

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 Преподаватель

_____/Клюкин В. Л./
(подпись)

Работа защищена

«__» _____ 2025 г.

Киров
2025

Цели лабораторной работы

- научиться использованию агрегатных функций;
- научиться использованию операторов и встроенных функций, 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;
```



1	4
---	---

1	3
---	---

Рисунок 1 - Вывод примера 1.

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;
```

	123 sum_id ▼	123 min_id ▼	123 max_id ▼	123 avg_id ▼
1	10	1	4	2.5

Рисунок 2 - Вывод примера 2.

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;
```

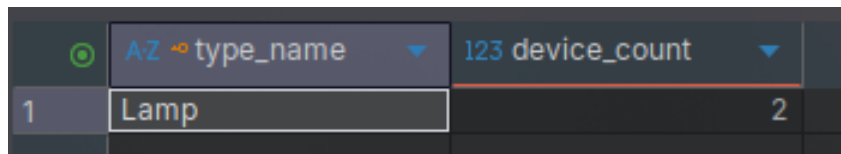
	AZ event_type ▼	123 event_count ▼
1	switch_on	1
2	motion_detected	1
3	temperature_change	1

Рисунок 3 - Вывод примера 3.

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;
```



	AZ ↗ type_name ▼	123 device_count ▼
1	Lamp	2

Рисунок 4 - Вывод примера 4.

Операторы, встроенные функции и работа с датами

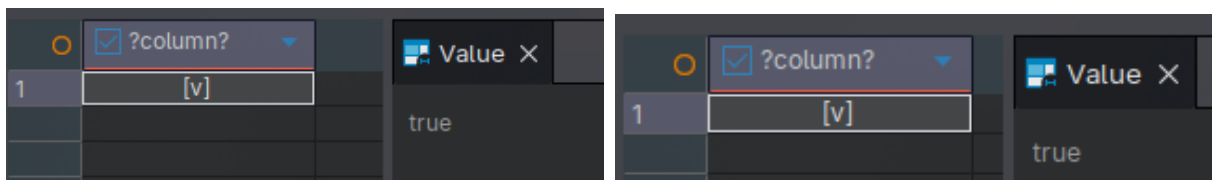
В PostgreSQL доступен широкий набор операторов и функций для обработки данных.

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
```



	?column? ▼	Value X
1	[v]	true

	?column? ▼	Value X
1	[v]	true

Рисунок 5 - Вывод примера 5.

Пример 6: Работа со строками

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

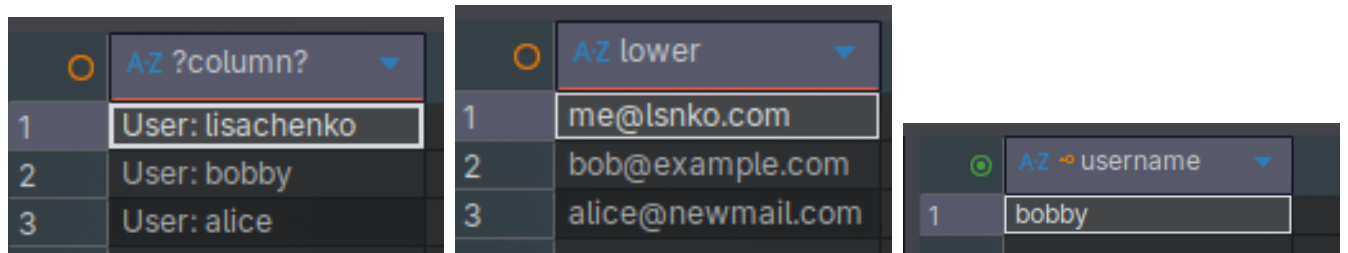
```
-- Конкатенация строк
select 'User: ' || username from users;
```

