

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО»**

Факультет инфокоммуникационных технологий

Дисциплина:

«Проектирование и реализация баз данных»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
«ЗАПРОСЫ НА ВЫБОРКУ И МОДИФИКАЦИЮ ДАННЫХ,
ПРЕДСТАВЛЕНИЯ И ИНДЕКСЫ В POSTGRESQL»**

Выполнил:

студент группы К32392

Стукалов Артем Сергеевич

(подпись)

Проверил(а):

Говорова Марина Михайловна

(отметка о выполнении)

(подпись)

Санкт-Петербург
2023 г.

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 посмотреть 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.

Индивидуальное практическое задание:

База данных “Ресторан”

Запросы на выборку:

1. Вывести данные официанта, принявшего заказы на максимальную сумму за истекший месяц.
2. Рассчитать премию каждого официанта за последние 10 дней (5% от стоимости каждого заказа).
3. Подсчитать, сколько ингредиентов содержит каждое блюдо.
4. Вывести название блюда, содержащее максимальное число ингредиентов.
5. Какой повар может приготовить максимальное число видов блюд?
6. Сколько закреплено столов за каждым из официантов?
7. Какой из ингредиентов используется в максимальном количестве блюд?

Представления:

1. для расчета стоимости ингредиентов для заданного блюда;
2. для всех поваров количество приготовленных блюд по каждому блюду за определенную дату.

Схема базы данных:

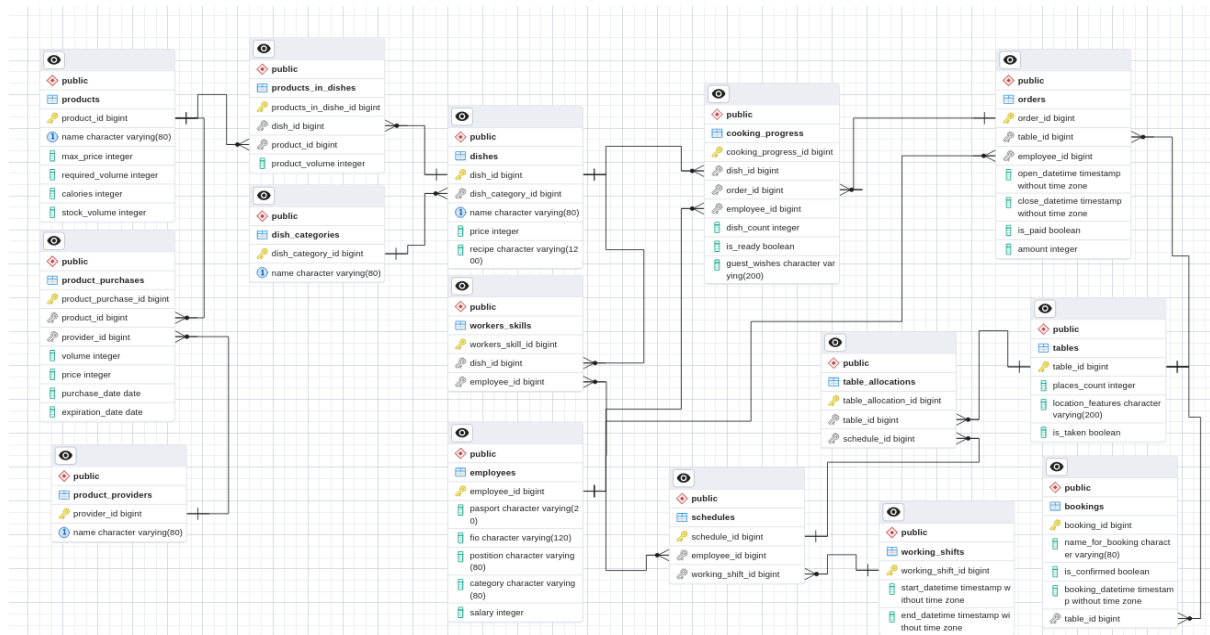


Рис. 1 - Схема базы данных
Выполнение

Запросы на выборку:

