МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

Основы DML-запросов в PostgreSQL.

Отчёт по лабораторной работе №2.2

по дисциплине

«Управление данными»

Разработал студент гр. ИВТб-2301-05-00 _____/Черкасов А. А./

(подпись)

Старший Преподователь _____/Клюкин В. Л./

(подпись)

Работа защищена «__»_____ 2025 г.

Киров

2025

Цели лабораторной работы

- научиться использованию агрегатных функций;
- научиться использованию операторов и встроенных функции, работе с датами.

Задание

1. Выполнить запросы с использованием агрегатных функций и предложения `HAVING`.
2. Продемонстрировать использование операторов, встроенных функций и работу с датами.
3. Применить подзапросы и предложение `WITH`.
4. Использовать предложение `RETURNING`.
5. Создать и выполнить SQL-скрипты для анализа планов выполнения запросов с помощью `EXPLAIN`.

Агрегатные функции и предложение HAVING

Агрегатные функции позволяют выполнять вычисления над множеством строк. В рамках работы были выполнены следующие запросы.

Пример 1: Подсчёт количества устройств и событий

Запросы подсчитывают общее количество записей в таблицах `devices` и `events`.

```
-- Количество устройств
select count(*) from devices;

-- Количество событий
select count(*) from events;
```

Пример 2: Статистика по идентификаторам устройств

Запрос выводит сумму, минимум, максимум и среднее значение по столбцу `id` в таблице `devices`. Результат аналогичен представлению `device_id_stats`, но в одной строке.

```
select
  sum(id) as sum_id,
  min(id) as min_id,
  max(id) as max_id,
  avg(id) as avg_id
from devices;
```

Пример 3: Группировка событий по типу

Запрос группирует события по их типу и подсчитывает количество событий каждого типа.

```
select event_type, count(*) as event_count
from events
group by event_type
order by event_count desc;
```

Пример 4: Фильтрация групп с помощью HAVING

Запрос находит типы устройств, у которых подключено более одного устройства. Предложение `HAVING` фильтрует результаты после группировки.

```
select dt.type_name, count(d.id) as device_count
from devices d
join device_types dt on d.type_id = dt.id
group by dt.type_name
having count(d.id) > 1;
```

Операторы, встроенные функции и работа с датами

В PostgreSQL доступен широкий набор операторов и функций для обработки данных.

Пример 5: Логические операторы и сравнение с NULL

Проверка логических выражений и корректное сравнение со значением `NULL`.

```
-- Логическое выражение
select not 1 < 2 and 3 < 4 or 5 = 5; -- true

-- Сравнение с NULL
select 1 is null or 5 is not null; -- true
```

Пример 6: Работа со строками

Использование функций для конкатенации, изменения регистра и проверки по шаблону.

```
-- Конкатенация строк
select 'User: ' || username from users;

-- Приведение к нижнему регистру
select lower(email) from users;

-- Поиск по шаблону (пользователи с почтой на example.com)
select username from users where email like '%@example.com';
```

Пример 7: Работа с датами

Примеры работы с типом данных `timestamp`, в частности со столбцом `created_at` в таблице `events`.

```
-- Текущая дата и время
select now();
```

```sql
-- События за последние 24 часа (гипотетически)
select * from events
where created_at > now() - interval '1 day';

-- Разница между датами (в днях)
select (now() - created_at) as age from events;
```

Подзапросы и предложение WITH

Подзапросы и общие табличные выражения (CTE) позволяют структурировать сложные запросы.

Пример 8: Простой подзапрос

Запрос находит имена пользователей, которые владеют хабами с устройствами типа "Lamp".

```sql
select username
from users
where id in (
  select distinct h.user_id
  from hubs h
  join devices d on h.id = d.hub_id
  join device_types dt on d.type_id = dt.id
  where dt.type_name = 'Lamp'
);
```

Пример 9: Использование WITH (CTE)

Тот же запрос, переписанный с использованием CTE для улучшения читаемости.

```sql
with lamp_hubs as (
  select distinct h.user_id
  from hubs h
  join devices d on h.id = d.hub_id
```

```
  join device_types dt on d.type_id = dt.id
  where dt.type_name = 'Lamp'
)
select username
from users
where id in (select user_id from lamp_hubs);
```

Предложение RETURNING

Предложение `RETURNING` позволяет получить данные об изменённых или вставленных строках в одном запросе.

