

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №4 «Запросы на выборку и модификацию данных. Представления. Работа с индексами»

по дисциплине «Проектирование и реализация баз данных»

Автор: Хисаметдинова Д.Н.

Факультет: ИКТ

Группа: К3241

Преподаватель: Говорова М.М.



Санкт-Петербург 2023

Оглавление

Цель работы	3
Практическое задание	3
Вариант 17. БД «Телефонный провайдер»	3
Выполнение ЛР	Error! Bookmark not defined.
Вывод по работе	Error! Bookmark not defined. 35

Цель работы

Овладеть практическими навыками создания представлений и запросов на выборку данных к базе данных PostgreSQL, использования подзапросов при модификации данных и индексов.

Практическое задание

1. Создать запросы и представления на выборку данных к базе данных PostgreSQL (согласно индивидуальному заданию, часть 2 и 3).
2. Составить 3 запроса на модификацию данных (INSERT, UPDATE, DELETE) с использованием подзапросов.
3. Изучить графическое представление запросов и просмотреть историю запросов.
4. Создать простой и составной индексы для двух произвольных запросов и сравнить время выполнения запросов без индексов и с индексами. Для получения плана запроса использовать команду EXPLAIN.

Вариант 17. БД «Телефонный провайдер»

Описание предметной области: Информационная система служит для хранения информации об абонентах телефонной компании и для учета оплаты всех видов услуг абонентами.

Каждый абонент подключен к определенному тарифу. Тариф определяет базовое количество минут, ГВт, смс. Кроме того, он может подключить дополнительные услуги за отдельную плату. Необходимо знать текущий баланс клиента. У клиента могут быть подключены сторонние ресурсы, требующие оплаты, не зависящие от текущего тарифа.

Клиент может менять тариф.

В системе должны храниться сведения о продолжительности разговоров каждого абонента, о стоимости внутренних и междугородных переговоров, о задолженности абонента.

БД должна содержать следующий минимальный набор сведений: ФИО абонента. Номер телефона. Адрес абонента. Город. Зона (город, республика, СНГ, дальнее зарубежье). Страна. Стоимость тарифа. Сроки действия тарифа. Продолжительность разговора в минутах. Дата звонка. Время звонка. Код зоны. Цена минуты. Сумма оплаты. Дата оплаты. Статус оплаты. Дата фактической оплаты.

Задание 2. Создать запросы:

- Вывести суммарное время переговоров каждого абонента за заданный период.
- Найти среднюю продолжительность разговора абонента АТС.
- Вывести количество междугородных переговоров каждого абонента.
- Вывести список абонентов, не внёсших оплату за прошедший месяц.
- Сколько звонков было сделано в каждый из следующих городов: в Москву, Лондон, Париж.
- Вывести список абонентов, звонивших только в ночное время.

- Вывести список абонентов, время разговоров которых превышает среднее для этой же зоны.

Задание 3. Создать представление:

- Содержащее сведения обо всех абонентах и их переговорах за прошедший месяц.
- Найти самую популярную зону звонков за истекший год..

Задание 4. Создать хранимые процедуры:

- вывести список всех звонков заданного абонента.
- вывести задолженность по оплате для заданного абонента.
- рассчитать общую стоимость звонков по каждой зоне за истекшую неделю.

Задание 5. Создать необходимые триггеры.

Выполнение практического задания

Запросы:

1. Вывести суммарное время переговоров каждого абонента за заданный период.

```

2. SELECT
3.   client_id,
4.   SUM(duration_minutes) AS total_duration_minutes
5. FROM
6.   (
7.     SELECT
8.       cl.client_id,
9.       EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time)) / 60 AS
        duration_minutes
10.    FROM
11.      client cl
12.    JOIN contract ct ON cl.client_id = ct.client_id
13.    JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
14.    JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
15.   WHERE
16.     dc.call_start_time >= '2000-01-01' AND
17.     dc.call_end_time <= '2022-01-31'
18.
19.   UNION ALL
20.
21.   SELECT
22.     cl.client_id,
23.     EXTRACT(EPOCH FROM (ic.international_call_end_time -
        ic.international_call_start_time)) / 60 AS duration_minutes
24.   FROM
25.     client cl
26.   JOIN contract ct ON cl.client_id = ct.client_id
27.   JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
28.   JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number

```

```

29.     WHERE
30.         ic.international_call_start_time >= '2000-01-01' AND
31.         ic.international_call_end_time <= '2022-01-31'
32. ) combined_calls
33. GROUP BY
34.     client_id;
35.

```

phone_provider/postgres@PostgreSQL 14*

phone_provider/postgres@PostgreSQL 14

No limit

Query Query History

```

1 SELECT
2     client_id,
3     SUM(duration_minutes) AS total_duration_minutes
4 FROM
5     (
6         SELECT
7             cl.client_id,
8             EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time)) / 60 AS duration_minutes
9         FROM
10            client cl
11        JOIN contract ct ON cl.client_id = ct.client_id
12        JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
13        JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
14        WHERE
15            dc.call_start_time >= '2000-01-01' AND
16            dc.call_end_time <= '2022-01-31'
17    )

```

Data Output Messages Notifications

	client_id integer	total_duration_minutes numeric
1	87	11038480.100000000000
2	80	2934827.166666666666
3	52	7159503.233333333333
4	84	3956044.533333333333
5	92	5648564.416666666667
6	65	4469183.316666666667
7	98	20768440.916666666667

Total rows: 50 of 50 Query complete 00:00:00.048

Query: Query History			
Data Output		Messages	Notifications
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>			
	client_id integer	total_duration_minutes numeric	
1	87	11038480.100000000000	
2	80	2934827.166666666666	
3	52	7159503.233333333333	
4	84	3956044.533333333333	
5	92	5648564.416666666667	
6	65	4469183.316666666667	
7	98	20768440.916666666667	
8	44	12806766.549999999999	
9	42	9136645.716666666667	
10	88	4295562.100000000000	
11	40	36494772.766666666668	
12	9	14187327.766666666667	
13	15	15023788.283333333333	
14	79	14603069.083333333333	
15	26	2662828.533333333333	
16	48	14694175.916666666666	
17	85	11635341.133333333334	
18	19	10668135.483333333333	
19	77	2349072.333333333333	
20	28	17269371.266666666667	
21	37	31290450.500000000000	
Total rows: 50 of 50		Query complete 00:00:00.048	

2) Найти среднюю продолжительность разговора абонента АТС.

```

SELECT
  client_id,
  AVG(duration_minutes) AS average_call_duration,
  COUNT(*) AS total_calls,
  SUM(duration_minutes) AS total_duration_minutes
FROM
  (
    SELECT
      cl.client_id,
      EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time)) / 60 AS duration_minutes
    FROM
      client cl
    JOIN contract ct ON cl.client_id = ct.client_id
    JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
    JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
    WHERE
      dc.call_start_time >= '2000-01-01' AND

```

```

dc.call_end_time <= '2022-01-31' AND
cl.client_id = 87

UNION ALL

SELECT
    cl.client_id,
    EXTRACT(EPOCH FROM (ic.international_call_end_time -
ic.international_call_start_time)) / 60 AS duration_minutes
FROM
    client cl
JOIN contract ct ON cl.client_id = ct.client_id
JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
WHERE
    ic.international_call_start_time >= '2000-01-01' AND
    ic.international_call_end_time <= '2022-01-31' AND
    cl.client_id = 87
) combined_calls
GROUP BY
    client_id;