```
-- 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';
```



The image shows three screenshots of a database query interface. The first screenshot shows a query result for 'select lower(email) from users;' with three rows: 'User: lisachenko', 'User: bobby', and 'User: alice'. The second screenshot shows a query result for 'select username from users where email like '%example.com';' with three rows: 'me@lsnko.com', 'bob@example.com', and 'alice@newmail.com'. The third screenshot shows a query result for 'select username from users where email like '%example.com';' with one row: 'bobby'.

	AZ ?column?
1	User: lisachenko
2	User: bobby
3	User: alice

	AZ lower
1	me@lsnko.com
2	bob@example.com
3	alice@newmail.com

	AZ username
1	bobby

Рисунок 6 - Вывод примера 6.

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();
```

```
-- 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;
```

	now	
1	2025-10-01 14:59:23.835 +0300	

id	device_id	event_type	event_data	created_at
1	1	switch_on	{"power": "on"}	2025-10-01 11:34:02.381
2	2	temperature_change	{"temperature": 23.0}	2025-10-01 11:34:02.381
3	4	motion_detected	{"movement": true}	2025-10-01 11:34:02.381

	age	
1	03:28:17.696211	
2	03:28:17.696211	
3	03:28:17.696211	

Рисунок 7 - Вывод примера 7.

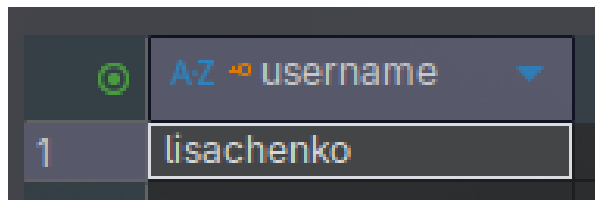
Подзапросы и предложение WITH

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

Пример 8: Простой подзапрос

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

```
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'
);
```



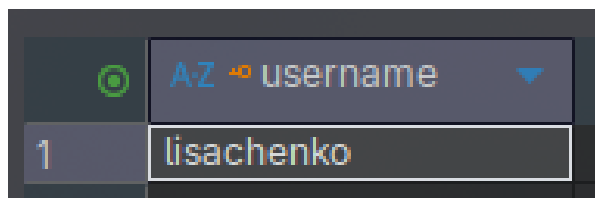
	A-Z username ▼
1	lisachenko

Рисунок 8 - Вывод примера 8.

Пример 9: Использование WITH (CTE)

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

```
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);
```



	A-Z username ▼
1	lisachenko

Рисунок 9 - Вывод примера 9.

RETURNING

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;
```

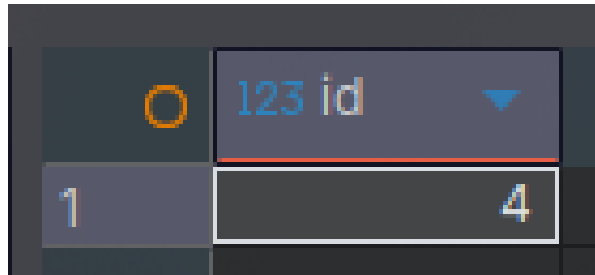


Рисунок 10 - Вывод примера 10.

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 *;
```



	 123 id ▼	123 hub_id ▼	123 type_id ▼	AZ name ▼	 status ▼
1	1	1	1	Ceiling Lamp	{"power": "off", "brightness": 50}

Рисунок 11 - Вывод примера 11.

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;
```

	AZ QUERY PLAN
1	Seq Scan on devices (cost=0.00..12.80 rows=280 width=262) (actual time=0.014..0.014 rows=4.00 loops=1)
2	Buffers: shared hit=1 dirtied=1
3	Planning Time: 0.036 ms
4	Execution Time: 0.023 ms

Рисунок 12 - Вывод примера 12.

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;
```

	AZ QUERY PLAN
1	Index Scan using devices_pkey on devices (cost=0.15..8.17 rows=1 width=262) (actual time=0.069..0.071 rows=1)
2	Index Cond: (id = 1)
3	Index Searches: 1
4	Buffers: shared hit=1 read=1
5	Planning Time: 0.052 ms
6	Execution Time: 0.088 ms

Рисунок 13 - Вывод примера 13.

