



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 4

Название: Оптимизация процессов в PostgreSQL


Дисциплина: Базы данных

Вариант: 23

Студент

ИУ6-33Б

(Группа)


(Подпись, дата)

Д. И. МIRONENKO

(И.О. Фамилия)

Преподаватель


(Подпись, дата)

М. А. Скворцова

(И.О. Фамилия)

Москва, 2023

Часть 1. Функции и язык PL/pgSQL

Задание

- Составить SQL-скрипты для создания нескольких функций, упрощающих манипуляции с данными.
- Продемонстрировать полученные знания о возможностях языка PL/pgSQL.

В скриптах должны использоваться:

- Циклы;
 - Ветвления;
 - Переменные;
 - Курсоры;
 - Исключения.
- Обосновать преимущества механизма функций перед механизмом представлений.

Создание функции для получения всех покупок указанного пользователя

```
CREATE OR REPLACE FUNCTION total_purchases(target_user uuid)
RETURNS integer AS
$$
DECLARE
    purchases_cursor CURSOR(target_user uuid)
        FOR SELECT software_id, price
        FROM store.purchase_history
        WHERE user_id = target_user;
    total_price integer DEFAULT 0;
    purchase_record record;
BEGIN
    OPEN purchases_cursor(target_user);

    LOOP
        FETCH purchases_cursor INTO purchase_record;
        EXIT WHEN NOT FOUND;

        total_price := total_price + purchase_record.price;
    END LOOP;
```


```

CLOSE purchases_cursor;

IF total_price = 0 THEN
    RAISE EXCEPTION 'No purchases found for the user %!', target_user;
END IF;

RETURN total_price;
END;
$$
LANGUAGE 'plpgsql';

```



The screenshot shows a terminal window titled "psql -h localhost -d ~/D/db-labs". The prompt is "big-labs=#". The user has entered the query "SELECT * FROM total_purchases('63ca0113-57c3-48d5-8330-c2a24262328e');". The result is displayed as a table with one row containing the value "1330". Below the table, it says "(1 row)". The prompt "big-labs=#" is shown again with a cursor.

```

psql -h localhost -d ~/D/db-labs
big-labs=# SELECT * FROM total_purchases('63ca0113-57c3-48d5-8330-c2a24262328e');
total_purchases
-----
1330
(1 row)
big-labs=#

```

Рисунок 1 – тестирование функции для получения всех покупок указанного пользователя

Создание функции, проверяющей доступна ли миграция для указанного пользователя

```

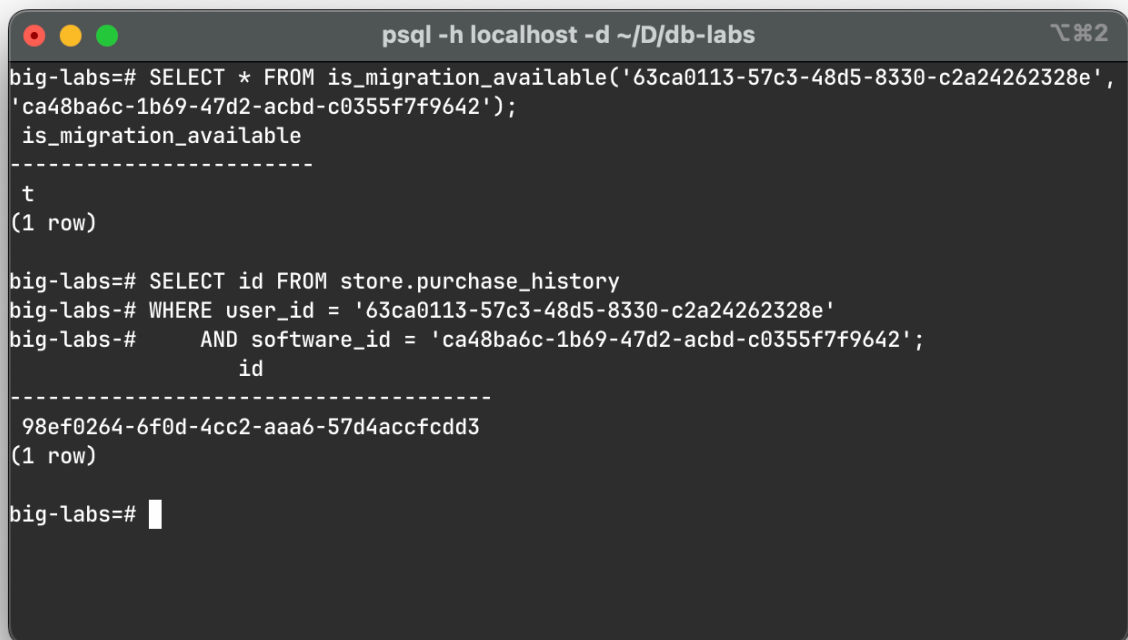
CREATE OR REPLACE FUNCTION is_migration_available(target_user uuid,
target_software uuid)
RETURNS boolean AS