1. Вывести данные официанта, принявшего заказы на максимальную сумму за истекший месяц.

```
SELECT
    e.employee_id,
    e.pasport,
    e.fio,
    e.postition,
    e.category,
    e.salary,
    SUM(o.amount),
FROM
    employees e
    LEFT JOIN orders o USING(employee_id)
WHERE
    e.postition = 'Официант'
    AND o.open_datetime >= '2024-03-01'
    AND o.open_datetime <= '2024-03-30'
GROUP BY
    e.employee_id
HAVING
    SUM(o.amount) = (
        SELECT
            SUM(amount) as a_sum
        FROM
            orders
```

```

WHERE
    open_datetime >= '2024-03-01'
    AND open_datetime <= '2024-03-30'
GROUP BY
    employee_id
ORDER BY
    a_sum DESC
LIMIT
    1
);

```

Рис. 2 - SELECT №1

	employee_id [PK] bigint	passport character varying (20)	fio character varying (120)	postition character varying (80)	category character varying (80)	salary integer	sum bigint
1	6	2234 123460	Иван Инванович Иванов	Официант	Обслуживание	50000	158386

Рис. 3 - Результат SELECT №1

2. Рассчитать премию каждого официанта за последние 10 дней (5% от стоимости каждого заказа).

```

SELECT
    e.employee_id,
    SUM(o.amount) * 1.05 as orders_sum
FROM
    employees e
LEFT JOIN orders o USING(employee_id)
WHERE
    e.postition = 'Официант'
    AND o.open_datetime >= now() - interval '10 day'
GROUP BY
    e.employee_id;

```

Рис. 4 - SELECT №2

	employee_id [PK] bigint	orders_sum numeric
1	6	3405126.90
2	5	3382324.05
3	7	3370901.10

Рис. 5 - Результат SELECT №2

3. Подсчитать, сколько ингредиентов содержит каждое блюдо.

```

SELECT

```

```

d.dish_id,
d.name,
COUNT(pd.product_id)
FROM
dishes d
LEFT JOIN products_in_dishes pd USING(dish_id)
GROUP BY
dish_id;

```

Рис. 6 - SELECT №3

	dish_id [PK] bigint	name character varying (80)	count bigint
1	55	dish_54	968
2	27	dish_26	1000
3	23	dish_22	984
4	56	dish_55	1028
5	91	dish_90	937
6	58	dish_57	1008
7	8	dish_7	949
8	87	dish_86	1023
9	74	dish_73	1107
10	29	dish_28	953
11	54	dish_53	952
12	71	dish_70	1007
13	4	dish_3	1071
14	68	dish_67	983
15	34	dish_33	1040
16	51	dish_50	988
17	96	dish_95	962

Рис. 7 - Результат SELECT №3

4. Вывести название блюда, содержащее максимальное число ингредиентов.

```

SELECT
dish_id,
name
FROM
dishes d
LEFT JOIN products_in_dishes pd USING(dish_id)
GROUP BY
d.dish_id
HAVING
COUNT(*) = (
SELECT
COUNT(*) p_count
FROM
products_in_dishes
GROUP BY
dish_id

```

```
ORDER BY
    p_count DESC
LIMIT
    1
);
```

Рис. 8 - SELECT №4

	dish_id [PK] bigint	name character varying (80)
1	74	dish_73

Рис. 9 - Результат SELECT №4

5. Какой повар может приготовить максимальное число видов блюд?

```
SELECT
    e.employee_id,
    e.fio
FROM
    employees e
LEFT JOIN workers_skills ws USING(employee_id)
GROUP BY
    e.employee_id
HAVING
    COUNT(*) = (
        SELECT
            COUNT(*) dish_count
        FROM
            workers_skills
        GROUP BY
            employee_id
        ORDER BY
            dish_count DESC
        LIMIT
            1
    );
```

Рис. 10 - SELECT №5

	employee_id [PK] bigint	fio character varying (120)
1	3	Егор Лавров Ефимович

Рис. 11 - Результат SELECT №5

6. Сколько закреплено столов за каждым из официантов?

```

SELECT
    e.employee_id,
    e.fio,
    COUNT(ta.table_id)
FROM
    employees e
    LEFT JOIN schedules s USING(employee_id)
    LEFT JOIN table_allocations ta USING(schedule_id)
WHERE
    s.working_shift_id = 2
    AND e.postition = 'Официант'
GROUP BY
    e.employee_id;