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;
```

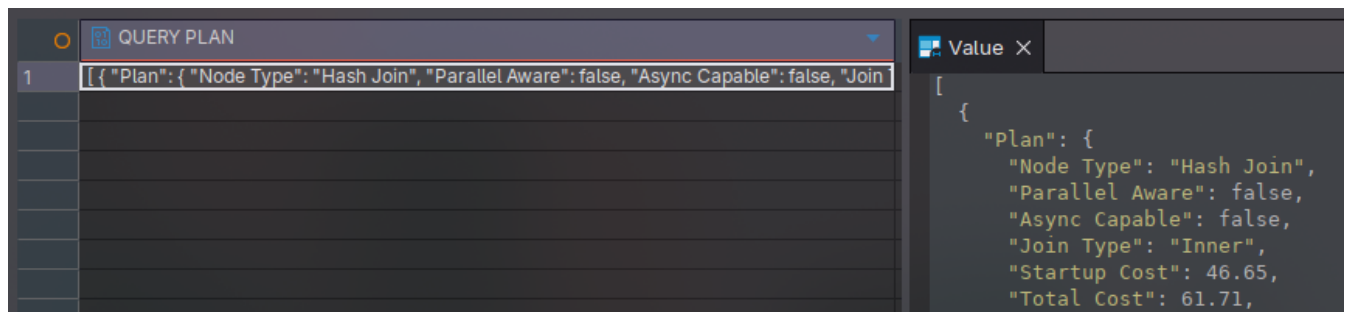


Рисунок 14 - Вывод примера 14.

```
[
{
  "Plan": {
    "Node Type": "Hash Join",
    "Parallel Aware": false,
    "Async Capable": false,
    "Join Type": "Inner",
    "Startup Cost": 46.65,
    "Total Cost": 61.71,
    "Plan Rows": 280,
    "Plan Width": 676,
    "Actual Startup Time": 0.100,
    "Actual Total Time": 0.104,
    "Actual Rows": 4.00,
    "Actual Loops": 1,
    "Disabled": false,
    "Inner Unique": true,
    "Hash Cond": "(h.user_id = u.id)",
    "Shared Hit Blocks": 4,
    "Shared Read Blocks": 0,
    "Shared Dirtied Blocks": 1,
    "Shared Written Blocks": 0,
    "Local Hit Blocks": 0,
    "Local Read Blocks": 0,
    "Local Dirtied Blocks": 0,

```

```

"Local Written Blocks": 0,
"Temp Read Blocks": 0,
"Temp Written Blocks": 0,
"Plans": [
  {
    "Node Type": "Hash Join",
    "Parent Relationship": "Outer",
    "Parallel Aware": false,
    "Async Capable": false,
    "Join Type": "Inner",
    "Startup Cost": 34.62,
    "Total Cost": 48.92,
    "Plan Rows": 280,
    "Plan Width": 562,
    "Actual Startup Time": 0.077,
    "Actual Total Time": 0.080,
    "Actual Rows": 4.00,
    "Actual Loops": 1,
    "Disabled": false,
    "Inner Unique": true,
    "Hash Cond": "(d.hub_id = h.id)",
    "Shared Hit Blocks": 3,
    "Shared Read Blocks": 0,
    "Shared Dirtied Blocks": 1,
    "Shared Written Blocks": 0,
    "Local Hit Blocks": 0,
    "Local Read Blocks": 0,
    "Local Dirtied Blocks": 0,
    "Local Written Blocks": 0,
    "Temp Read Blocks": 0,
    "Temp Written Blocks": 0,
    "Plans": [
      { ...