```

11038480 ÷ 3 =

3 679 493,3333333333

SQL phone_provide... phone_provider/postgres@PostgreSQL 14* Properties

phone_provider/postgres@PostgreSQL 14

Query Query History

```

1 SELECT
2   client_id,
3   AVG(duration_minutes) AS average_call_duration,
4   COUNT(*) AS total_calls,
5   SUM(duration_minutes) AS total_duration_minutes
6 FROM
7   (
8     SELECT
9       cl.client_id,
10      EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time)) / 60 AS duration_minutes
11    FROM
12      client cl
13    JOIN contract ct ON cl.client_id = ct.client_id
14    JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
15    JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
16    WHERE
17      dc.call_start_time >= '2000-01-01' AND
18      dc.call_end_time <= '2022-01-31' AND
19      cl.client_id = 87
20
21    UNION ALL
22
23    SELECT

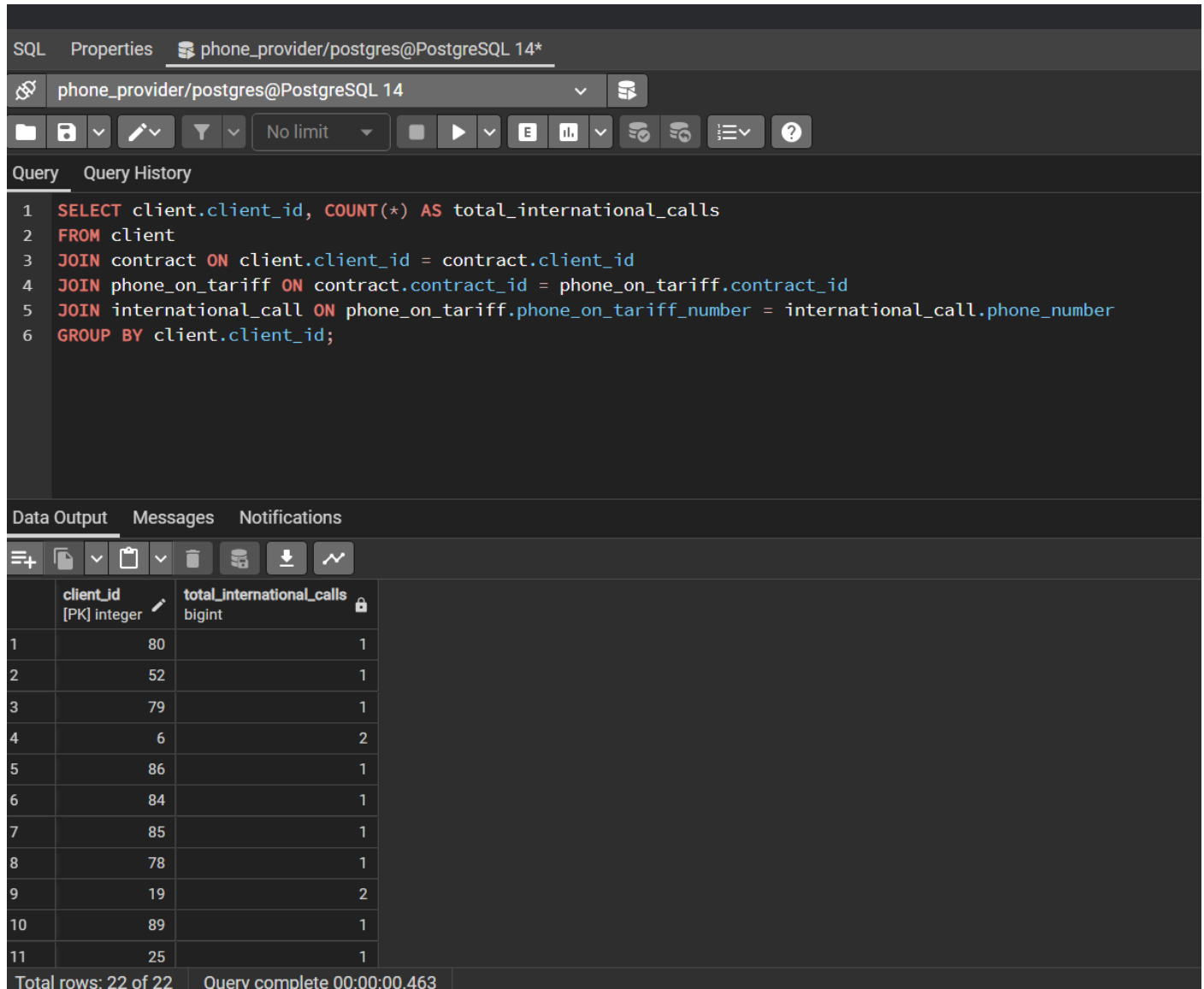
```

Data Output Messages Notifications

	client_id integer	average_call_duration numeric	total_calls bigint	total_duration_minutes numeric
1	87	3679493.36666666667	3	11038480.100000000000

3) Вывести количество международных переговоров каждого абонента.

```
SELECT client.client_id, COUNT(*) AS total_international_calls
FROM client
JOIN contract ON client.client_id = contract.client_id
JOIN phone_on_tariff ON contract.contract_id = phone_on_tariff.contract_id
JOIN international_call ON phone_on_tariff.phone_on_tariff_number =
international_call.phone_number
GROUP BY client.client_id;
```



The screenshot shows a PostgreSQL client interface with the following components:

- SQL Properties:** phone_provider/postgres@PostgreSQL 14*
- Query:** The same SQL query as in the previous block is entered.
- Data Output:** A table with 2 columns: `client_id` [PK] integer and `total_international_calls` bigint. The results are as follows:

	client_id	total_international_calls
1	80	1
2	52	1
3	79	1
4	6	2
5	86	1
6	84	1
7	85	1
8	78	1
9	19	2
10	89	1
11	25	1

Total rows: 22 of 22 Query complete 00:00:00.463

4) Вывести список абонентов, не внёсших оплату за прошедший месяц.

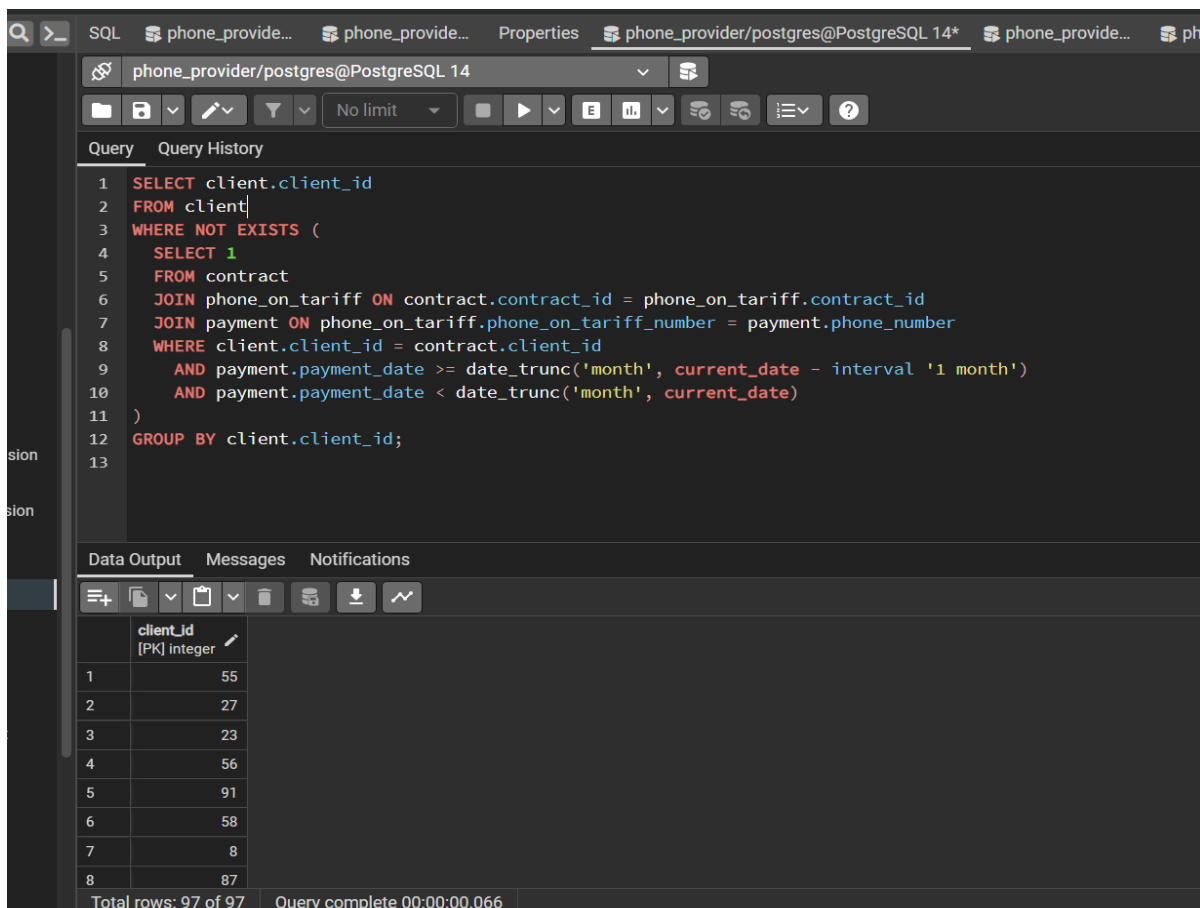
(в генераторе данных оплат в прошлом месяце вообще не было, поэтому добавила)

```
INSERT INTO "payment" ("payment_id", "payment_status", "phone_number", "payment_date",
"payment_amount") VALUES (101, True, '81839725897', '2023-10-13', '117');
INSERT INTO "payment" ("payment_id", "payment_status", "phone_number", "payment_date",
"payment_amount") VALUES (102, True, '85126663834', '2023-10-21', '535');
```



```
INSERT INTO "payment" ("payment_id", "payment_status", "phone_number", "payment_date",
"payment_amount") VALUES (103, True, '88686637097', '2023-10-18', '134');
```

```
SELECT client.client_id
FROM client
WHERE NOT EXISTS (
    SELECT 1
    FROM contract
    JOIN phone_on_tariff ON contract.contract_id = phone_on_tariff.contract_id
    JOIN payment ON phone_on_tariff.phone_on_tariff_number = payment.phone_number
    WHERE client.client_id = contract.client_id
        AND payment.payment_date >= date_trunc('month', current_date - interval '1 month')
        AND payment.payment_date < date_trunc('month', current_date)
)
GROUP BY client.client_id;
```



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is to 'phone_provider/postgres@PostgreSQL 14*'. The 'Query' tab is active, displaying the following SQL query:

```
1 SELECT client.client_id
2 FROM client
3 WHERE NOT EXISTS (
4     SELECT 1
5     FROM contract
6     JOIN phone_on_tariff ON contract.contract_id = phone_on_tariff.contract_id
7     JOIN payment ON phone_on_tariff.phone_on_tariff_number = payment.phone_number
8     WHERE client.client_id = contract.client_id
9         AND payment.payment_date >= date_trunc('month', current_date - interval '1 month')
10        AND payment.payment_date < date_trunc('month', current_date)
11 )
12 GROUP BY client.client_id;
13
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with 8 rows and 1 column, 'client_id'. The values are: 55, 27, 23, 56, 91, 58, 8, and 87. The status bar at the bottom indicates 'Total rows: 97 of 97' and 'Query complete 00:00:00.066'.

client_id [PK] integer	
1	55
2	27
3	23
4	56
5	91
6	58
7	8
8	87

5) Сколько звонков было сделано в каждый из следующих городов: в Москву, Лондон, Париж.