```

Рис. 12 - SELECT №6

	employee_id [PK] bigint	fio character varying (120)	count bigint
1	5	Влад Анисимов Юрьевич	1
2	6	Иван Инванович Иванов	2
3	7	Петр Анисимов Юрьевич	2

Рис. 13 - Результат SELECT №6

7. Какой из ингредиентов используется в максимальном количестве блюд?

```

SELECT
    p.product_id,
    p.name
FROM
    products p
    LEFT JOIN products_in_dishes pd USING(product_id)
GROUP BY
    p.product_id
HAVING
    COUNT(pd.dish_id) = (
        SELECT
            COUNT(pd.dish_id) dish_count
        FROM
            products p
            LEFT JOIN products_in_dishes pd USING(product_id)
        GROUP BY
            p.product_id
        ORDER BY
            dish_count DESC
        LIMIT

```

```
1  
) ;
```

Рис. 14 - SELECT №7

	product_id [PK] bigint	name character varying (80)
1	355	product_354
2	378	product_377
3	409	product_408
4	293	product_292
5	796	product_795
6	681	product_680
7	514	product_513
8	60	product_59
9	975	product_974
10	414	product_413
11	302	product_301
12	316	product_315
13	877	product_876
14	798	product_797
15	79	product_78
16	362	product_361

Рис. 15 - Результат SELECT №7

Представления:

1. Для расчета стоимости ингредиентов для заданного блюда.

```
CREATE VIEW products_cost_for_dish AS  
SELECT  
    pd.dish_id,  
    SUM(p.max_price * pd.product_volume * 0.01)  
FROM  
    products_in_dishes pd  
    LEFT JOIN products p USING (product_id)  
WHERE  
    pd.dish_id = 1  
GROUP BY  
    pd.dish_id;
```

Рис. 16 - VIEW №1

2. Для всех поваров количество приготовленных блюд по каждому блюду за определенную дату.

```
CREATE VIEW cooked_dished_count AS  
SELECT  
    cp.employee_id,  
    cp.dish_id,  
    SUM(cp.dish_count) dish_count  
FROM
```



```

    cooking_progress cp
  LEFT JOIN orders o USING(order_id)
WHERE
  cp.is_ready = TRUE
  AND DATE(o.open_datetime) = '2024-03-13'
GROUP BY
  cp.employee_id,
  cp.dish_id;

```

Рис. 17 - VIEW №1

INSERT:

Отдает приготовление блюда в самом позднем заказе повару, который на данный момент готовит меньше всех блюд.

```

INSERT INTO
  cooking_progress (
    dish_id,
    order_id,
    employee_id,
    dish_count,
    is_ready,
    guest_wishes
  )
VALUES
  (
    4,
    10,
    (
      SELECT
        employee_id
      FROM
        (
          SELECT
            employee_id,
            SUM(dish_count) dishes_count
          FROM
            cooking_progress
          WHERE
            is_ready = FALSE
            AND employee_id IN(
              SELECT
                employee_id
              FROM
                workers_skills
            )
        )
    )
  )

```

```

        WHERE
            dish_id = 1
        )
    GROUP BY
        employee_id
    ORDER BY
        dishes_count ASC
    LIMIT
        1
    ) available_employees
), 3, FALSE, ''
);

```

Рис. 18 - INSERT

	cooking_progress_id [PK] bigint	dish_id bigint	order_id bigint	employee_id bigint	dish_count integer	is_ready boolean	guest_wishes character varying (200)
1	1	1	1	1	2	true	
2	2	2	1	1	1	true	
3	3	3	1	2	5	true	
4	4	3	2	2	2	true	
5	5	4	1	1	3	true	
6	6	1	5	1	2	false	
7	7	4	5	2	5	false	
8	8	2	5	3	3	false	
9	9	3	5	1	1	false	
10	11	4	10	3	3	false	
11	13	4	11	3	3	false	
12	14	4	12	3	3	false	
13	15	2	12	3	3	false	
14	16	4	10	1	3	false	
15	17	4	15	2	3	false	
16	18	4	16	1	3	false	

Рис. 19 - Результат INSERT

UPDATE:

Увеличивает необходимый объем продуктов на N, если продукт используется более чем в K блюдах.

```

UPDATE
    products
SET
    required_volume = required_volume + 100
WHERE
    product_id IN (
        SELECT
            product_id
        FROM

```

```

products_in_dishes
GROUP BY
    product_id
HAVING
    COUNT(dish_id) > 1
);

```

Рис. 20 - UPDATE

	product_id [PK] bigint	name character varying (80)	max_price integer	required_volume integer	calories integer	stock_volume integer
1	18	product_17	41	1229	100	1500
2	46	product_45	13	1046	100	1472
3	78	product_77	37	1272	100	1453
4	176	product_175	89	706	100	1298
5	223	product_222	37	1051	100	1137
6	298	product_297	94	1286	100	1375
7	314	product_313	56	632	100	1370
8	106	product_105	83	1184	100	1403
9	382	product_381	20	2228	100	1314
10	383	product_382	48	1791	100	1468
11	384	product_383	85	2378	100	1300
12	385	product_384	66	1665	100	1202
13	386	product_385	16	1585	100	1481
14	387	product_386	64	2203	100	1242
15	388	product_387	92	1552	100	1409
16	389	product_388	85	2151	100	1252
17	390	product_389	26	1582	100	1361
18	391	product_390	44	1564	100	1372

Рис. 21 - Результат UPDATE

DELETE:

Удаляет всех категории блюд, которые не используются ни в одном блюде.

```

DELETE FROM
    dish_categories
WHERE
    dish_category_id NOT IN (
        SELECT
            DISTINCT dish_category_id
        FROM
            dishes
    );

```

Рис. 22 - DELETE

Простой индекс:

```
SELECT
    COUNT(*)
FROM
    orders
WHERE
    DATE(o.open_datetime) >= '2025-01-01'
    AND DATE(o.open_datetime) <= '2025-12-31';
CREATE INDEX o_index ON orders (open_datetime);
```

Рис. 23 - простой INDEX

Проведя серию замеров из 30 измерений, получилось сократить среднее время выполнения с 156мс до 147мс.

Составной индекс:

```
SELECT
    *
FROM
    (
        SELECT
            o.order_id as order_id,
            e.employee_id,
            e.fio,
            e.postition
        FROM
            orders o
            LEFT JOIN employees e USING(employee_id)
        WHERE
            DATE(o.open_datetime) >= '2023-01-01'
            AND DATE(o.open_datetime) <= '2023-12-31'
        ) as orders_with_emp
    LEFT JOIN (
        SELECT
            order_id,
            SUM(dish_price * dish_count) as calc_order_price
        FROM
            cooking_progress cp
            LEFT JOIN (
                SELECT
                    pd.dish_id as dish_id,
                    SUM(p.max_price * pd.product_volume * 0.01) as dish_price
                FROM
                    products_in_dishes pd
                    LEFT JOIN products p USING (product_id)
```

```

        GROUP BY
            pd.dish_id
    ) as dish_costs USING(dish_id)
GROUP BY
    order_id
) as orders_cost USING (order_id);

CREATE INDEX o_index ON orders (open_datetime, employee_id);
CREATE INDEX cp_index ON cooking_progress (dish_id, order_id);