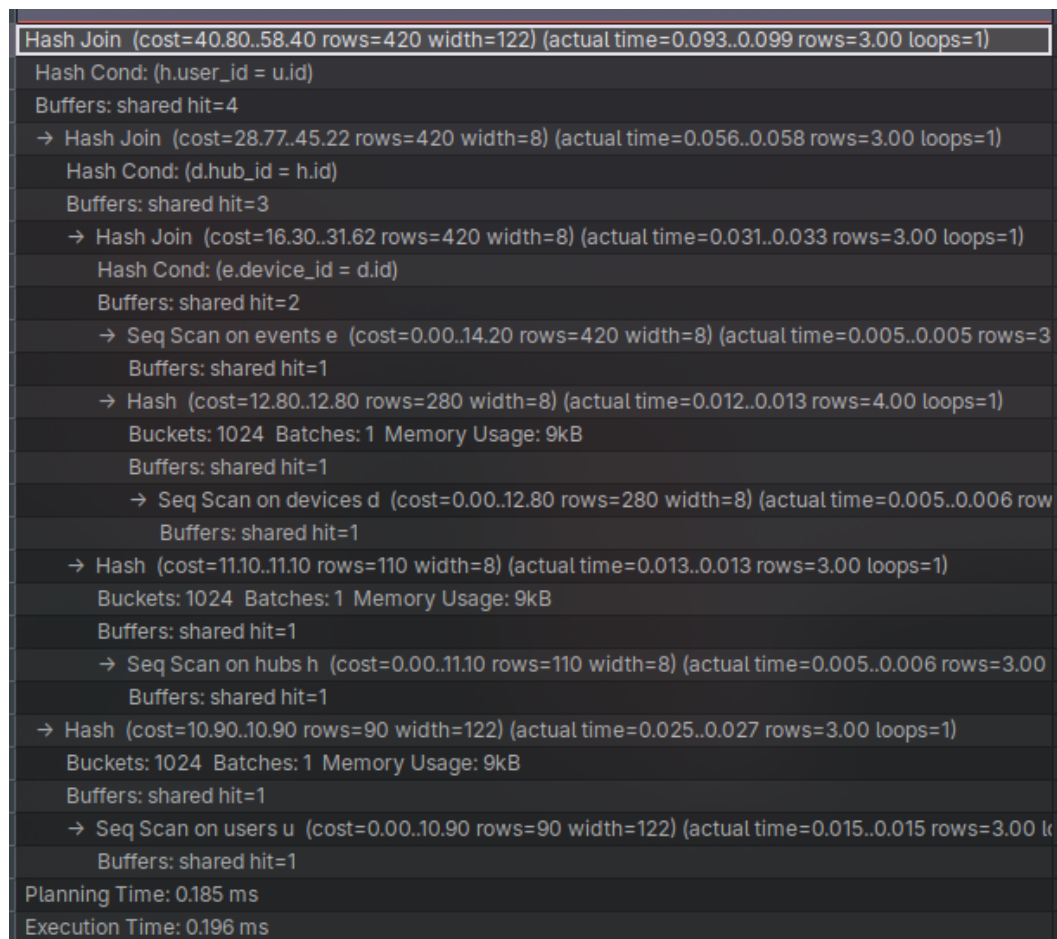
```

Пример 15: Анализ плана соединения четырёх таблиц

Запрос из Примера 10 соединяет четыре таблицы. PostgreSQL может выбрать разные стратегии соединения. Рассмотрим их.

По умолчанию (обычно Hash Join или Nested Loop)

```
explain (analyze, buffers)
select e.id, u.username
from events e
join devices d on e.device_id = d.id
join hubs h on d.hub_id = h.id
join users u on h.user_id = u.id;
```



Hash Join (cost=40.80..58.40 rows=420 width=122) (actual time=0.093..0.099 rows=3.00 loops=1)
Hash Cond: (h.user_id = u.id)
Buffers: shared hit=4
→ Hash Join (cost=28.77..45.22 rows=420 width=8) (actual time=0.056..0.058 rows=3.00 loops=1)
Hash Cond: (d.hub_id = h.id)
Buffers: shared hit=3
→ Hash Join (cost=16.30..31.62 rows=420 width=8) (actual time=0.031..0.033 rows=3.00 loops=1)
Hash Cond: (e.device_id = d.id)
Buffers: shared hit=2
→ Seq Scan on events e (cost=0.00..14.20 rows=420 width=8) (actual time=0.005..0.005 rows=3.00 loops=1)
Buffers: shared hit=1
→ Hash (cost=12.80..12.80 rows=280 width=8) (actual time=0.012..0.013 rows=4.00 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
Buffers: shared hit=1
→ Seq Scan on devices d (cost=0.00..12.80 rows=280 width=8) (actual time=0.005..0.006 rows=4.00 loops=1)
Buffers: shared hit=1
→ Hash (cost=11.10..11.10 rows=110 width=8) (actual time=0.013..0.013 rows=3.00 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
Buffers: shared hit=1
→ Seq Scan on hubs h (cost=0.00..11.10 rows=110 width=8) (actual time=0.005..0.006 rows=3.00 loops=1)
Buffers: shared hit=1
→ Hash (cost=10.90..10.90 rows=90 width=122) (actual time=0.025..0.027 rows=3.00 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
Buffers: shared hit=1
→ Seq Scan on users u (cost=0.00..10.90 rows=90 width=122) (actual time=0.015..0.015 rows=3.00 loops=1)
Buffers: shared hit=1
Planning Time: 0.185 ms
Execution Time: 0.196 ms

Рисунок 15.1 - Вывод примера 15 с Hash Join.

Принудительное использование Nested Loop