```

```
$$
DECLARE
    purchases CURSOR(target_user uuid, target_software uuid)
        FOR SELECT id
            FROM store.purchase_history
            WHERE user_id = target_user
                AND software_id = target_software;
    purchase_record record;
BEGIN
    OPEN purchases(target_user, target_software);

    FETCH purchases INTO purchase_record;
    IF NOT FOUND THEN
        CLOSE purchases;
        RETURN false;
    END IF;

    CLOSE purchases;
    RETURN true;
END;
$$
LANGUAGE 'plpgsql';
```



```
psql -h localhost -d ~/D/db-labs
big-labs=# SELECT * FROM is_migration_available('63ca0113-57c3-48d5-8330-c2a24262328e',
'ca48ba6c-1b69-47d2-acbd-c0355f7f9642');
 is_migration_available
-----
 t
(1 row)

big-labs=# SELECT id FROM store.purchase_history
big-labs=# WHERE user_id = '63ca0113-57c3-48d5-8330-c2a24262328e'
big-labs=#      AND software_id = 'ca48ba6c-1b69-47d2-acbd-c0355f7f9642';
      id
-----
 98ef0264-6f0d-4cc2-aaa6-57d4accfcdd3
(1 row)

big-labs=#
```

Рисунок 2 – тестирование функции, проверяющей доступна ли миграция для указанного пользователя

В чем преимущества механизма функций перед механизмом представлений?

Функции могут принимать параметры и возвращать результаты, что делает их более гибкими для обработки различных вариантов запросов к данным. Они также могут быть повторно использованы в различных запросах и процедурах, что улучшает эффективность разработки и сокращает дублирование кода. Кроме того, в отличие от представлений, функции могут оптимизироваться для выполнения сложных операций, обеспечивая более эффективную обработку данных.

Часть 2. Оптимизация запросов. Индексы

Задание

Для выполнения задания необходим достаточно большой объем данных, чтобы оптимизация была целесообразной (порядка 1 млн. строк в каждой таблице).

Необходимо подготовить два запроса:

- Запрос к одной таблице, содержащий фильтрацию по нескольким полям.
- Запрос к нескольким связанным таблицам, содержащий фильтрацию по нескольким полям.

Для каждого из этих запросов необходимо провести следующие шаги:

- Получить план выполнения запроса без использования индексов.
- Получить статистику (IO и Time) выполнения запроса без использования индексов.
- Создать нужные индексы, позволяющие ускорить запрос.
- Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.
- Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.
- Оценить эффективность выполнения оптимизированного запроса. Также необходимо продемонстрировать полезность индексов для организации полнотекстового поиска, фильтрации с использованием массива и json-формата.

Для таблицы объемом больше 100 млн. записей произвести оптимизацию, позволяющую быстро удалять старые данные, ускорить вставку и чтение данных.

Запрос на получение наиболее важной информации о ПО

```
SELECT
    name,
    description,
    tags,
    license_price
FROM store.softwares
WHERE
    category = 'education'
    AND 'creation' = ANY(tags);
```

```
psql -h localhost -d ~/db-labs
big-labs=# EXPLAIN ANALYZE
SELECT
  name,
  description,
  tags,
  license_price
FROM store.softwares
WHERE
  category = 'education'
  AND 'creation' = ANY(tags);

                                QUERY PLAN
-----
Gather  (cost=1000.00..22675.74 rows=7 width=92) (actual time=28.072..344.208 rows=6 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Seq Scan on softwares  (cost=0.00..21675.04 rows=3 width=92) (actual time=182.258..278.108 rows=2 loops=3)
        Filter: ((category = 'education'::software_categories) AND ('creation'::text = ANY ((tags)::text[])))
        Rows Removed by Filter: 166665
Planning Time: 7.257 ms
Execution Time: 345.008 ms
(8 rows)

big-labs=#
```