```

Рис. 24 - составной INDEX


	QUERY PLAN	
	text	
1	Nested Loop Left Join (cost=620724.98..42277955.54 rows=10000 width=484)	
2	-> Nested Loop Left Join (cost=620724.83..42277706.23 rows=10000 width=48)	
3	Join Filter: (o.order_id = orders_cost.order_id)	
4	-> Index Scan using orders_pkey on orders o (cost=0.43..90639.43 rows=10000 width=16)	
5	Filter: ((date(open_datetime) >= '2025-01-01'::date) AND (date(open_datetime) <= '2025-12-31'::date))	
6	-> Materialize (cost=620724.40..639295.13 rows=182131 width=40)	
7	-> Subquery Scan on orders_cost (cost=620724.40..636961.47 rows=182131 width=40)	
8	-> HashAggregate (cost=620724.40..635140.16 rows=182131 width=40)	
9	Group Key: cp.order_id	
10	Planned Partitions: 16	
11	-> Nested Loop Left Join (cost=1.00..551450.50 rows=690581 width=44)	
12	Join Filter: (cp.dish_id = dish_costs.dish_id)	
13	-> Index Scan using cp_index on cooking_progress cp (cost=0.42..35747.02 rows=690581 width=20)	
14	-> Materialize (cost=0.57..2744.24 rows=50 width=40)	
15	-> Subquery Scan on dish_costs (cost=0.57..2743.99 rows=50 width=40)	
16	-> GroupAggregate (cost=0.57..2743.49 rows=50 width=40)	
17	Group Key: pd.dish_id	
18	-> Nested Loop Left Join (cost=0.57..2430.29 rows=25006 width=16)	
19	-> Index Scan using pd_index on products_in_dishes pd (cost=0.29..1502.84 rows=25006 width=...	
20	-> Memoize (cost=0.29..0.31 rows=1 width=12)	
21	Cache Key: pd.product_id	
22	Cache Mode: logical	
23	-> Index Scan using products_pkey on products p (cost=0.28..0.30 rows=1 width=12)	
24	Index Cond: (product_id = pd.product_id)	
25	-> Memoize (cost=0.15..0.17 rows=1 width=444)	
26	Cache Key: o.employee_id	
27	Cache Mode: logical	
28	-> Index Scan using employees_pkey on employees e (cost=0.14..0.16 rows=1 width=444)	
29	Index Cond: (employee_id = o.employee_id)	

Рис. 25 - EXPLAIN до индексации

QUERY PLAN	
	text
1	Nested Loop Left Join (cost=40000646307.74..40042301428.16 rows=10000 width=484)
2	Join Filter: (o.order_id = orders_cost.order_id)
3	-> Nested Loop Left Join (cost=10000000000.57..10000092285.45 rows=10000 width=452)
4	-> Index Scan using orders_pkey on orders o (cost=0.43..90639.43 rows=10000 width=16)
5	Filter: ((date(open_datetime) >= '2025-01-01'::date) AND (date(open_datetime) <= '2025-12-31'::date))
6	-> Index Scan using employees_pkey on employees e (cost=0.14..0.16 rows=1 width=444)
7	Index Cond: (employee_id = o.employee_id)
8	-> Materialize (cost=20000646307.17..20000661371.04 rows=182131 width=40)
9	-> Subquery Scan on orders_cost (cost=20000646307.17..20000659037.38 rows=182131 width=40)
10	-> GroupAggregate (cost=20000646307.17..20000657216.07 rows=182131 width=40)
11	Group Key: cp.order_id
12	-> Sort (cost=20000646307.17..20000648033.63 rows=690581 width=44)
13	Sort Key: cp.order_id
14	-> Nested Loop Left Join (cost=20000000000.99..20000558084.62 rows=690581 width=44)
15	Join Filter: (cp.dish_id = dish_costs.dish_id)
16	-> Index Scan using cp_index on cooking_progress cp (cost=0.42..35747.02 rows=690581 width=20)
17	-> Materialize (cost=10000000000.56..10000009378.36 rows=50 width=40)
18	-> Subquery Scan on dish_costs (cost=10000000000.56..10000009378.11 rows=50 width=40)
19	-> GroupAggregate (cost=10000000000.56..10000009377.61 rows=50 width=40)
20	Group Key: pd.dish_id
21	-> Nested Loop Left Join (cost=10000000000.56..10000009064.41 rows=25006 width=16)
22	-> Index Scan using pd_index on products_in_dishes pd (cost=0.29..1502.84 rows=25006 width=...
23	-> Index Scan using products_pkey on products p (cost=0.28..0.30 rows=1 width=12)
24	Index Cond: (product_id = pd.product_id)

Рис. 26 - EXPLAIN после индексации

Проведя серию замеров из 30 измерений, получилось сократить среднее время выполнения с 1236мс до 1183мс.

Выводы:

В процессе выполнения лабораторной работы получилось ознакомиться с составлением INSERT, UPDATE, DELETE запросов. Также удалось ознакомиться с графическим представлением запросов. Были составлены простые и составные индексы, что позволило наглядно увидеть уменьшение кол-ва этапов при выполнении запроса, а также уменьшение времени выполнения.