```
INSERT INTO "call_zone" ("call_zone_id", "country", "cost_per_minute_call_zone", "region",
"city") VALUES (16, 'Russia', '1', 'Central Russia', 'Moscow');
```

```

INSERT INTO "call_zone" ("call_zone_id", "country", "cost_per_minute_call_zone", "region",
"city") VALUES (17, 'United Kingdom', '100', 'England', 'London');
INSERT INTO "call_zone" ("call_zone_id", "country", "cost_per_minute_call_zone", "region",
"city") VALUES (18, 'France', '80', 'Paris region', 'Paris');

INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (31, '84925489447', '2001-05-03
16:51:26', '2001-05-03 17:40:57', '84034176596', 16);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (32, '87972625997', '2016-09-23
12:16:19', '2016-09-23 13:01:25', '89991677667', 16);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (33, '84371531905', '2000-09-04
00:13:32', '2000-09-04 00:16:09', '86754876851', 18);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (34, '89622760197', '2017-08-09
05:11:22', '2017-08-09 05:13:39', '88803311209', 17);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (35, '89742497510', '2010-06-22
01:23:40', '2010-06-22 04:38:35', '82307566768', 16);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (36, '81539108668', '2004-10-03
02:29:31', '2004-10-03 10:04:15', '86300043270', 17);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (37, '88686637097', '2021-03-27
07:16:38', '2021-03-27 16:31:09', '81517335555', 16);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (38, '83164665019', '2003-01-05
22:58:43', '2003-01-05 23:21:03', '88963847481', 18);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (39, '84157873461', '2022-09-25
12:29:38', '2022-09-25 16:32:10', '80680854021', 16);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (40, '88686637097', '2023-01-11
20:18:06', '2023-01-11 22:09:58', '88427512477', 18);

```

```

-- Для Москвы
SELECT COUNT(*) AS total_calls_to_moscow
FROM international_call
WHERE call_zone_id = (SELECT call_zone_id FROM call_zone WHERE city = 'Moscow');

-- Для Лондона
SELECT COUNT(*) AS total_calls_to_london
FROM international_call
WHERE call_zone_id = (SELECT call_zone_id FROM call_zone WHERE city = 'London');

-- Для Парижа
SELECT COUNT(*) AS total_calls_to_paris
FROM international_call
WHERE call_zone_id = (SELECT call_zone_id FROM call_zone WHERE city = 'Paris');

```

SQL Properties phone_provide... phone_provider/postgres@PostgreSQL 14*

phone_provider/postgres@PostgreSQL 14

Query Query History

```

1 -- Для Москвы
2 SELECT COUNT(*) AS total_calls_to_moscow
3 FROM international_call
4 WHERE call_zone_id = (SELECT call_zone_id FROM call_zone WHERE city = 'Moscow');
5

```

Data Output Messages Notifications

	total_calls_to_moscow	
	bigint	
1		5

SQL Properties phone_provider/postgres@PostgreSQL 14*

phone_provider/postgres@PostgreSQL 14

Query Query History

```
1 -- Для Лондона
2 SELECT COUNT(*) AS total_calls_to_london
3 FROM international_call
4 WHERE call_zone_id = (SELECT call_zone_id FROM call_zone WHERE city = 'London');
```

Data Output Messages Notifications

	total_calls_to_london	
	bigint	
1		2

SQL Properties phone_provider/postgres@PostgreSQL 14*

phone_provider/postgres@PostgreSQL 14

Query Query History

```
1 -- Для Парижа
2 SELECT COUNT(*) AS total_calls_to_paris
3 FROM international_call
4 WHERE call_zone_id = (SELECT call_zone_id FROM call_zone WHERE city = 'Paris');
```

Data Output Messages Notifications

	total_calls_to_paris	
	bigint	
1		3

6) Вывести список абонентов, звонивших только в ночное время.

```
SELECT
    client_id,
    SUM(duration_minutes) AS total_night_duration_minutes
FROM
    (
        SELECT
            cl.client_id,
            EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time)) / 60 AS
duration_minutes,
            dc.call_start_time AT TIME ZONE 'UTC' as start_time_utc,
            dc.call_end_time AT TIME ZONE 'UTC' as end_time_utc
        FROM
            client cl
        JOIN contract ct ON cl.client_id = ct.client_id
        JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
        JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
        WHERE
            (dc.call_start_time::time >= '22:00' OR dc.call_start_time::time < '06:00') AND
            (dc.call_end_time::time >= '22:00' OR dc.call_end_time::time < '06:00')

        UNION ALL

        SELECT
            cl.client_id,
            EXTRACT(EPOCH FROM (ic.international_call_end_time -
ic.international_call_start_time)) / 60 AS duration_minutes,
            ic.international_call_start_time AT TIME ZONE 'UTC' as start_time_utc,
            ic.international_call_end_time AT TIME ZONE 'UTC' as end_time_utc
        FROM
            client cl
        JOIN contract ct ON cl.client_id = ct.client_id
        JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
        JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
        WHERE
            (ic.international_call_start_time::time >= '22:00' OR
ic.international_call_start_time::time < '06:00') AND
            (ic.international_call_end_time::time >= '22:00' OR
ic.international_call_end_time::time < '06:00')
    ) combined_calls
GROUP BY
    client_id;
```

SQL phone_provide... phone_provide... Properties phone_provide... phone_provider/postgres@PostgreSQL 14*

phone_provider/postgres@PostgreSQL 14

No limit

Query Query History

```

1 SELECT
2   client_id,
3   SUM(duration_minutes) AS total_night_duration_minutes
4 FROM
5   (
6     SELECT
7       cl.client_id,
8       EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time)) / 60 AS duration_minutes,
9       dc.call_start_time AT TIME ZONE 'UTC' as start_time_utc,
10      dc.call_end_time AT TIME ZONE 'UTC' as end_time_utc
11    FROM
12      client cl
13    JOIN contract ct ON cl.client_id = ct.client_id
14    JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
15    JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
16   WHERE
17     (dc.call_start_time::time >= '22:00' OR dc.call_start_time::time < '06:00') AND
18     (dc.call_end_time::time >= '22:00' OR dc.call_end_time::time < '06:00')
19   UNION ALL
20   SELECT
21     ...
22   ...

```

Data Output Messages Notifications

	client_id integer	total_night_duration_minutes numeric
1	74	5159516.90000000000000
2	45	24496.18333333333333
3	6	2698767.15000000000000

Total rows: 15 of 15 Query complete 00:00:00.047

Data Output Messages Notifications			
	client_id integer	total_night_duration_minutes numeric	
1	74	5159516.900000000000	
2	45	24496.183333333333	
3	6	2698767.150000000000	
4	86	5039812.600000000000	
5	13	1319098.050000000000	
6	98	3025256.950000000000	
7	100	7932887.500000000000	
8	15	6146401.766666666666	
9	12	1857960.866666666667	
10	85	6479994.366666666667	
11	49	3093161.700000000000	
12	28	1394020.750000000000	
13	37	6159990.666666666666	
14	76	12105870.083333333334	
15	23	1566717.516666666667	

7) Вывести список абонентов, время разговоров которых превышает среднее для этой же зоны.

(Запрос, выводящий и client_id, и среднюю продолжительность в его call_zone, и продолжительность конкретного звонка)

```
WITH CallDurations AS (
    SELECT
        cl.client_id,
        ic.call_zone_id,
        EXTRACT(EPOCH FROM (ic.international_call_end_time -
ic.international_call_start_time)) / 60 AS call_duration_minutes
    FROM
        client cl
    JOIN contract ct ON cl.client_id = ct.client_id
    JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
    JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
),
AverageDurations AS (
    SELECT
        call_zone_id,
        AVG(call_duration_minutes) AS avg_call_duration_minutes
    FROM
        CallDurations
    GROUP BY
        call_zone_id
)
```

```

        call_zone_id
    )
SELECT
    cd.client_id,
    cd.call_duration_minutes,
    ad.avg_call_duration_minutes
FROM
    CallDurations cd
JOIN
    AverageDurations ad ON cd.call_zone_id = ad.call_zone_id
WHERE
    cd.call_duration_minutes > ad.avg_call_duration_minutes;

```

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```

1  WITH CallDurations AS (
2      SELECT
3          cl.client_id,
4          ic.call_zone_id,
5          EXTRACT(EPOCH FROM (ic.international_call_end_time - ic.international_call_start_time)) / 60 AS call_duration_minutes
6      FROM
7          client cl
8      JOIN contract ct ON cl.client_id = ct.client_id
9      JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
10     JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
11 ),
12 AverageDurations AS (
13     SELECT
14         call_zone_id,
15         AVG(call_duration_minutes) AS avg_call_duration_minutes
16     FROM
17         CallDurations
18 )
19 SELECT
20     cd.client_id,
21     cd.call_duration_minutes,
22     ad.avg_call_duration_minutes
23 FROM
24     CallDurations cd
25 JOIN
26     AverageDurations ad ON cd.call_zone_id = ad.call_zone_id
27 WHERE
28     cd.call_duration_minutes > ad.avg_call_duration_minutes;