Рисунок 3 – анализ запроса на получение наиболее важной информации о ПО

Добавление нужных индексов для оптимизации запроса выше

```
CREATE INDEX idx_category ON "store"."softwares" ("category");
CREATE INDEX idx_tags ON "store"."softwares" USING GIN("tags");
```

```
psql -h localhost -d ~/db-labs
big-labs=# EXPLAIN ANALYZE
big-labs=# SELECT
big-labs=#     name,
big-labs=#     description,
big-labs=#     tags,
big-labs=#     license_price
big-labs=# FROM store.softwares
big-labs=# WHERE
big-labs=#     category = 'education'
big-labs=#     AND 'creation' = ANY(tags);

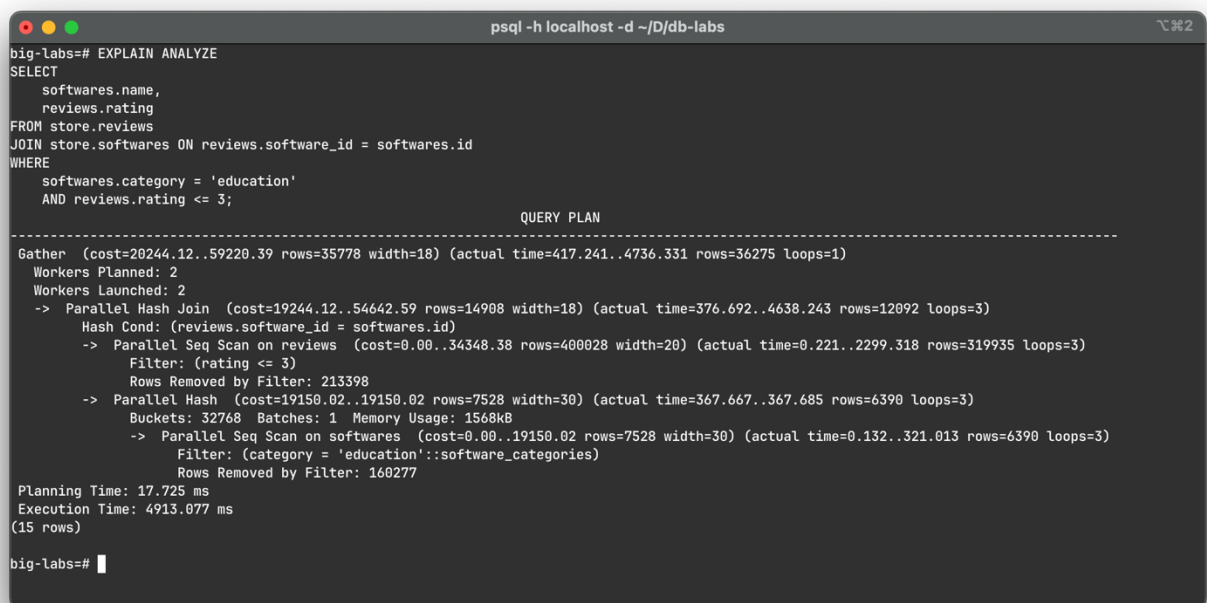
                                QUERY PLAN
-----
Bitmap Heap Scan on softwares  (cost=211.50..16350.17 rows=2 width=92) (actual time=21.774..306.159 rows=5 loops=1)
  Recheck Cond: (category = 'education'::software_categories)
  Filter: ('creation'::text = ANY ((tags)::text[]))
  Rows Removed by Filter: 11470
  Heap Blocks: exact=6705
  -> Bitmap Index Scan on idx_category  (cost=0.00..211.50 rows=11610 width=0) (actual time=8.752..8.758 rows=11475 loops=1)
        Index Cond: (category = 'education'::software_categories)
Planning Time: 1.803 ms
Execution Time: 307.296 ms
(9 rows)

big-labs=#
```

Рисунок 4 – анализ запроса на получение наиболее важной информации о ПО после оптимизации