```
set enable_hashjoin = off;
set enable_mergejoin = off;
explain (analyze, buffers)
select e.id, u.username
from events e
join devices d on e.device_id = d.id
join hubs h on d.hub_id = h.id
join users u on h.user_id = u.id;
-- reset
set enable_hashjoin = on;
set enable_mergejoin = on;
```

	AZ QUERY PLAN
1	Hash Join (cost=40.80..58.40 rows=420 width=122) (actual time=0.089..0.095 rows=3.00 loops=1)
2	Hash Cond: (h.user_id = u.id)
3	Buffers: shared hit=4
4	→ Hash Join (cost=28.77..45.22 rows=420 width=8) (actual time=0.051..0.054 rows=3.00 loops=1)
5	Hash Cond: (d.hub_id = h.id)
6	Buffers: shared hit=3
7	→ Hash Join (cost=16.30..31.62 rows=420 width=8) (actual time=0.026..0.028 rows=3.00 loops=1)
8	Hash Cond: (e.device_id = d.id)
9	Buffers: shared hit=2
10	→ Seq Scan on events e (cost=0.00..14.20 rows=420 width=8) (actual time=0.005..0.005 rows=3.00 loops=1)
11	Buffers: shared hit=1
12	→ Hash (cost=12.80..12.80 rows=280 width=8) (actual time=0.012..0.012 rows=4.00 loops=1)
13	Buckets: 1024 Batches: 1 Memory Usage: 9kB
14	Buffers: shared hit=1
15	→ Seq Scan on devices d (cost=0.00..12.80 rows=280 width=8) (actual time=0.005..0.005 rows=4.00 loops=1)
16	Buffers: shared hit=1
17	→ Hash (cost=11.10..11.10 rows=110 width=8) (actual time=0.013..0.014 rows=3.00 loops=1)
18	Buckets: 1024 Batches: 1 Memory Usage: 9kB
19	Buffers: shared hit=1
20	→ Seq Scan on hubs h (cost=0.00..11.10 rows=110 width=8) (actual time=0.006..0.006 rows=3.00 loops=1)
21	Buffers: shared hit=1
22	→ Hash (cost=10.90..10.90 rows=90 width=122) (actual time=0.024..0.027 rows=3.00 loops=1)
23	Buckets: 1024 Batches: 1 Memory Usage: 9kB
24	Buffers: shared hit=1
25	→ Seq Scan on users u (cost=0.00..10.90 rows=90 width=122) (actual time=0.014..0.015 rows=3.00 loops=1)
26	Buffers: shared hit=1
27	Planning Time: 0.174 ms
28	Execution Time: 0.193 ms

Рисунок 15.2 - Вывод примера 15 с Nested Loop.

Принудительное использование Merge Join