```

The results are displayed in a table with the following columns: client_id [PK] integer, call_duration_minutes numeric, and avg_call_duration_minutes numeric. The table contains 7 rows of data.

	client_id [PK] integer	call_duration_minutes numeric	avg_call_duration_minutes numeric
1	6	111.86666666666667	45.60555555555556
2	6	554.5166666666667	217.3166666666667
3	6	8455491.866666667	5876829.291666667
4	19	8085102.033333333	7428328.275000000
5	25	8163602.616666667	5797670.820833333
6	28	9272273.750000000	6918525.875000000
7	31	454.7333333333333	228.5083333333333

Total rows: 16 of 16. Query complete: 00:00:00.151

(Запрос, непосредственно выводящий только client_id)

```

WITH CallDurations AS (
    SELECT
        cl.client_id,
        ic.call_zone_id,
        (EXTRACT(EPOCH FROM (ic.international_call_end_time -
ic.international_call_start_time)) / 60) AS call_duration_minutes
    FROM
        client cl
    JOIN contract ct ON cl.client_id = ct.client_id
    JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id

```



```

        JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
    ),
    AverageDurations AS (
        SELECT
            call_zone_id,
            AVG(call_duration_minutes) AS avg_call_duration_minutes
        FROM
            CallDurations
        GROUP BY
            call_zone_id
    )
SELECT DISTINCT
    cd.client_id
FROM
    CallDurations cd
JOIN
    AverageDurations ad ON cd.call_zone_id = ad.call_zone_id
WHERE
    cd.call_duration_minutes > ad.avg_call_duration_minutes;

```

SQL Properties phone_provider/postgres@PostgreSQL 14

phone_provider/postgres@PostgreSQL 14

Query Query History

```

1 WITH CallDurations AS (
2     SELECT
3         cl.client_id,
4         ic.call_zone_id,
5         (EXTRACT(EPOCH FROM (ic.international_call_end_time - ic.international_call_start_time)) / 60) AS call_duration_minutes
6     FROM
7         client cl
8     JOIN contract ct ON cl.client_id = ct.client_id
9     JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
10    JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
11 ),
12 AverageDurations AS (
13     SELECT
14         call_zone_id,
15         AVG(call_duration_minutes) AS avg_call_duration_minutes
16     FROM
17         CallDurations
18 )
19 SELECT DISTINCT
20     cd.client_id
21 FROM
22     CallDurations cd
23 JOIN
24     AverageDurations ad ON cd.call_zone_id = ad.call_zone_id
25 WHERE
26     cd.call_duration_minutes > ad.avg_call_duration_minutes;

```

Data Output Messages Notifications

	client_id [PK] integer
1	79
2	6
3	49
4	75
5	52
6	56
7	28

Total rows: 14 of 14 Query complete 00:00:00.129

Задание 3. Создать представление:

- 1) Содержащее сведения обо всех абонентах и их переговорах за прошедший месяц.

(так как в прошлый месяц не было звонков, добавила INSERTами)

```
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (41, '88686637097', '2023-10-01  
20:18:06', '2023-10-01 20:48:06', '88427505477', 2);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (42, '83939283170', '2023-10-23  
16:19:05', '2023-10-23 16:39:05', '88007805173', 4);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (43, '84548540986', '2023-10-26  
12:17:24', '2023-10-26 13:15:24', '83950008013', 5);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (44, '89175874533', '2023-10-27  
18:25:57', '2023-10-27 19:25:57', '86924801576', 6);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (45, '87395751914', '2023-10-03  
11:42:40', '2023-10-03 18:42:40', '81028402443', 14);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (46, '84608960717', '2023-10-02  
12:38:06', '2023-10-02 14:58:06', '86600198499', 15);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (47, '83495694331', '2023-10-09  
16:04:02', '2023-10-09 16:34:02', '85525676401', 29);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (48, '87816542499', '2023-10-08  
12:42:37', '2023-10-08 16:42:37', '85666856990', 5);  
INSERT INTO "international_call" ("international_call_id", "phone_number",  
"international_call_start_time", "international_call_end_time",  
"international_callee_number", "call_zone_id") VALUES (49, '81341681439', '2023-10-06  
04:05:38', '2023-10-06 05:05:30', '84662915076', 5);
```

```
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",  
"call_end_time", "domestic_callee_number") VALUES (201, '87127921928', '2023-10-03  
21:52:31', '2023-10-03 23:52:31', '88125094548');
```

```

INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (202, '81290751976', '2023-10-01
03:51:21', '2023-10-01 04:51:21', '89415803323');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (203, '86948251411', '2023-10-04
11:14:04', '2023-10-04 12:14:04', '89246801095');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (204, '82884816243', '2023-10-08
10:16:29', '2023-10-08 12:15:29', '84557604220');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (205, '86948251411', '2023-10-02
10:21:40', '2023-10-02 12:25:45', '83833805987');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (206, '83253580355', '2023-10-09
05:58:30', '2023-10-09 05:59:30', '88939347162');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (207, '85411706214', '2023-10-06
02:15:38', '2023-10-06 03:10:38', '81453154633');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (208, '82651331557', '2023-10-12
17:14:15', '2023-10-12 19:16:15', '85768773631');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (209, '81351445183', '2023-10-05
04:01:07', '2023-10-05 09:01:07', '82303663629');
INSERT INTO "domestic_call" ("domestic_call_id", "phone_number", "call_start_time",
"call_end_time", "domestic_callee_number") VALUES (210, '84445401791', '2023-10-07
00:08:53', '2023-10-07 01:08:53', '81655367593');

```

(Текст запроса на создание VIEW)

```

CREATE OR REPLACE VIEW client_calls_summary AS
SELECT
    cl.client_id,
    COALESCE(SUM(dc.duration_minutes), 0) AS total_domestic_duration_minutes,
    COALESCE(SUM(ic.duration_minutes), 0) AS total_international_duration_minutes,
    COALESCE(COUNT(dc.domestic_call_id), 0) AS number_of_domestic_calls,
    COALESCE(COUNT(ic.international_call_id), 0) AS number_of_international_calls,
    STRING_AGG(DISTINCT cz.country, ', ') AS countries_called -- список стран международных
ЗВОНКОВ
FROM
    client cl
JOIN contract ct ON cl.client_id = ct.client_id
JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
LEFT JOIN (
    SELECT domestic_call_id, phone_number, EXTRACT(EPOCH FROM (call_end_time -
call_start_time)) / 60 AS duration_minutes
    FROM domestic_call
    WHERE call_start_time >= date_trunc('month', CURRENT_DATE - INTERVAL '1 month') AND
        call_start_time < date_trunc('month', CURRENT_DATE)
) dc ON pt.phone_on_tariff_number = dc.phone_number

```

```

LEFT JOIN (
  SELECT international_call_id, phone_number, call_zone_id, EXTRACT(EPOCH FROM
(international_call_end_time - international_call_start_time)) / 60 AS duration_minutes
  FROM international_call
  WHERE international_call_start_time >= date_trunc('month', CURRENT_DATE - INTERVAL '1
month') AND
    international_call_start_time < date_trunc('month', CURRENT_DATE)
) ic ON pt.phone_on_tariff_number = ic.phone_number
LEFT JOIN call_zone cz ON ic.call_zone_id = cz.call_zone_id
GROUP BY cl.client_id;

```

The screenshot shows a PostgreSQL query editor with a query that has been executed successfully. The query is a complex SQL statement involving multiple joins and aggregations. The message pane at the bottom indicates that the query was returned successfully in 112 msec.

```

11 JOIN contract ct ON ct.contract_id = cl.contract_id
12 JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
13 LEFT JOIN (
14   SELECT domestic_call_id, phone_number, EXTRACT(EPOCH FROM (call_end_time - call_start_time)) / 60 AS duration_minutes
15   FROM domestic_call
16   WHERE call_start_time >= date_trunc('month', CURRENT_DATE - INTERVAL '1 month') AND
17     call_start_time < date_trunc('month', CURRENT_DATE)
18 ) dc ON pt.phone_on_tariff_number = dc.phone_number
19 LEFT JOIN (
20   SELECT international_call_id, phone_number, call_zone_id, EXTRACT(EPOCH FROM (international_call_end_time - international_call_start_time)) / 60
21   FROM international_call
22   WHERE international_call_start_time >= date_trunc('month', CURRENT_DATE - INTERVAL '1 month') AND
23     international_call_start_time < date_trunc('month', CURRENT_DATE)
24 ) ic ON pt.phone_on_tariff_number = ic.phone_number
25 LEFT JOIN call_zone cz ON ic.call_zone_id = cz.call_zone_id
26 GROUP BY cl.client_id;
27
Data Output Messages Notifications
CREATE VIEW
Query returned successfully in 112 msec.

```

The screenshot shows a PostgreSQL query editor with a simple query executed. The result is displayed in a table with 7 columns: client_id, total_domestic_duration_minutes, total_international_duration_minutes, number_of_domestic_calls, number_of_international_calls, and countries_called. The table contains 6 rows of data, with the last row being the total for all clients.

```

1 SELECT * FROM client_calls_summary;

```

	client_id	total_domestic_duration_minutes	total_international_duration_minutes	number_of_domestic_calls	number_of_international_calls	countries_called
33	48	0	0	0	0	[null]
34	49	0	0	0	0	[null]
35	50	0	0	0	0	[null]
36	52	0	60.0000000000000000	0	1	Liechtenstein
37	53	0	0	0	0	[null]
38	55	300.0000000000000000	0	1	0	[null]
39	56	119.0000000000000000	0	1	0	[null]
Total rows: 65 of 65 Query complete 00:00:00.138						

2) Найти самую популярную зону звонков за истекший год.

