1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 5**

**Оптимизация запросов и управление производительностью**

по дисциплине «Системы управления базами данных»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ассистент Зубков Е.А.
3. Санкт-Петербург
4. 2023

# **Цель**

Получение навыков повышения производительности, оптимизации выполнения запросов и анализа планов запросов в СУБД.

# Задание

Работа выполняется над базой данных, разработанной при выполнении работы 1. Для оптимизации используйте запросы, разработанные для работы №3.

1. Заполните тестовую базу данных не менее чем 10 000 записями в каждом отношении, (возможно применение автоматической генерации). Обязательно учтите:
   1. Число тестовых записей может быть увеличено или уменьшено в зависимости от характеристик аппаратного обеспечения (*ориентируйтесь на время выполнения одного запроса более 2-3 секунд*).
   2. *Селективность* атрибутов, свойственная предметной области, должна быть в общем виде соблюдена в отношении заполняемых данных.
2. Ознакомьтесь с инструментарием, который ваша СУБД предоставляет для отслеживания состояния транзакций и параметров выполнения запросов. Ознакомьтесь с инструментарием просмотра планов выполнения запросов, определите какие типы отображения планов доступны.
3. Для каждого запроса, созданного к данным ранее согласно варианту:
   1. Составьте логический план выполнения запроса в соответствии с правилами логической оптимизации (только запросы 1,2; запрос 3 – по желанию).
   2. Определите и приведите физический план запроса при помощи инструментов СУБД:
      1. в графической форме (если это возможно);
      2. в текстовой или XML форме (обязательно).
   3. Если возможно построение предварительного и реального планов выполнения, приведите оба плана*, в случае если они различаются.*
   4. Зафиксируйте время выполнения запроса.
   5. Соотнесите физический план запроса с реляционным представлением запроса и логическим планом. Сделайте выводы.
4. Определите «узкие места» при выполнении запроса, воспользовавшись инструментами СУБД. При необходимости увеличьте число тестовых записей.
5. Создайте *необходимые* индексы исходя из запросов *и характеристик данных* во всех случаях, когда это может вызвать прирост производительности. Иначе аргументируйте неэффективность использования индекса для повышения производительности.
6. Для каждого запроса:
   1. Получите новый план запроса, представив его в графической или текстовой (XML) форме.
   2. Зафиксируйте все изменения в плане выполнения запроса и новое время выполнения каждого запроса.
   3. Укажите, используются ли предложенные вами индексы при выполнении плана. Если индекс не использован, определите причину его неэффективности.
7. Если это возможно, преобразуйте запросы для большей эффективности выполнения, в том числе, преобразовав логические операции. Получите новый план и сравните его показатели с предыдущими.
8. Сделайте выводы.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 ― Вариант задания

# Ход работы

## Создание базы

В соответствии с выданным вариантом (рисунок 1) была реализована логическая схема данных (рисунок 2) и создана база данных, состоящая из двух таблиц (рисунки 3, 4).

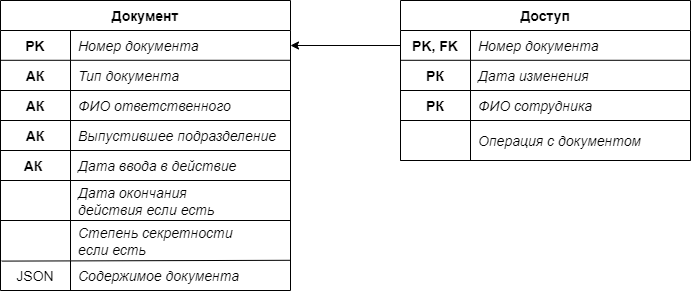


Рисунок 2 ― Реализованные связи

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 ― Схема для отношения 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 ― Схема для отношения 2

Требуется заполнить тестовую базу данных не менее чем 10 000 записями в каждом отношении. Данные были получены путем автоматической генерации с помощью generate\_series().

