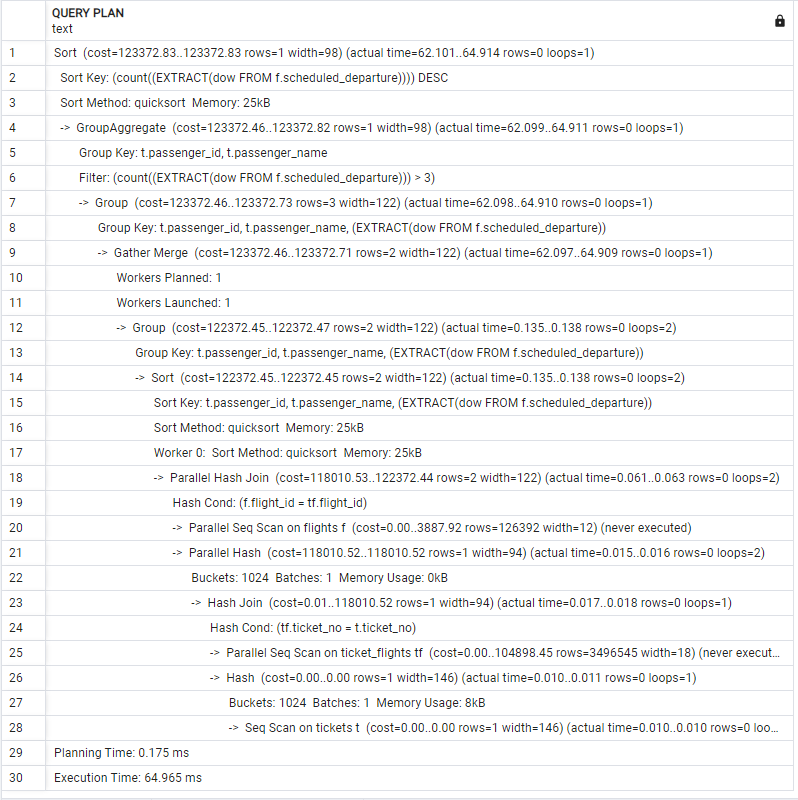
FROM passenger\_days

GROUP BY passenger\_id, passenger\_name

HAVING COUNT(day\_of\_week) > 3 -- Фильтруем только пассажиров с большим количеством дней недели

ORDER BY unique\_days\_of\_week 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 ~65мс.

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

Создание индексов и запрос с ними

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\_scheduled\_departure ON flights USING btree(scheduled\_departure);

SET enable\_seqscan = OFF;

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT EXTRACT(DOW FROM f.scheduled\_departure)) AS unique\_days\_of\_week

FROM tickets t

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

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

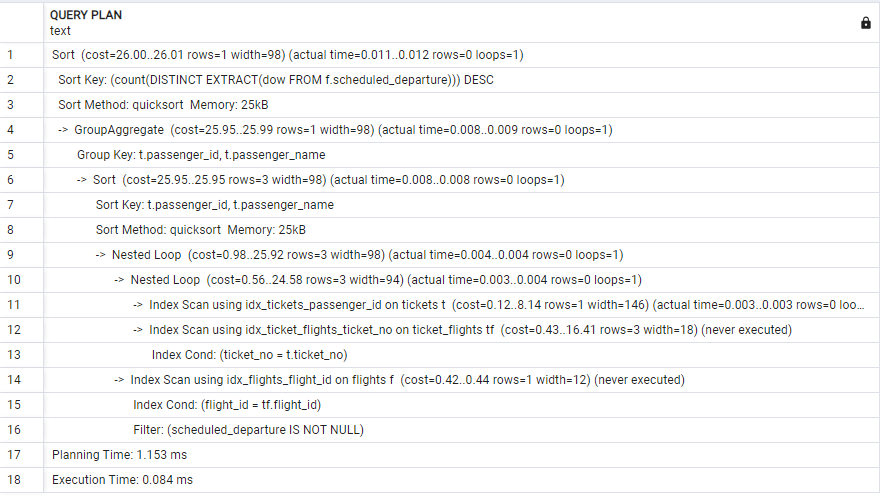
WHERE f.scheduled\_departure IS NOT NULL -- Используем B-tree индексGROUP BY t.passenger\_id, t.passenger\_name

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

ORDER BY unique\_days\_of\_week 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

Как видно на рисунке 9.3. Время выполнения запроса значительно сократилось. Увеличилось время планирования, однако повышение скорости выполнения запроса стало значительно выше.