(Вставляю новые даты за прошлый год)

```
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (50, '88686637097', '2022-10-01
20:18:06', '2022-10-01 20:48:06', '88427505478', 10);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (51, '83939283170', '2022-01-23
16:19:05', '2022-01-23 16:39:05', '88007805178', 4);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (52, '84548540986', '2022-12-26
12:17:24', '2022-12-26 13:15:24', '83950008018', 5);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (53, '89175874533', '2022-08-27
18:25:57', '2022-08-27 19:25:57', '86924801578', 2);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (54, '87395751914', '2022-05-03
11:42:40', '2022-05-03 18:42:40', '81028402448', 10);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (55, '84608960717', '2022-03-02
12:38:06', '2022-03-02 14:58:06', '86600198498', 10);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (56, '83495694331', '2022-07-09
16:04:02', '2022-07-09 16:34:02', '85525676408', 10);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (57, '87816542499', '2022-04-08
12:42:37', '2022-04-08 16:42:37', '85666856998', 5);
INSERT INTO "international_call" ("international_call_id", "phone_number",
"international_call_start_time", "international_call_end_time",
"international_callee_number", "call_zone_id") VALUES (58, '81341681439', '2022-06-06
04:05:38', '2022-06-06 05:05:30', '84662915078', 5);
```

```
CREATE OR REPLACE VIEW most_popular_call_zone_yearly AS
SELECT
    cz.call_zone_id,
    cz.city,
    cz.country,
    COUNT(*) AS call_count
FROM
```

```

international_call ic
JOIN call_zone cz ON ic.call_zone_id = cz.call_zone_id
WHERE
ic.international_call_start_time >= date_trunc('year', CURRENT_DATE - INTERVAL '1 year')
AND
ic.international_call_start_time < date_trunc('year', CURRENT_DATE)
GROUP BY
cz.call_zone_id, cz.city, cz.country
ORDER BY
call_count DESC
LIMIT 1;

```

The screenshot shows a PostgreSQL IDE interface with a dark theme. The top toolbar includes icons for file operations, query execution, and settings. The main editor displays a SQL query to create or replace a view named 'most_popular_call_zone_yearly'. The query selects columns from 'call_zone' and counts international calls, filtered by a one-year time window. The bottom panel shows the 'Messages' tab with a confirmation message: 'Query returned successfully in 65 msec.'

```

1 CREATE OR REPLACE VIEW most_popular_call_zone_yearly AS
2 SELECT
3   cz.call_zone_id,
4   cz.city,
5   cz.country,
6   COUNT(*) AS call_count
7 FROM
8   international_call ic
9 JOIN call_zone cz ON ic.call_zone_id = cz.call_zone_id
10 WHERE
11   ic.international_call_start_time >= date_trunc('year', CURRENT_DATE - INTERVAL '1 year') AND
12   ic.international_call_start_time < date_trunc('year', CURRENT_DATE)
13 GROUP BY
14   cz.call_zone_id, cz.city, cz.country
15 ORDER BY
16   call_count DESC
17 LIMIT 1;
18
Data Output  Messages  Notifications
CREATE VIEW
Query returned successfully in 65 msec.

```

SQL Properties phone_provider/postgres@PostgreSQL 14

Query Query History

```
1 SELECT * FROM most_popular_call_zone_yearly;
```

Data Output Messages Notifications

	call_zone_id integer	city character varying (60)	country character varying (60)	call_count bigint
1	10	Stacyport	Poland	4

Создание INSERT, UPDATE и DELETE запросов

UPDATE запрос

```
UPDATE phone_on_tariff
SET current_balance = current_balance + 500
WHERE tariff_id IN (
    SELECT tariff_id FROM basic_tariff WHERE gb_amount < 100
)
AND phone_on_tariff_number IN (
    SELECT pot.phone_on_tariff_number
    FROM phone_on_tariff pot
    JOIN internal_service_inclusion isi ON pot.phone_on_tariff_number = isi.phone_number
    JOIN internal_service isv ON isi.internal_service_id = isv.internal_service_id
    WHERE isv.internal_service_periodicity = 'yearly'
)
```

```

AND phone_on_tariff_number NOT IN (
  -- Исключаю номера с количеством звонков выше среднего
  SELECT dc.phone_number
  FROM domestic_call dc
  GROUP BY dc.phone_number
  HAVING AVG(EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time))) > (
    SELECT AVG(EXTRACT(EPOCH FROM (d.call_end_time - d.call_start_time)))
    FROM domestic_call d
  )
);

```

The screenshot shows a PostgreSQL client window titled 'phone_provider/postgres@PostgreSQL 14'. The interface includes a toolbar with icons for file operations, query execution, and settings. The 'Query' tab is active, displaying a SQL query that updates the 'current_balance' of phone tariffs. The query is as follows:

```

1 UPDATE phone_on_tariff
2 SET current_balance = current_balance + 500
3 WHERE tariff_id IN (
4   SELECT tariff_id FROM basic_tariff WHERE gb_amount < 100
5 )
6 AND phone_on_tariff_number IN (
7   SELECT pot.phone_on_tariff_number
8   FROM phone_on_tariff pot
9   JOIN internal_service_inclusion isi ON pot.phone_on_tariff_number = isi.phone_number
10  JOIN internal_service isv ON isi.internal_service_id = isv.internal_service_id
11  WHERE isv.internal_service_periodicity = 'yearly'
12 )
13 AND phone_on_tariff_number NOT IN (
14   -- Исключаю номера с количеством звонков выше среднего
15   SELECT dc.phone_number
16   FROM domestic_call dc
17   GROUP BY dc.phone_number
18   HAVING AVG(EXTRACT(EPOCH FROM (dc.call_end_time - dc.call_start_time))) > (
19     SELECT AVG(EXTRACT(EPOCH FROM (d.call_end_time - d.call_start_time)))
20     FROM domestic_call d
21   )
22 );

```

Below the query, the 'Data Output' tab shows the result: 'UPDATE 5'. A message at the bottom states: 'Query returned successfully in 73 msec.'

INSERT

```

INSERT INTO payment (phone_number, payment_date, payment_amount, payment_status)
SELECT
  pot.phone_on_tariff_number,
  CURRENT_DATE,
  100.00,
  true
FROM
  phone_on_tariff pot
JOIN contract c ON pot.contract_id = c.contract_id
JOIN client cl ON c.client_id = cl.client_id

```



```

WHERE
    pot.tariff_id IN (SELECT tariff_id FROM basic_tariff WHERE gb_amount > 10)
AND NOT EXISTS (
    SELECT 1 FROM payment p
    WHERE p.phone_number = pot.phone_on_tariff_number
        AND p.payment_date = CURRENT_DATE
);

```

The screenshot shows a PostgreSQL IDE window with the title 'phone_provider/postgres@PostgreSQL 14*'. The 'Query' tab is active, displaying the following SQL script:

```

1 INSERT INTO payment (phone_number, payment_date, payment_amount, payment_status)
2 SELECT
3     pot.phone_on_tariff_number,
4     CURRENT_DATE,
5     100.00,
6     true
7 FROM
8     phone_on_tariff pot
9 JOIN contract c ON pot.contract_id = c.contract_id
10 JOIN client cl ON c.client_id = cl.client_id
11 WHERE
12     pot.tariff_id IN (SELECT tariff_id FROM basic_tariff WHERE gb_amount > 10)
13 AND NOT EXISTS (
14     SELECT 1 FROM payment p
15     WHERE p.phone_number = pot.phone_on_tariff_number
16         AND p.payment_date = CURRENT_DATE
17 );
18
19

```

The 'Messages' tab at the bottom shows the execution results:

```

INSERT 0 194

Query returned successfully in 103 msec.

```

A green status bar at the bottom right indicates: 'Successfully run. Total query runtime: 121 msec. 1 rows affected.'