Запрос на получение отрицательных отзывов о ПО из категории “education”

```
SELECT
    softwares.name,
    reviews.rating,
    reviews.review_content
FROM store.reviews
JOIN store.softwares ON reviews.software_id = softwares.id
WHERE
    softwares.category = 'education'
    AND reviews.rating <= 3;
```



```
big-labs=# EXPLAIN ANALYZE
SELECT
    softwares.name,
    reviews.rating
FROM store.reviews
JOIN store.softwares ON reviews.software_id = softwares.id
WHERE
    softwares.category = 'education'
    AND reviews.rating <= 3;

                                QUERY PLAN
-----
Gather  (cost=20244.12..59220.39 rows=35778 width=18) (actual time=417.241..4736.331 rows=36275 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Hash Join (cost=19244.12..54642.59 rows=14908 width=18) (actual time=376.692..4638.243 rows=12092 loops=3)
    Hash Cond: (reviews.software_id = softwares.id)
    -> Parallel Seq Scan on reviews  (cost=0.00..34348.38 rows=400028 width=20) (actual time=0.221..2299.318 rows=319935 loops=3)
      Filter: (rating <= 3)
      Rows Removed by Filter: 213398
    -> Parallel Hash  (cost=19150.02..19150.02 rows=7528 width=30) (actual time=367.667..367.685 rows=6390 loops=3)
      Buckets: 32768  Batches: 1  Memory Usage: 1568kB
      -> Parallel Seq Scan on softwares  (cost=0.00..19150.02 rows=7528 width=30) (actual time=0.132..321.013 rows=6390 loops=3)
        Filter: (category = 'education'::software_categories)
        Rows Removed by Filter: 160277
Planning Time: 17.725 ms
Execution Time: 4913.077 ms
(15 rows)

big-labs=#
```

Рисунок 5 – анализ запроса на получение отрицательных отзывов о ПО из категории “education”

Добавление нужных индексов для оптимизации запроса выше

```
CREATE INDEX idx_category ON "store"."softwares" ("category");
CREATE INDEX idx_rating ON "store"."reviews" ("rating");
```



```
psql -h localhost -d ~/D/db-labs
big-labs=# EXPLAIN ANALYZE
SELECT
    softwares.name,
    reviews.rating
FROM store.reviews
JOIN store.softwares ON reviews.software_id = softwares.id
WHERE
    softwares.category = 'education'
    AND reviews.rating <= 3;

QUERY PLAN

-----
Gather  (cost=18807.13..57781.71 rows=35773 width=18) (actual time=197.384..4670.452 rows=36275 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Hash Join (cost=17807.13..53204.41 rows=14905 width=18) (actual time=157.483..4562.593 rows=12092 loops=3)
    Hash Cond: (reviews.software_id = softwares.id)
    -> Parallel Seq Scan on reviews (cost=0.00..34347.33 rows=399978 width=20) (actual time=0.294..2399.504 rows=319935 loops=3)
      Filter: (rating <= 3)
      Rows Removed by Filter: 213398
    -> Parallel Hash (cost=17710.08..17710.08 rows=7764 width=30) (actual time=151.868..151.881 rows=6390 loops=3)
      Buckets: 32768 Batches: 1 Memory Usage: 1536kB
      -> Parallel Bitmap Heap Scan on softwares (cost=208.83..17710.08 rows=7764 width=30) (actual time=9.549..108.224 rows=6390 loops=3)
        Recheck Cond: (category = 'education'::software_categories)
        Heap Blocks: exact=5744
        -> Bitmap Index Scan on idx_category (cost=0.00..204.17 rows=18633 width=0) (actual time=19.109..19.112 rows=19170 loops=1)
          Index Cond: (category = 'education'::software_categories)

Planning Time: 19.117 ms
Execution Time: 4839.805 ms
(17 rows)
big-labs=#
```

Рисунок 6 – анализ запроса на получение отрицательных отзывов о ПО из категории “education” после оптимизации

Вывод

Реализованы полезные функции для анализа базы данных; добавлены индексы для некоторых колонок с целью повысить эффективность часто используемых запросов.

Контрольные вопросы

1. *Описать в каких случаях целесообразно создавать функции.*

Функции могут использоваться для объединения серии операций в единое действие, что улучшает модульность, обслуживаемость и повторное использование кода. Функции могут содержать сложные запросы, которые могут быть повторно использованы в различных контекстах.

2. *Рассказать о курсорах, как и зачем используются.*

Курсоры - это механизм, который позволяет выполнять запросы к базе данных построчно, шаг за шагом. Они используются для обработки больших объемов данных и ускорения выполнения запросов.