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

Создание хэш индексов и запрос сними

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

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

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

CREATE INDEX idx\_flights\_scheduled\_departure\_hash ON flightsUSINGhash(scheduled\_departure);

SET enable\_seqscan = OFF;

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT EXTRACT(DOW FROM f.scheduled\_departure)) AS unique\_days\_of\_week

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 EXTRACT(DOW FROM f.scheduled\_departure) = ANY(ARRAY[0, 1,2,3,4,5, 6]) -- Используем хеш-индекс

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

ORDER BY unique\_days\_of\_week DESC;

План выполнения запроса представлен на рисунке 9.4.



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

Как видно на рисунке 9.4. время выполнения запроса также значительно сократилось. Время планирования также остаётся достаточно низким.

Рассмотрим другой запрос. В листинге 9.5. представлен неоптимизированный запрос, который анализирует в какие часы пассажиры совершали перелёты.

Неоптимизированный запрос

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT EXTRACT(HOUR FROM f.scheduled\_departure)) AS unique\_hours

FROM tickets t

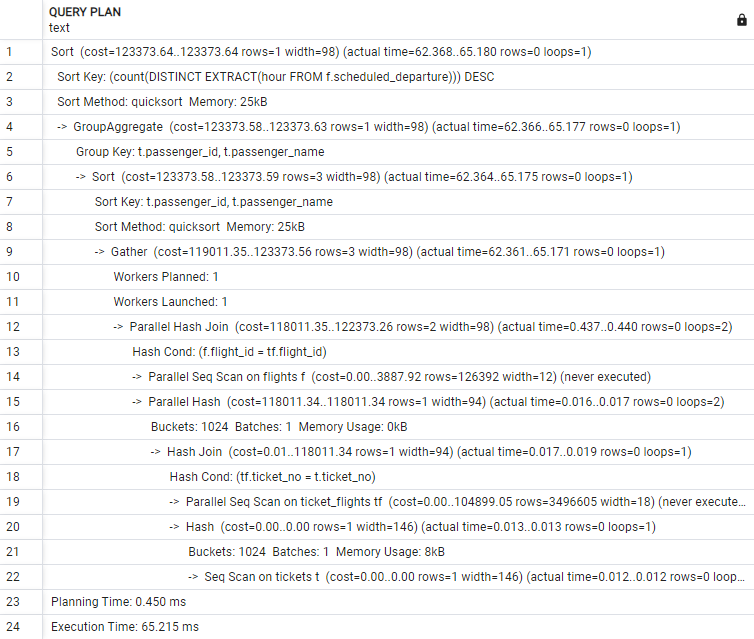
JOIN ticket\_flights tf ON t.ticket\_no = tf.ticket\_no

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

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

ORDER BY unique\_hours 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 ~65мс. Оптимизируем запрос и сравним время.

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

Оптимизирован

EXPLAIN ANALYZE

WITH passenger\_hours AS (SELECT t.passenger\_id, t.passenger\_name,EXTRACT(HOUR FROM f.scheduled\_departure) AS hour\_of\_day

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, hour\_of\_day)

SELECT passenger\_id, passenger\_name, COUNT(hour\_of\_day) AS unique\_hoursFROM passenger\_hours

GROUP BY passenger\_id, passenger\_name

HAVING COUNT(hour\_of\_day) > 5 -- Фильтруем только пассажировсбольшимколичеством часов

ORDER BY unique\_hours 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 ~63мс, 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 EXTRACT(HOUR FROM f.scheduled\_departure)) AS unique\_hours

FROM tickets t

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

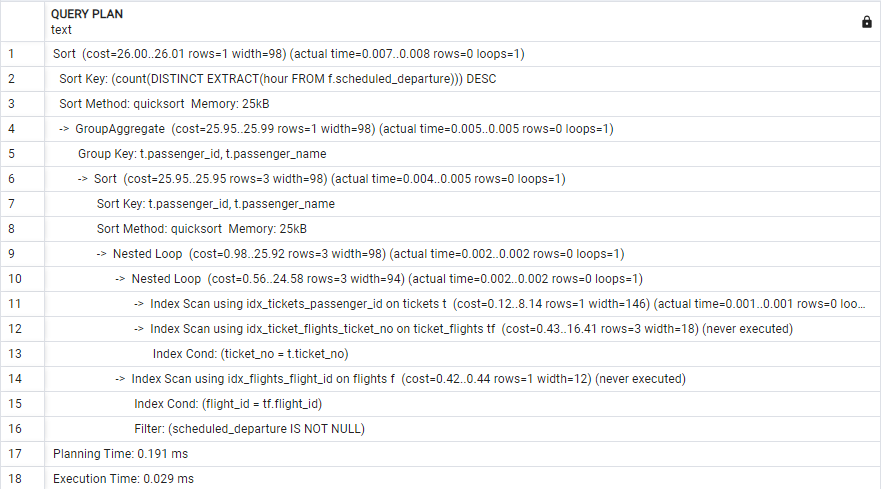
JOIN flights f ON tf.flight\_id = f.flight\_id