DELETE

```

DELETE FROM contract
WHERE client_id IN (
    SELECT cl.client_id
    FROM client cl
    JOIN phone_on_tariff pot ON cl.client_id = (
        SELECT ct.client_id
        FROM contract ct
        WHERE ct.client_id = cl.client_id
        LIMIT 1
    )
    LEFT JOIN external_service_inclusion esi ON pot.phone_on_tariff_number =
esi.phone_number
    WHERE esi.external_service_id IS NULL -- Нет активных услуг
        AND pot.current_balance < (
            SELECT AVG(p.current_balance)

```

```

        FROM phone_on_tariff p
        WHERE p.tariff_id = pot.tariff_id
    )
    AND pot.deletion_date IS NULL -- Активные тарифы
)
AND NOT EXISTS (
    -- Убедимся, что у клиента нет активных звонков
    SELECT 1
    FROM phone_on_tariff pt
    JOIN domestic_call dc ON pt.phone_on_tariff_number = dc.phone_number
    WHERE pt.contract_id = contract.contract_id
        AND dc.call_start_time > CURRENT_DATE - INTERVAL '1 year'
);

```

Создание индексов

1) Оператор EXPLAIN ANALYZE в запросе без индексов:

```

• EXPLAIN ANALYZE WITH CallDurations AS (
•     SELECT
•         cl.client_id,
•         ic.call_zone_id,
•         EXTRACT(EPOCH FROM (ic.international_call_end_time -
ic.international_call_start_time)) / 60 AS call_duration_minutes
•     FROM
•         client cl
•     JOIN contract ct ON cl.client_id = ct.client_id
•     JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
•     JOIN international_call ic ON pt.phone_number = ic.phone_number
• ),
• AverageDurations AS (
•     SELECT
•         call_zone_id,
•         AVG(call_duration_minutes) AS avg_call_duration_minutes
•     FROM
•         CallDurations
•     GROUP BY
•         call_zone_id
• )
• SELECT DISTINCT
•     cd.client_id
• FROM
•     CallDurations cd
• JOIN
•     AverageDurations ad ON cd.call_zone_id = ad.call_zone_id
• WHERE
•     cd.call_duration_minutes > ad.avg_call_duration_minutes;
•

```

phone_provider/postgres@PostgreSQL 14

Query Query History

```

1 EXPLAIN ANALYZE WITH CallDurations AS (
2     SELECT
3         cl.client_id,
4         ic.call_zone_id,
5         EXTRACT(EPOCH FROM (ic.international_call_end_time - ic.international_call_start_time)) / 60 AS call_duration_minutes
6     FROM
7         client cl
8     JOIN contract ct ON cl.client_id = ct.client_id
9     JOIN phone_on_tariff pt ON ct.contract_id = pt.contract_id
10    JOIN international_call ic ON pt.phone_on_tariff_number = ic.phone_number
11 ),
12 AverageDurations AS (
13     SELECT
14         call_zone_id,
15         AVG(call_duration_minutes) AS avg_call_duration_minutes
16     FROM
17         CallDurations
18 )

```

Data Output Messages Notifications

QUERY PLAN

text

1	HashAggregate (cost=24.98..25.17 rows=19 width=4) (actual time=0.267..0.269 rows=16 loops=1)
2	Group Key: cd.client_id
3	Batches: 1 Memory Usage: 24kB
4	CTE calldurations
5	-> Hash Join (cost=14.21..20.14 rows=58 width=40) (actual time=0.130..0.187 rows=58 loops=1)
6	Hash Cond: (ct.client_id = cl.client_id)
7	-> Hash Join (cost=8.96..14.29 rows=58 width=24) (actual time=0.090..0.119 rows=58 loops=1)

Total rows: 33 of 33 Query complete 00:00:00.046

	QUERY PLAN	
	text	
1	HashAggregate (cost=24.98..25.17 rows=19 width=4) (actual time=0.267..0.269 rows=16 loops=1)	
2	Group Key: cd.client_id	
3	Batches: 1 Memory Usage: 24kB	
4	CTE calldurations	
5	-> Hash Join (cost=14.21..20.14 rows=58 width=40) (actual time=0.130..0.187 rows=58 loops=1)	
6	Hash Cond: (ct.client_id = cl.client_id)	
7	-> Hash Join (cost=8.96..14.29 rows=58 width=24) (actual time=0.090..0.119 rows=58 loops=1)	
8	Hash Cond: (ct.contract_id = pt.contract_id)	
9	-> Seq Scan on contract ct (cost=0.00..4.00 rows=200 width=8) (actual time=0.004..0.011 rows=200 loops=1)	
10	-> Hash (cost=8.23..8.23 rows=58 width=24) (actual time=0.080..0.080 rows=58 loops=1)	
11	Buckets: 1024 Batches: 1 Memory Usage: 12kB	
12	-> Hash Join (cost=6.50..8.23 rows=58 width=24) (actual time=0.057..0.071 rows=58 loops=1)	
13	Hash Cond: (ic.phone_number = pt.phone_on_tariff_number)	
14	-> Seq Scan on international_call ic (cost=0.00..1.58 rows=58 width=28) (actual time=0.005..0.010 rows=58 loops=1)	
15	-> Hash (cost=4.00..4.00 rows=200 width=12) (actual time=0.047..0.047 rows=200 loops=1)	
16	Buckets: 1024 Batches: 1 Memory Usage: 18kB	
17	-> Seq Scan on phone_on_tariff pt (cost=0.00..4.00 rows=200 width=12) (actual time=0.005..0.026 rows=200 loop...	
18	-> Hash (cost=4.00..4.00 rows=100 width=4) (actual time=0.030..0.031 rows=100 loops=1)	
19	Buckets: 1024 Batches: 1 Memory Usage: 12kB	
20	-> Seq Scan on client cl (cost=0.00..4.00 rows=100 width=4) (actual time=0.010..0.019 rows=100 loops=1)	
21	-> Hash Join (cost=3.48..4.79 rows=19 width=4) (actual time=0.246..0.264 rows=20 loops=1)	
22	Hash Cond: (cd.call_zone_id = calldurations.call_zone_id)	

23	Join Filter: (cd.call_duration_minutes > (avg(calldurations.call_duration_minutes)))
24	Rows Removed by Join Filter: 38
25	-> CTE Scan on calldurations cd (cost=0.00..1.16 rows=58 width=40) (actual time=0.131..0.135 rows=58 loops=1)
26	-> Hash (cost=2.75..2.75 rows=58 width=36) (actual time=0.102..0.103 rows=18 loops=1)
27	Buckets: 1024 Batches: 1 Memory Usage: 9kB
28	-> HashAggregate (cost=1.45..2.17 rows=58 width=36) (actual time=0.091..0.099 rows=18 loops=1)
29	Group Key: calldurations.call_zone_id
30	Batches: 1 Memory Usage: 32kB
31	-> CTE Scan on calldurations (cost=0.00..1.16 rows=58 width=36) (actual time=0.000..0.071 rows=58 loops=1)
32	Planning Time: 0.517 ms
33	Execution Time: 0.345 ms

(Добавила простые индексы)

```

phone_provider/postgres@PostgreSQL 14
[Icons] [No limit] [Icons] [Icons] [Icons] [Icons] [Icons] [Icons] [Icons] [Icons] [Icons]
Query Query History
1  -- Индекс на client_id в таблице client
2  CREATE INDEX idx_client_id ON client(client_id);
3
4  -- Индекс на contract_id в таблице contract
5  CREATE INDEX idx_contract_id ON contract(contract_id);
6
7  -- Индекс на phone_number в таблице phone_on_tariff
8  CREATE INDEX idx_phone_on_tariff_number ON phone_on_tariff(phone_on_tariff_number);
9
10 -- Индекс на call_zone_id в таблице international_call
11 CREATE INDEX idx_international_call_zone_id ON international_call(call_zone_id);
12
13 -- Индекс на international_call_start_time и international_call_end_time для фильтрации
14 CREATE INDEX idx_international_call_times ON international_call(international_call_start_time, international_call_end_time);
15

Data Output Messages Notifications
CREATE INDEX
Query returned successfully in 42 msec.

```

	QUERY PLAN text	
10	-> Hash (cost=8.23..8.23 rows=58 width=24) (actual time=0.092..0.093 rows=58 loops=1)	
11	Buckets: 1024 Batches: 1 Memory Usage: 12kB	
12	-> Hash Join (cost=6.50..8.23 rows=58 width=24) (actual time=0.068..0.083 rows=58 loops=1)	
13	Hash Cond: (ic.phone_number = pt.phone_on_tariff_number)	
14	-> Seq Scan on international_call ic (cost=0.00..1.58 rows=58 width=28) (actual time=0.008..0.014 rows=58 loops=1)	
15	-> Hash (cost=4.00..4.00 rows=200 width=12) (actual time=0.053..0.053 rows=200 loops=1)	
16	Buckets: 1024 Batches: 1 Memory Usage: 18kB	
17	-> Seq Scan on phone_on_tariff pt (cost=0.00..4.00 rows=200 width=12) (actual time=0.007..0.031 rows=200 loop...)	
18	-> Hash (cost=4.00..4.00 rows=100 width=4) (actual time=0.038..0.039 rows=100 loops=1)	
19	Buckets: 1024 Batches: 1 Memory Usage: 12kB	
20	-> Seq Scan on client cl (cost=0.00..4.00 rows=100 width=4) (actual time=0.016..0.027 rows=100 loops=1)	
21	-> Hash Join (cost=3.48..4.79 rows=19 width=4) (actual time=0.275..0.293 rows=20 loops=1)	
22	Hash Cond: (cd.call_zone_id = calldurations.call_zone_id)	
23	Join Filter: (cd.call_duration_minutes > (avg(calldurations.call_duration_minutes)))	
24	Rows Removed by Join Filter: 38	
25	-> CTE Scan on calldurations cd (cost=0.00..1.16 rows=58 width=40) (actual time=0.156..0.161 rows=58 loops=1)	
26	-> Hash (cost=2.75..2.75 rows=58 width=36) (actual time=0.106..0.106 rows=18 loops=1)	
27	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
28	-> HashAggregate (cost=1.45..2.17 rows=58 width=36) (actual time=0.093..0.101 rows=18 loops=1)	
29	Group Key: calldurations.call_zone_id	
30	Batches: 1 Memory Usage: 32kB	
31	-> CTE Scan on calldurations (cost=0.00..1.16 rows=58 width=36) (actual time=0.002..0.070 rows=58 loops=1)	
32	Planning Time: 2.411 ms	
33	Execution Time: 0.386 ms	