```
set enable_hashjoin = off;
set enable_nestloop = off;
explain (analyze, buffers)
select e.id, u.username
from events e
join devices d on e.device_id = d.id
join hubs h on d.hub_id = h.id
join users u on h.user_id = u.id;
-- reset
set enable_hashjoin = on;
set enable_nestloop = on;
```

	AZ QUERY PLAN
1	Hash Join (cost=40.80..58.40 rows=420 width=122) (actual time=0.089..0.094 rows=3.00 loops=1)
2	Hash Cond: (h.user_id = u.id)
3	Buffers: shared hit=4
4	→ Hash Join (cost=28.77..45.22 rows=420 width=8) (actual time=0.053..0.056 rows=3.00 loops=1)
5	Hash Cond: (d.hub_id = h.id)
6	Buffers: shared hit=3
7	→ Hash Join (cost=16.30..31.62 rows=420 width=8) (actual time=0.030..0.031 rows=3.00 loops=1)
8	Hash Cond: (e.device_id = d.id)
9	Buffers: shared hit=2
10	→ Seq Scan on events e (cost=0.00..14.20 rows=420 width=8) (actual time=0.005..0.005 rows=3.00 loops=1)
11	Buffers: shared hit=1
12	→ Hash (cost=12.80..12.80 rows=280 width=8) (actual time=0.012..0.012 rows=4.00 loops=1)
13	Buckets: 1024 Batches: 1 Memory Usage: 9kB
14	Buffers: shared hit=1
15	→ Seq Scan on devices d (cost=0.00..12.80 rows=280 width=8) (actual time=0.005..0.005 rows=4.00 loops=1)
16	Buffers: shared hit=1
17	→ Hash (cost=11.10..11.10 rows=110 width=8) (actual time=0.013..0.013 rows=3.00 loops=1)
18	Buckets: 1024 Batches: 1 Memory Usage: 9kB
19	Buffers: shared hit=1
20	→ Seq Scan on hubs h (cost=0.00..11.10 rows=110 width=8) (actual time=0.005..0.005 rows=3.00 loops=1)
21	Buffers: shared hit=1
22	→ Hash (cost=10.90..10.90 rows=90 width=122) (actual time=0.021..0.023 rows=3.00 loops=1)
23	Buckets: 1024 Batches: 1 Memory Usage: 9kB
24	Buffers: shared hit=1
25	→ Seq Scan on users u (cost=0.00..10.90 rows=90 width=122) (actual time=0.011..0.012 rows=3.00 loops=1)
26	Buffers: shared hit=1
27	Planning Time: 0.164 ms
28	Execution Time: 0.181 ms

Рисунок 15.3 - Вывод примера с Merge Join.

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, включая сравнение различных стратегий соединения (`Nested Loop`, `Hash Join`, `Merge Join`) для запросов с участием нескольких таблиц, что является важным инструментом для оптимизации производительности базы данных.

Приложение A1. 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/01_init.sql
# Лаб 2.1 - Наполнение таблиц и представления
COPY lab2_1/code/init_data.sql /docker-entrypoint-initdb.d/02_init_data.sql
COPY lab2_1/code/views.sql /docker-entrypoint-initdb.d/03_views.sql
# Лаб 3 - пользовательские функции и триггеры
COPY lab3/code/functions_triggers.sql /docker-entrypoint-initdb.d/04_functions_triggers.sql
```

Приложение A2. SQL-скрипты

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;
```

```
-- 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

```
-- 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

```
-- 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
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

```
-- Пример 11
insert into device_types(type_name) values ('Smart Socket') returning id;
```

```
-- Пример 12
update devices set status = '{"power": "off", "brightness": 50}' where name = 'Ceiling Lamp';
```

```
-- Пример 13
explain (analyze, buffers) select * from devices;
```

```
-- Пример 14
explain (analyze, buffers) select * from devices where id = 1;
```

```
-- Пример 15 - Nested Loop
set enable_hashjoin = off; set enable_mergejoin = off;
explain (analyze, buffers) select e.id, u.username from events e join devices d on e.device_id = d.id;
set enable_hashjoin = on; set enable_mergejoin = on;
```

```
-- Пример 15 - Merge Join
set enable_hashjoin = off; set enable_nestloop = off;
explain (analyze, buffers) select e.id, u.username from events e join devices d on e.device_id = d.id;
set enable_hashjoin = on; set enable_nestloop = on;
```