Пример 10: Вставка с RETURNING

Добавление нового типа устройства и получение его идентификатора.

```
insert into device_types(type_name)
values ('Smart Socket')
returning id;
```

Пример 11: Обновление с RETURNING

Обновление статуса устройства и возврат его нового состояния.

```
update devices
set status = '{"power": "off", "brightness": 50}'
where name = 'Ceiling Lamp'
returning *;
```

Анализ плана выполнения запросов (EXPLAIN)

Для оптимизации производительности используется команда `EXPLAIN`.

Пример 12: Последовательное сканирование

План запроса на выборку всех устройств. Так как в условии нет фильтрации по индексу, используется последовательное сканирование (Sequential Scan).

```
explain (analyze, buffers) select * from devices;
```

Пример 13: Индексное сканирование

План запроса на выборку устройства по его уникальному идентификатору. Здесь используется индексное сканирование (Index Scan), так как столбец `id` является первичным ключом и имеет индекс.

```
explain (analyze, buffers) select * from devices where id = 1;
```

Пример 14: Соединение таблиц

План запроса для представления `device_full_info`, которое соединяет несколько таблиц. PostgreSQL может выбрать различные стратегии соединения (Nested Loop, Hash Join, Merge Join) в зависимости от статистики и настроек.

```
explain (analyze, format json)
select * from device_full_info;
```

Вывод

В ходе выполнения лабораторной работы №2_2 были освоены ключевые аспекты языка DML в PostgreSQL. Были изучены и применены на практике агрегатные функции (`count`, `sum`, `avg`, `min`, `max`) и предложение `HAVING` для фильтрации сгруппированных данных. Продемонстрировано использование различных операторов, встроенных функций для работы со строками и датами. Рассмотрены механизмы подзапросов и общих табличных выражений (`WITH`) для повышения читаемости сложных запросов. Также изучено предложение `RETURNING` для получения информации об изменённых строках. Наконец, с помощью команды `EXPLAIN` был проведен анализ планов выполнения запросов, что является важным инструментом для оптимизации производительности базы данных.

Приложение А1. Исходный код

```
# Общий Containerfile
FROM docker.io/library/postgres:alpine3.22

ENV POSTGRES_USER=pozordom_user
ENV POSTGRES_PASSWORD=pozordom_pass
ENV POSTGRES_DB=pozordom

# Лаб 1 - Создание таблиц
COPY lab1/code/init.sql /docker-entrypoint-initdb.d/
# Лаб 2.1 - Наполнение таблиц и представления
COPY lab2_1/code/init_data.sql /docker-entrypoint-initdb.d/
COPY lab2_1/code/views.sql /docker-entrypoint-initdb.d/
```

Приложение А2. SQL-скрипты для отчёта

Агрегатные функции и HAVING

```
-- Пример 1
select count(*) from devices;
select count(*) from events;

-- Пример 2
select
  sum(id) as sum_id,
  min(id) as min_id,
  max(id) as max_id,
  avg(id) as avg_id
from devices;

-- Пример 3
select event_type, count(*) as event_count
from events
group by event_type
order by event_count desc;
```

```sql
-- Пример 4
select dt.type_name, count(d.id) as device_count
from devices d
join device_types dt on d.type_id = dt.id
group by dt.type_name
having count(d.id) > 1;
```

Операторы и функции

```sql
-- Пример 5
select not 1 < 2 and 3 < 4 or 5 = 5;
select 1 is null or 5 is not null;

-- Пример 6
select 'User: ' || username from users;
select lower(email) from users;
select username from users where email like '%@example.com';

-- Пример 7
select now();
select * from events where created_at > now() - interval '1 day';
select (now() - created_at) as age from events;
```

Подзапросы и WITH

```sql
-- Пример 8
select username
from users
where id in (
  select distinct h.user_id
  from hubs h
  join devices d on h.id = d.hub_id
  join device_types dt on d.type_id = dt.id
  where dt.type_name = 'Lamp'
);

-- Пример 9
```

```sql
with lamp_hubs as (
  select distinct h.user_id
  from hubs h
  join devices d on h.id = d.hub_id
  join device_types dt on d.type_id = dt.id
  where dt.type_name = 'Lamp'
)
select username
from users
where id in (select user_id from lamp_hubs);
```

RETURNING и EXPLAIN

```sql
-- Пример 10
insert into device_types(type_name) values ('Smart Socket') returning id;

-- Пример 11
update devices set status = '{"power": "off", "brightness": 50}' where name = 'Ceiling Lamp

-- Пример 12
explain (analyze, buffers) select * from devices;

-- Пример 13
explain (analyze, buffers) select * from devices where id = 1;

-- Пример 14
explain (analyze, format json) select * from device_full_info;
```