(Добавила составные индексы)

```
-- Составной индекс на contract_id и client_id в таблице contract
CREATE INDEX idx_contract_client ON contract(client_id, contract_id);

-- Составной индекс на phone_number и contract_id в таблице phone_on_tariff
CREATE INDEX idx_phone_on_tariff_contract ON phone_on_tariff(contract_id,
phone_on_tariff_number);

-- Составной индекс на phone_number и call_zone_id в таблице international_call для
ускорения JOIN
CREATE INDEX idx_international_call_phone_zone ON international_call(phone_number,
call_zone_id);

-- Составной индекс для таблицы international_call, который может помочь в фильтрации по
датам и агрегации
CREATE INDEX idx_international_call_date_zone ON international_call(call_zone_id,
international_call_start_time, international_call_end_time);
```

QueryQuery History

```

1  -- Составной индекс на contract_id и client_id в таблице contract
2  CREATE INDEX idx_contract_client ON contract(client_id, contract_id);
3
4  -- Составной индекс на phone_number и contract_id в таблице phone_on_tariff
5  CREATE INDEX idx_phone_on_tariff_contract ON phone_on_tariff(contract_id, phone_on_tariff_number);
6
7  -- Составной индекс на phone_number и call_zone_id в таблице international_call для ускорения JOIN
8  CREATE INDEX idx_international_call_phone_zone ON international_call(phone_number, call_zone_id);
9
10 -- Составной индекс для таблицы international_call, который может помочь в фильтрации по датам и агрегации
11 CREATE INDEX idx_international_call_date_zone ON international_call(call_zone_id, international_call_start_time, international_call_end_time);
12

```

Data OutputMessagesNotifications

CREATE INDEX

Query returned successfully in 66 msec.

	<div> <div>QUERY PLAN</div> <div>text</div> <div></div> </div>
1	HashAggregate (cost=24.98..25.17 rows=19 width=4) (actual time=0.267..0.270 rows=16 loops=1)
2	Group Key: cd.client_id
3	Batches: 1 Memory Usage: 24kB
4	CTE calldurations
5	-> Hash Join (cost=14.21..20.14 rows=58 width=40) (actual time=0.133..0.190 rows=58 loops=1)
6	Hash Cond: (ct.client_id = cl.client_id)
7	-> Hash Join (cost=8.96..14.29 rows=58 width=24) (actual time=0.092..0.122 rows=58 loops=1)
8	Hash Cond: (ct.contract_id = pt.contract_id)
9	-> Seq Scan on contract ct (cost=0.00..4.00 rows=200 width=8) (actual time=0.006..0.015 rows=200 loops=1)
10	-> Hash (cost=8.23..8.23 rows=58 width=24) (actual time=0.079..0.080 rows=58 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 12kB
12	-> Hash Join (cost=6.50..8.23 rows=58 width=24) (actual time=0.057..0.071 rows=58 loops=1)
13	Hash Cond: (ic.phone_number = pt.phone_on_tariff_number)
14	-> Seq Scan on international_call ic (cost=0.00..1.58 rows=58 width=28) (actual time=0.004..0.009 rows=58 loops=1)
15	-> Hash (cost=4.00..4.00 rows=200 width=12) (actual time=0.047..0.047 rows=200 loops=1)
16	Buckets: 1024 Batches: 1 Memory Usage: 18kB
17	-> Seq Scan on phone_on_tariff pt (cost=0.00..4.00 rows=200 width=12) (actual time=0.005..0.025 rows=200 loop...
18	-> Hash (cost=4.00..4.00 rows=100 width=4) (actual time=0.031..0.031 rows=100 loops=1)
19	Buckets: 1024 Batches: 1 Memory Usage: 12kB
20	-> Seq Scan on client cl (cost=0.00..4.00 rows=100 width=4) (actual time=0.010..0.019 rows=100 loops=1)
21	-> Hash Join (cost=3.48..4.79 rows=19 width=4) (actual time=0.246..0.264 rows=20 loops=1)
22	Hash Cond: (cd.call_zone_id = calldurations.call_zone_id)

23	Join Filter: (cd.call_duration_minutes > (avg(calldurations.call_duration_minutes)))
24	Rows Removed by Join Filter: 38
25	-> CTE Scan on calldurations cd (cost=0.00..1.16 rows=58 width=40) (actual time=0.134..0.139 rows=58 loops=1)
26	-> Hash (cost=2.75..2.75 rows=58 width=36) (actual time=0.104..0.104 rows=18 loops=1)
27	Buckets: 1024 Batches: 1 Memory Usage: 9kB
28	-> HashAggregate (cost=1.45..2.17 rows=58 width=36) (actual time=0.093..0.101 rows=18 loops=1)
29	Group Key: calldurations.call_zone_id
30	Batches: 1 Memory Usage: 32kB
31	-> CTE Scan on calldurations (cost=0.00..1.16 rows=58 width=36) (actual time=0.001..0.069 rows=58 loops=1)
32	Planning Time: 1.455 ms
33	Execution Time: 0.356 ms

Вывод: в данном SELECT-запросе использование простых индексов замедляет выполнение, так как он включает в себя много комплексных операций с данными, и запрос не оптимизируется, а только замедляется, тратя время на чтение этих индексов, а в случае со сложными индексами, написанными специально под этот запрос, всё наоборот – операция ускоряется, JOINы оптимизируются.

Создам простые индексы для другого запроса

Снова EXPLAIN ANALYZE до индексов:

	QUERY PLAN
	text
1	HashAggregate (cost=14.62..15.61 rows=99 width=4) (actual time=0.149..0.156 rows=97 loops=1)
2	Group Key: client.client_id
3	Batches: 1 Memory Usage: 24kB
4	-> Hash Anti Join (cost=9.12..14.37 rows=99 width=4) (actual time=0.116..0.134 rows=97 loops=1)
5	Hash Cond: (client.client_id = contract.client_id)
6	-> Seq Scan on client (cost=0.00..4.00 rows=100 width=4) (actual time=0.010..0.017 rows=100 loops=1)
7	-> Hash (cost=9.11..9.11 rows=1 width=4) (actual time=0.101..0.102 rows=3 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Nested Loop (cost=4.25..9.11 rows=1 width=4) (actual time=0.083..0.098 rows=3 loops=1)
10	-> Hash Join (cost=4.10..8.86 rows=1 width=4) (actual time=0.075..0.087 rows=3 loops=1)
11	Hash Cond: (phone_on_tariff.phone_on_tariff_number = payment.phone_number)
12	-> Seq Scan on phone_on_tariff (cost=0.00..4.00 rows=200 width=12) (actual time=0.008..0.030 rows=200 loops=1)
13	-> Hash (cost=4.09..4.09 rows=1 width=8) (actual time=0.034..0.034 rows=3 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB
15	-> Seq Scan on payment (cost=0.00..4.09 rows=1 width=8) (actual time=0.028..0.029 rows=3 loops=1)
16	Filter: ((payment_date >= date_trunc('month',text, (CURRENT_DATE - '1 mon'::interval month))) AND (payment_date < date_trunc('month',text, (CURRENT_DATE)::timestamp with time ...
17	Rows Removed by Filter: 100
18	-> Index Scan using contract_pkey on contract (cost=0.14..0.24 rows=1 width=8) (actual time=0.003..0.003 rows=1 loops=3)
19	Index Cond: (contract_id = phone_on_tariff.contract_id)
20	Planning Time: 0.683 ms
21	Execution Time: 0.214 ms

```
-- Индекс на client_id в таблице contract для улучшения связывания
CREATE INDEX idx_contract_client_id ON contract(client_id);
```

```
-- Индекс на contract_id в таблице phone_on_tariff для улучшения связывания
CREATE INDEX idx_phone_on_tariff_contract_id ON phone_on_tariff(contract_id);

-- Индекс на phone_number в таблице payment для улучшения связывания
CREATE INDEX idx_payment_phone_number ON payment(phone_number);

-- Индекс на payment_date в таблице payment для улучшения фильтрации по дате
CREATE INDEX idx_payment_payment_date ON payment(payment_date);
```

The screenshot shows a PostgreSQL client window titled "phone_provider/postgres@PostgreSQL 14". The interface includes a toolbar with icons for file operations, query execution, and settings. The "Query" tab is active, displaying a SQL script with 12 lines. The script contains four comments in Russian and four "CREATE INDEX" statements. The "Messages" tab is also visible, showing the output "CREATE INDEX" and a confirmation message: "Query returned successfully in 67 msec.".

```
1  -- Индекс на client_id в таблице contract для улучшения связывания
2  CREATE INDEX idx_contract_client_id ON contract(client_id);
3
4  -- Индекс на contract_id в таблице phone_on_tariff для улучшения связывания
5  CREATE INDEX idx_phone_on_tariff_contract_id ON phone_on_tariff(contract_id);
6
7  -- Индекс на phone_number в таблице payment для улучшения связывания
8  CREATE INDEX idx_payment_phone_number ON payment(phone_number);
9
10 -- Индекс на payment_date в таблице payment для улучшения фильтрации по дате
11 CREATE INDEX idx_payment_payment_date ON payment(payment_date);
12
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 67 msec.