INSERT INTO document (type\_doc, fio\_respons, department, date\_start, date\_end, secret)

SELECT

md5(random()::text),

md5(random()::text),

md5(random()::text),

now() - (random() \* 365 \* 10 || ' days')::interval,

now() + (random() \* 365 \* 10 || ' days')::interval,

(random() \* 10)::integer

FROM generate\_series(1, 10000);

INSERT INTO change\_log (date\_change, fio, operation)

SELECT

now() - (random() \* 365 \* 5 || ' days')::interval,

md5(random()::text),

md5(random()::text)

FROM generate\_series(1, 10000);

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 ― Заполненные таблицы

## Отслеживание транзакций

Затем были изучен инструментарий, который СУБД предоставляет для отслеживания состояния транзакций и параметров и плана выполнения запросов. В представлении pg\_stat\_activity для каждого серверного процесса будет присутствовать по одной строке с информацией, относящейся к текущей деятельности этого процесса. pg\_stat\_activity содержит информацию о текущих активных соединениях к базе данных, выполняемых запросах, их длительности, статусе и других параметрах. На рисунке 6 представлен пример использования данного средства.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6 ― Результат запроса

EXPLAIN — инструмент, который выводит план выполнения запроса, позволяя оценить, каким образом PostgreSQL выполняет запрос, и предоставляет информацию о том, какие индексы и операции используются.

Типы отображения планов выполнения запросов включают:

Text — текстовый формат плана выполнения запроса, который выводится с помощью команды EXPLAIN.

Graphical — графический формат плана выполнения запроса, который выводится в pgAdmin и других графических интерфейсах для управления PostgreSQL.

JSON — формат плана выполнения запроса, представленный в формате JSON, который может быть использован для дальнейшей обработки программами и скриптами.

На рисунке 7 представлен результат выполнения запроса.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 7 — План запроса

Поскольку запрос был по первичному ключу, то поиск проходит по индексу, о чем свидетельствует строка *Index Scan using document\_pkey on document.*

*cost=0.29..8.30 rows=1 width=119*: оценка стоимости выполнения запроса. В данном случае стоимость от 0,29 до 8,30 и оценка количества строк равна 1, а ширина данных 119 байта.

*actual time=0.412..0.416 rows=1 loops=1*: фактическое время выполнения запроса. В данном случае время выполнения составило от 0,412 до 0,416 миллисекунд, найдена 1 строка, за 1 цикл выполнения.

## Анализ запросов

### Запрос 1

Требуется выбрать ФИО ответственных, которые отвечают более чем за один секретный документ заданной степени.

CREATE OR REPLACE FUNCTION first\_query(level int)

RETURNS TABLE (fio\_respons text, docs int)

AS $$

SELECT fio\_respons, COUNT(\*) as docs FROM document

WHERE fio\_respons IN

(SELECT DISTINCT fio\_respons FROM document WHERE secret = 6)

GROUP BY fio\_respons

HAVING COUNT(\*) > 1;

$$ LANGUAGE SQL;

SELECT \* FROM first\_query('1');

На рисунке 8 представлен результат запроса. Он не выводит ничего, поскольку в реализованной бд каждый пользователь может владеть только одним документом из-за ограничений уникальности.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 ― Результат запроса

Логический план выглядит следующим образом:

1. Вызов функции с указанием параметра.
2. Данная функция сначала выбирает из таблицы все записи с заданным уровнем секретности, удаляя повторы с помощью слова DISTINCT.
3. С помощью count(\*) подсчитывается общее количество строк, а затем выводятся только те строки, где ФИО повторяется более 1 раза.

Для составления физического плана воспользуемся инструментом EXPLAIN с параметрами FORMAT TEXT, ANALYZE, TIMING, VERBOSE, COSTS (рисунок 8).

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 8 — Результат запроса 1

Поиск происходит по хэшу, поэтому сначала создается таблица. Группировка значений для агрегации и устранение дубликатов могут выполняться алгоритмами, схожими с алгоритмами соединения. Один из способов состоит в том, чтобы построить хеш-таблицу по нужным полям. Каждое значение помещается в хеш-таблицу, только если его там еще нет. Таким образом в итоге в хеш-таблице остаются только уникальные значения.

HashAggregate (cost=656.02..668.76 rows=340 width=41) (actual time=7.003..7.006 rows=0 loops=1)

Output: document.fio\_respons, count(\*)

Group Key: document.fio\_respons

Filter: (count(\*) > 1)

Batches: 1 Memory Usage: 193kB

Rows Removed by Filter: 1019

Идея соединения с помощью хеширования состоит в поиске подходящих строк с помощью заранее подготовленной хеш-таблицы. На первом этапе узел Hash Join обращается к узлу Hash. Тот получает от своего дочернего узла весь внутренний набор строк и помещает его в хеш-таблицу. Хеш-таблица позволяет сохранять пары, составленные из ключа хеширования и значения, а затем искать значения по ключу за фиксированное время, не зависящее от размера хеш-таблицы. Для этого с помощью хеш-функции ключи хеширования распределяются случайно, но равномерно по ограниченному количеству корзин (bucket) хеш-таблицы. Число корзин всегда является степенью 2, поэтому номер корзины можно получить, взяв нужное количество двоичных разрядов значения хеш-функции.