WHERE f.scheduled\_departure IS NOT NULL -- Используем B-tree индексGROUP BY t.passenger\_id, t.passenger\_name

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

ORDER BY unique\_hours DESC;

План выполнения запроса представлен на рисунке 9.7.



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

Как видно на рисунке 9.7. время выполнения значительно сократилось в сравнении с запросом без использования 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

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 USINGhash(flight\_id);

CREATE INDEX idx\_flights\_scheduled\_departure\_hash ON flightsUSINGhash(scheduled\_departure);

SET enable\_seqscan = OFF;

EXPLAIN ANALYZE

SELECT t.passenger\_id, t.passenger\_name, COUNT(DISTINCT EXTRACT(HOURFROM f.scheduled\_departure)) AS unique\_hours

FROM tickets t

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

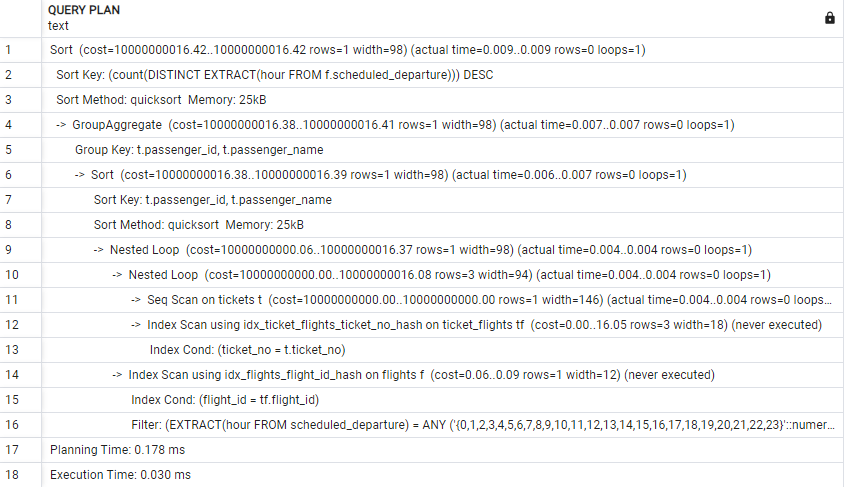
JOIN flights f ON tf.flight\_id = f.flight\_id

WHERE EXTRACT(HOUR FROM f.scheduled\_departure) = ANY(ARRAY[0,1,2,3,4,5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,23])--Используем хеш-индекс

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

ORDER BY unique\_hours DESC;

План выполнения запроса представлен на рисунке 9.8.



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

Как видно на рисунке 9.8. время выполнения запроса также невелико, как и с использованием B-tree индексов, время планирования немного сократилось.

## Вывод

В ходе выполнения лабораторной работы была исследована оптимизация производительности SQL-запросов на примере демонстрационной базы данных «авиаперелёты» от PostgresPro. Были рассмотрены разные подходы к оптимизации: переписывание запросов и создание индексов на основе B-деревьев и хэш-индексов.

На первом этапе были проанализированы исходные неоптимизированные запросы для определения пассажиров, летавших в наибольшее количество различных дней недели и в наибольшее количество различных часов. Время выполнения таких запросов составляло около 65–70 мс.

Далее была проведена оптимизация запросов, направленная на сокращение времени выполнения за счёт улучшения структуры запросов. Затем применялись индексы: сначала на основе B-деревьев, что позволило значительно ускорить выполнение запросов за счёт более эффективного поиска и соединения данных. Аналогичное ускорение было достигнуто при использовании хэш-индексов, которые проявляют эффективность при точечных операциях поиска.

Сравнительный анализ показал, что индексация данных оказывает наибольшее влияние на время выполнения запросов, значительно сокращая его по сравнению с неоптимизированными вариантами. При этом время планирования запросов немного увеличивалось, однако общий выигрыш в производительности полностью компенсировал это.

Таким образом, в результате лабораторной работы было подтверждено, что оптимизация запросов и использование индексов являются важнейшими средствами повышения эффективности работы с базами данных.