```
EXPLAIN ANALYZE
SELECT client.client_id
FROM client
WHERE NOT EXISTS (
    SELECT 1
    FROM contract
    JOIN phone_on_tariff ON contract.contract_id = phone_on_tariff.contract_id
    JOIN payment ON phone_on_tariff.phone_on_tariff_number = payment.phone_number
    WHERE client.client_id = contract.client_id
    AND payment.payment_date >= date_trunc('month', current_date - interval '1' month)
```



```

    AND payment.payment_date < date_trunc('month', current_date)
)
GROUP BY client.client_id;

```

	QUERY PLAN text	
1	HashAggregate (cost=14.62..15.61 rows=99 width=4) (actual time=0.218..0.229 rows=97 loops=1)	
2	Group Key: client.client_id	
3	Batches: 1 Memory Usage: 24kB	
4	-> Hash Anti Join (cost=9.12..14.37 rows=99 width=4) (actual time=0.162..0.194 rows=97 loops=1)	
5	Hash Cond: (client.client_id = contract.client_id)	
6	-> Seq Scan on client (cost=0.00..4.00 rows=100 width=4) (actual time=0.023..0.040 rows=100 loops=1)	
7	-> Hash (cost=9.11..9.11 rows=1 width=4) (actual time=0.130..0.131 rows=3 loops=1)	
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
9	-> Nested Loop (cost=4.25..9.11 rows=1 width=4) (actual time=0.107..0.128 rows=3 loops=1)	
10	-> Hash Join (cost=4.10..8.86 rows=1 width=4) (actual time=0.098..0.115 rows=3 loops=1)	
11	Hash Cond: (phone_on_tariff.phone_on_tariff_number = payment.phone_number)	
12	-> Seq Scan on phone_on_tariff (cost=0.00..4.00 rows=200 width=12) (actual time=0.009..0.039 rows=200 loops=1)	
13	-> Hash (cost=4.09..4.09 rows=1 width=8) (actual time=0.051..0.051 rows=3 loops=1)	
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
15	-> Seq Scan on payment (cost=0.00..4.09 rows=1 width=8) (actual time=0.045..0.046 rows=3 loops=1)	
16	Filter: ((payment_date >= date_trunc('month'::text, (CURRENT_DATE - '1 mon'::interval month))) AND (payment_date < date_trunc('month'::text, (CURRENT_DATE)::timestamp with time ...	
17	Rows Removed by Filter: 100	
18	-> Index Scan using contract_pkey on contract (cost=0.14..0.24 rows=1 width=8) (actual time=0.004..0.004 rows=1 loops=3)	
19	Index Cond: (contract_id = phone_on_tariff.contract_id)	
20	Planning Time: 17.479 ms	
21	Execution Time: 0.420 ms	

(Создала составной индекс для данного запроса)

```

-- Составной индекс для фильтрации платежей во временном промежутке
CREATE INDEX idx_payment_date_phone_number ON payment(payment_date, phone_number);

```

Query

Query History

```

1  -- Составной индекс для фильтрации платежей во временном промежутке
2  CREATE INDEX idx_payment_date_phone_number ON payment(payment_date, phone_number);
3

```

Data Output

Messages

Notifications

CREATE INDEX

Query returned successfully in 39 msec.

QUERY PLAN		
text		
1	HashAggregate (cost=14.62..15.61 rows=99 width=4) (actual time=0.136..0.144 rows=97 loops=1)	
2	Group Key: client.client_id	
3	Batches: 1 Memory Usage: 24kB	
4	-> Hash Anti Join (cost=9.12..14.37 rows=99 width=4) (actual time=0.102..0.121 rows=97 loops=1)	
5	Hash Cond: (client.client_id = contract.client_id)	
6	-> Seq Scan on client (cost=0.00..4.00 rows=100 width=4) (actual time=0.012..0.020 rows=100 loops=1)	
7	-> Hash (cost=9.11..9.11 rows=1 width=4) (actual time=0.085..0.085 rows=3 loops=1)	
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
9	-> Nested Loop (cost=4.25..9.11 rows=1 width=4) (actual time=0.068..0.083 rows=3 loops=1)	
10	-> Hash Join (cost=4.10..8.86 rows=1 width=4) (actual time=0.063..0.075 rows=3 loops=1)	
11	Hash Cond: (phone_on_tariff.phone_on_tariff_number = payment.phone_number)	
12	-> Seq Scan on phone_on_tariff (cost=0.00..4.00 rows=200 width=12) (actual time=0.007..0.028 rows=200 loops=1)	
13	-> Hash (cost=4.09..4.09 rows=1 width=8) (actual time=0.031..0.031 rows=3 loops=1)	
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
15	-> Seq Scan on payment (cost=0.00..4.09 rows=1 width=8) (actual time=0.027..0.028 rows=3 loops=1)	
16	Filter: ((payment_date >= date_trunc('month':text, (CURRENT_DATE - '1 mon':interval month))) AND (payment_date < date_trunc('month':text, (CURRENT_DATE)::timestamp with time ...	
17	Rows Removed by Filter: 100	
18	-> Index Scan using contract_pkey on contract (cost=0.14..0.24 rows=1 width=8) (actual time=0.002..0.002 rows=1 loops=3)	
19	Index Cond: (contract_id = phone_on_tariff.contract_id)	
20	Planning Time: 0.738 ms	
21	Execution Time: 0.193 ms	

С помощью DROP INDEX удаляла индексы

Вывод: этот запрос слишком прост для индексов, поэтому скорость: без индексов > сложные индексы > простые индексы

Вывод по работе: были изучены практические навыки создания представлений и запросов на выборку данных к базе данных PostgreSQL, использования подзапросов при модификации данных и индексов.

```
SELECT
  cz.city,
  COUNT(ic.*) AS total_calls
FROM
  international_call ic
JOIN
  call_zone cz ON ic.call_zone_id = cz.call_zone_id
WHERE
  cz.city IN ('Moscow', 'London', 'Paris')
GROUP BY
  cz.city;
```

The screenshot shows a PostgreSQL query editor interface. The top section is titled "Query" and "Query History". The query text is as follows:

```
1 SELECT
2   cz.city,
3   COUNT(ic.*) AS total_calls
4 FROM
5   international_call ic
6 JOIN
7   call_zone cz ON ic.call_zone_id = cz.call_zone_id
8 WHERE
9   cz.city IN ('Moscow', 'London', 'Paris')
10 GROUP BY
11   cz.city;
12 |
```

Below the query editor, there is a section titled "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table with the following data:

	city character varying (60)	total_calls bigint
1	London	2
2	Moscow	5
3	Paris	3

At the bottom of the interface, it says "Total rows: 3 of 3" and "Query complete 00:00:00.467".

```
CREATE OR REPLACE VIEW most_popular_call_zone_yearly AS
SELECT
  cz.call_zone_id,
  cz.city,
  cz.country,
  COUNT(*) AS call_count
FROM
  international_call ic
JOIN call_zone cz ON ic.call_zone_id = cz.call_zone_id
```

```

WHERE
    ic.international_call_start_time BETWEEN date_trunc('year', CURRENT_DATE - INTERVAL '1
year') AND (date_trunc('year', CURRENT_DATE) - INTERVAL '1 second')
GROUP BY
    cz.call_zone_id, cz.city, cz.country
HAVING
    COUNT(*) = (
        SELECT MAX(call_count) FROM (
            SELECT
                COUNT(*) AS call_count
            FROM
                international_call ic2
            JOIN call_zone cz2 ON ic2.call_zone_id = cz2.call_zone_id
            WHERE
                ic2.international_call_start_time >= date_trunc('year', CURRENT_DATE - INTERVAL '1
year') AND
                ic2.international_call_start_time < date_trunc('year', CURRENT_DATE)
            GROUP BY
                cz2.call_zone_id
        ) AS max_subquery
    );

```

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is to 'phone_provider/postgres@PostgreSQL 14'. Below the toolbar, the 'Query' tab is active, displaying a SQL query that creates or replaces a view named 'most_popular_call_zone_yearly'. The query uses a complex subquery to find the maximum call count for each call zone, city, and country combination over the last year. The bottom panel shows the 'Messages' tab with the message 'Query returned successfully in 95 msec.'

```

1 CREATE OR REPLACE VIEW most_popular_call_zone_yearly AS
2 SELECT
3     cz.call_zone_id,
4     cz.city,
5     cz.country,
6     COUNT(*) AS call_count
7 FROM
8     international_call ic
9 JOIN call_zone cz ON ic.call_zone_id = cz.call_zone_id
10 WHERE
11     ic.international_call_start_time BETWEEN date_trunc('year', CURRENT_DATE - INTERVAL '1 year') AND (date_trunc('year', CURRENT_DATE) - 1
12 GROUP BY
13     cz.call_zone_id, cz.city, cz.country
14 HAVING
15     COUNT(*) = (
16         SELECT MAX(call_count) FROM (
17             SELECT
18                 COUNT(*) AS call_count
19             FROM
20                 international_call ic2
21             JOIN call_zone cz2 ON ic2.call_zone_id = cz2.call_zone_id
22             WHERE
23                 ic2.international_call_start_time >= date_trunc('year', CURRENT_DATE - INTERVAL '1 year') AND

```

CREATE VIEW

Query returned successfully in 95 msec.

phone_provider/postgres@PostgreSQL 14

Query Query History

```
1 SELECT * FROM most_popular_call_zone_yearly;
```

Data Output Messages Notifications

	call_zone_id integer	city character varying (60)	country character varying (60)	call_count bigint
1	10	Stacyport	Poland	4

Интересный факт 😊

Query Query History

```
1 SELECT MAX(EXTRACT(EPOCH FROM (international_call_end_time - international_call_start_time)) / (60 * 60 * 24 * 365)) AS max_call_duration
2 FROM international_call;
3
```

Data Output Messages Notifications

	max_call_duration_years numeric
1	19.6432277714358194