-> Hash Join (cost=342.67..650.93 rows=1019 width=33) (actual time=2.860..6.236 rows=1019 loops=1)

Output: document.fio\_respons

Inner Unique: true

Hash Cond: ((document.fio\_respons)::text = (document\_1.fio\_respons)::text)

Далее уже идет обычный последовательный поиск.

-> Seq Scan on public.document (cost=0.00..282.00 rows=10000 width=33) (actual time=0.020..1.041 rows=10000 loops=1)

Output: document.num\_doc, document.type\_doc, document.fio\_respons, document.department, document.date\_start, document.date\_end, document.secret

На втором этапе (хеш-таблица к этому моменту уже готова) узел Hash Join обращается ко второму дочернему узлу за внешним набором строк. Для каждой прочитанной строки проверяется наличие соответствующих ей строк в хеш-таблице. Для этого хеш-функция вычисляется от значений полей внешнего набора, входящих в условие соединения.

-> Hash (cost=329.93..329.93 rows=1019 width=33) (actual time=2.788..2.789 rows=1019 loops=1)

Output: document\_1.fio\_respons

Buckets: 1024 Batches: 1 Memory Usage: 73kB

-> HashAggregate (cost=309.55..319.74 rows=1019 width=33) (actual time=2.276..2.465 rows=1019 loops=1)

Output: document\_1.fio\_respons

Group Key: document\_1.fio\_respons

Batches: 1 Memory Usage: 193kB

Также происходит фильтрация по указанному уровню секретности.

-> Seq Scan on public.document document\_1 (cost=0.00..307.00 rows=1019 width=33) (actual time=0.019..1.863 rows=1019 loops=1)

Output: document\_1.num\_doc, document\_1.type\_doc, document\_1.fio\_respons, document\_1.department, document\_1.date\_start, document\_1.date\_end, document\_1.secret

Filter: (document\_1.secret = 6)

Rows Removed by Filter: 8981

Таким образом, время выполнения запроса составило 7 миллисекунд.

Planning Time: 1.462 ms

Execution Time: 7.269 ms

Реляционная схема запроса представлена на рисунке 10.

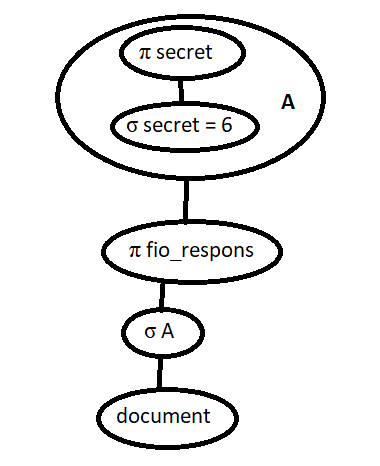


Рисунок 10 — Реляционная схема запроса 1

Попробуем оптимизировать запрос путем добавления индекса на параметр fio\_respons, поскольку поиск идет именно по нему, и повторим запрос (рисунок 11).

create index fio\_i on document(fio\_respons);

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 11 — Результат запроса после добавления индекса

Было проведено ещё несколько экспериментов и время запроса в среднем ухудшилось. Скорей всего это происходит потому, что долго строится таблица хэш-значений. При этом делать индекс на уровень секретности нет смысла из-за высокой селективности.

### Запрос 2

Требуется выбрать номера документов и ФИО ответственных для документов, срок окончания действия которых не в течение перечня заданных месяцев (в общем случае нескольких, не обязательно последовательных, в пределах года). На рисунке 12 представлен результат.

CREATE OR REPLACE FUNCTION second\_query(VARIADIC months integer[])

RETURNS TABLE (fio\_respons text, date\_end date, num\_doc bigint)

AS $$

SELECT fio\_respons, date\_end, num\_doc FROM document

WHERE (EXTRACT(MONTH FROM date\_end) NOT IN (SELECT unnest($1)))

ORDER BY fio\_respons;

$$ LANGUAGE SQL;

SELECT \* FROM second\_query(1, 4, 7, 12);

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 12 — Результат запроса 2

Логический план:

1. Вызов функции с указанием параметра.
2. С помощью слова EXTRACT извлекаем месяц из даты окончания.
3. Затем последовательно выбираем в таблице все записи, в которых месяц НЕ совпадает с переданными аргументами.