3. *Рассказать о работе с циклами.*

Циклы - это механизм, который позволяет выполнять определенные

операции с каждым элементом данных в таблице или наборе данных. Они могут использоваться для решения различных задач.

```
DECLARE c CURSOR FOR SELECT * FROM <table>;
```

```
FETCH NEXT FROM c INTO <variable>;
```

```
WHILE <variable> IS NOT NULL DO
```

```
    -- выполнение операций с каждой строкой данных
```

```
    FETCH NEXT FROM c INTO <variable>;
```

```
END WHILE;
```

4. *В чем отличие первичного ключа и уникального индекса?*

Основное различие между ними заключается в том, что первичный ключ обычно используется для однозначной идентификации каждой строки в таблице, в то время как уникальный индекс просто гарантирует уникальность значений в столбцах, но не обязательно для идентификации строк. Кроме того, для первичного ключа по умолчанию создается кластеризованный индекс для ускорения производительности доступа к данным.

5. *В каких случаях имеет смысл создавать индексы? Какие колонки следует включать в индекс и почему?*

Чтобы ускорить частые операции поиска. Индексы также могут быть использованы при выполнении операций объединения (JOIN) для ускорения запросов. Кроме того, индексы могут улучшить производительность операций сортировки и группировки.

В индекс нужно включать колонки, которые используются в условиях WHERE или которые используются в операциях объединения таблиц.

6. *Какие существуют способы внутренней организации индексов?*

В-дерево, hash, gin (generalized search tree), GiST (generalized search tree).

7. *Рассказать о проблеме фрагментации индексов. Как бороться с фрагментацией?*

Фрагментация индексов возникает, когда данные в таблице часто добавляются, обновляются или удаляются, что приводит к тому, что индексы становятся разрозненными и неоптимальными для поиска. Это

может привести к увеличению времени выполнения запросов, уменьшению производительности базы данных и ухудшению общего опыта работы с системой.

Для борьбы с фрагментацией индексов можно применять следующие методы:

- VACUUM и REINDEX. Нужно регулярно выполнять операцию VACUUM, чтобы освобождать место, занимаемое удаленными строками таблицы, и уменьшать фрагментацию индексов. Также, нужно использовать команду REINDEX для восстановления порядка данных в индексах.

- CLUSTER. Команда CLUSTER позволяет упорядочить данные в таблице в соответствии с порядком индекса, что снижает фрагментацию и улучшает производительность запросов.

8. *Имеет ли значение порядок указания колонок при создании индекса?*

Да. Порядок колонок определяет, какой тип запросов может использоваться для ускорения. Например, составной индексы на столбцы (A, B) может быть эффективен при выполнении запросов, которые фильтруют данные по столбцу A или используют порядок (A, B). Однако он может быть менее эффективен для запросов, которые ищут данные только по столбцу B.

9. *В чем разница между Index Scan и Index Seek?*

Index Scan и Index Seek – это два способа доступа к данным через индексы, и их разница заключается в способе выполнения запроса.

Index Scan означает, что база данных просматривает все дерево индекса и находит нужные записи по условию запроса. Это может быть эффективно, если необходимо найти большой диапазон данных или если индекс отсортирован в нужном порядке.

Index Seek означает точный поиск определенной записи в индексе. База данных использует индекс, чтобы найти нужную запись без просмотра всех остальных записей. Это обычно более эффективный способ доступа к данным при поиске конкретных значений.

10. *В чем разница между секционированием и наследованием?*

Секционирование – это процесс разделения большой таблицы на несколько

меньших частей, называемых секциями. При этом каждая секция может иметь свою собственную структуру данных и правила доступа.

Наследование — это процесс создания новой таблицы на основе существующей таблицы. Новая таблица наследует структуру данных и правила доступа существующей таблицы. Наследование позволяет упростить проектирование базы данных и улучшить ее производительность, так как запросы к базе данных могут выполняться быстрее, если они выполняются на таблице с более низкой нагрузкой.

11. *Зачем нужен ANALYZE?*

ANALYZE - это команда, которая используется для анализа производительности базы данных. Она позволяет определить, какие запросы занимают больше всего времени выполнения и какие операции выполняются медленнее всего.

12. *Могут ли индексы ухудшить производительность?*

Да, индексы могут ухудшить производительность в базе данных. При обновлении данных на индексированных столбцах база данных может столкнуться с проблемой медленных операций обновления из-за необходимости обновления индексов. Со временем индексы могут стать фрагментированными, что снижает их эффективность и ведет к ухудшению производительности базы данных.

13. *На что влияет порядок сортировки (ASC\DESC) при создании индекса?*

Порядок сортировки при создании индекса влияет на скорость поиска и выборки данных из таблицы. Если указать DESC, то слева в дереве окажутся большие значения, а справа - меньшие.