На рисунке 13 представлен физический план запроса. На рисунке 14 представлена реляционная схема запроса. πσ

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 13 — Проверка доступа для ответственного пользователя

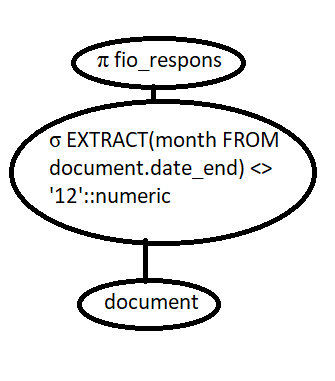


Рисунок 14 — Реляционная схема запроса 2

Сначала идет последовательный поиск по заданному фильтру (извлеченный месяц не в заданном диапазоне). Данный запрос уже является достаточно быстрым. Добавить индекс в данной ситуации возможно опять же только на ФИО, поскольку месяцы обладают высокой селективностью (рисунок 15).

Изображение выглядит как снимок экрана, текст, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 15 — Проверка доступа для неответственного пользователя

Время немного изменилось, но не значительно. Попробуем увеличить таблицу до 100000 данных, возможно разница станет заметнее (рисунок 16).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 16 — Логирование действий пользователя

Как видно из рисунка 16, есть разница в 4 миллисекунды. Скорей всего, в базах данных объемом более миллиона записей разница будет ещё более явной.

# Вывод

В ходе выполнения данной лабораторной работы были изучены методы оптимизации запросов, построения логических и физических планов выполнения запросов. Рассмотрены возможности PostgreSQL по созданию физических планов и получен опыт их анализа. Была произведена аналитика физических планов и производительности запросов, на основе которых можно сказать, что оптимизация запросов и введение индексов не всегда положительно влияет на время выполнения запросов.

ПРИЛОЖЕНИЕ 1

CREATE DATABASE lab5;

\c lab5

CREATE TABLE document (

num\_doc BIGSERIAL NOT NULL PRIMARY KEY,

type\_doc VARCHAR(50) NOT NULL,

fio\_respons VARCHAR(100) NOT NULL,

department VARCHAR(50) NOT NULL,

date\_start DATE NOT NULL,

date\_end DATE,

secret INT);

CREATE TABLE change\_log (

num\_doc BIGSERIAL NOT NULL UNIQUE REFERENCES document (num\_doc),

date\_change DATE NOT NULL,

fio VARCHAR(100) NOT NULL,

operation text,

PRIMARY KEY (num\_doc, date\_change, fio));

///////////////////////////////////////////////////////

INSERT INTO document (type\_doc, fio\_respons, department, date\_start, date\_end, secret)

SELECT

md5(random()::text),

md5(random()::text),

md5(random()::text),

now() - (random() \* 365 \* 10 || ' days')::interval,

now() + (random() \* 365 \* 10 || ' days')::interval,

(random() \* 10)::integer

FROM generate\_series(1, 100000);

INSERT INTO change\_log (date\_change, fio, operation)

SELECT

now() - (random() \* 365 \* 5 || ' days')::interval,

md5(random()::text),

md5(random()::text)

FROM generate\_series(1, 100000);

//////////////////////////////////////////////////

select pid, usename, datname, state, query

from pg\_stat\_activity

where datname = ‘lab5’;

explain analyze select \* from document where num\_doc = 5236;

EXPLAIN (ANALYZE TRUE, FORMAT TEXT, TIMING TRUE, VERBOSE TRUE, COSTS TRUE)

SELECT fio\_respons, COUNT(\*) as docs FROM document

WHERE fio\_respons IN

(SELECT DISTINCT fio\_respons FROM document WHERE secret = 6)

GROUP BY fio\_respons

HAVING COUNT(\*) > 1;

EXPLAIN (ANALYZE TRUE, FORMAT TEXT, TIMING TRUE, VERBOSE TRUE, COSTS TRUE)

SELECT fio\_respons, date\_end, num\_doc FROM document

WHERE (EXTRACT(MONTH FROM date\_end) NOT IN (12));