

**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»

Обучающийся: Богданов Максим Александрович

Факультет прикладной информатики

Группа K3240

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2023

Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

Цель работы

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

Программное обеспечение

СУБД PostgreSQL 1X, pgAdmin 4.

Практическое задание

1. Создать запросы и представления на выборку данных к базе данных PostgreSQL (согласно индивидуальному заданию лабораторной работы №2, 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.

Схема базы данных

Модель описывает систему управления такси или автопарком с возможностью учета сотрудников, машин, заказов и клиентов. Ознакомиться с моделью можно на рисунке 1. Для отображения был использован генератор ERD-схемы в pgAdmin, настроено удобное отображение связей между таблицами.

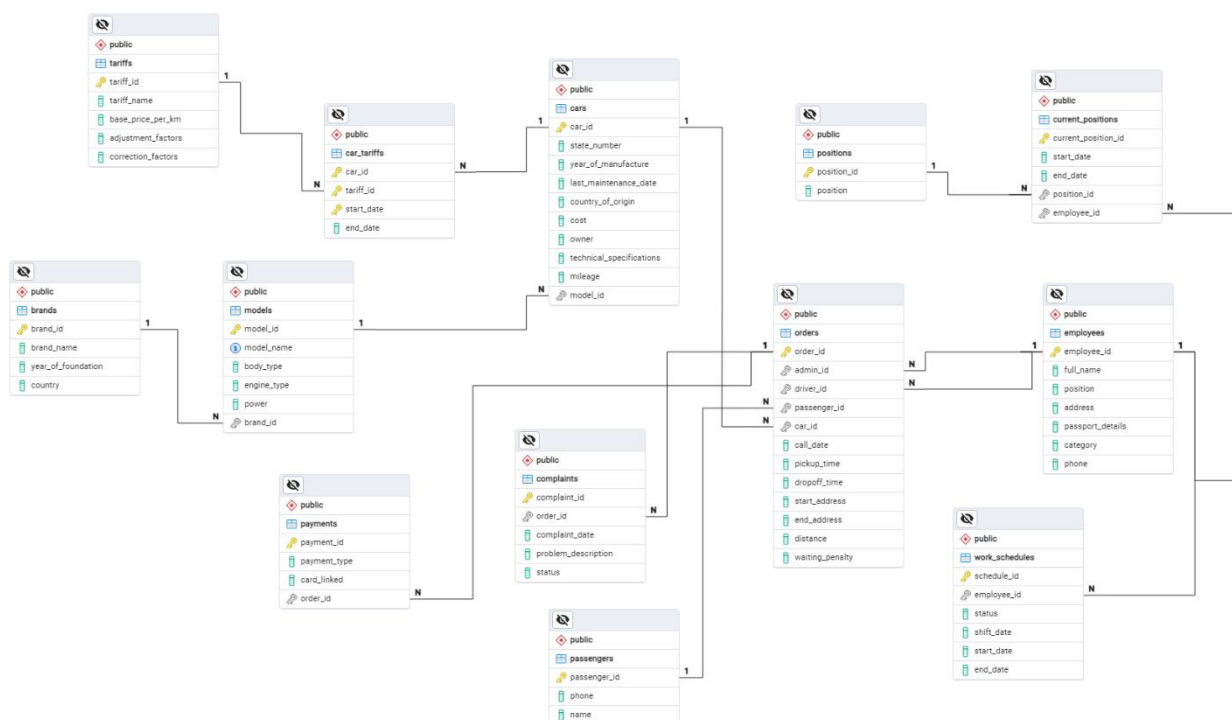


Рисунок 1 – Схема базы данных

Ход работы

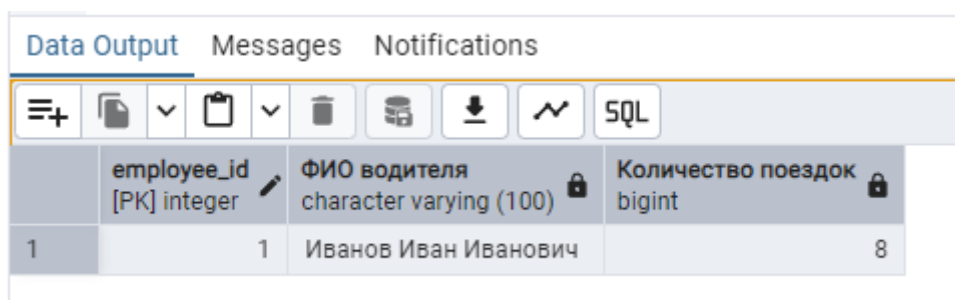
1. Запросы к БД

Выполнить запросы согласно индивидуальному заданию, часть 2. В отчете привести формулировку запроса из задания, SQL-команду, скриншот выполнения запроса с результатом

1.1. Вывести данные о водителе, который чаще всех доставляет пассажиров на заданную улицу (выбрана ул. Ленина).

```
SELECT e.employee_id, e.full_name AS "ФИО водителя", COUNT(*)  
AS "Количество поездок"  
FROM orders o  
JOIN employees e ON o.driver_id = e.employee_id  
WHERE o.end_address LIKE '%ул. Ленина%'  
GROUP BY e.employee_id, e.full_name  
ORDER BY "Количество поездок" DESC  
LIMIT 1;
```

Результат:













	employee_id [PK] integer	ФИО водителя character varying (100)	Количество поездок bigint
1	1	Иванов Иван Иванович	8

1.2. Вывести данные об автомобилях, которые имеют пробег более 250 тыс. км и не проходили ТО в 2025 году.

```
SELECT c.car_id, c.state_number AS "Госномер", c.mileage AS  
"Пробег, км", c.last_maintenance_date AS "Дата последнего ТО"  
FROM cars c  
WHERE c.mileage > 250000  
AND EXTRACT(YEAR FROM c.last_maintenance_date) != 2025;
```











Результат:

Data Output Messages Notifications				
<div></div>				
	car_id [PK] integer	Госномер character varying (9)	Пробег, км integer	Дата последнего ТО date
1	1	A123BC77	300000	2023-06-15
2	3	T789H023	280000	2022-12-20
3	4	M234PC78	260000	2023-09-01
4	6	H890YX66	320000	2022-11-10
5	8	P654AM25	270000	2023-07-25
6	10	Y432TM34	290000	2023-08-30

1.3. Сколько раз каждый пассажир воспользовался услугами таксопарка?

```
SELECT p.passenger_id, p.name AS "ФИО пассажира",  
COUNT(o.order_id) AS "Количество поездок"  
FROM passengers p  
LEFT JOIN orders o ON p.passenger_id = o.passenger_id  
GROUP BY p.passenger_id, p.name  
ORDER BY "Количество поездок" DESC;
```

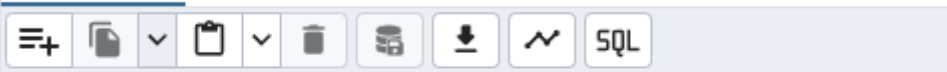
Результат:

Data Output Messages Notifications			
<div></div>			
	passenger_id [PK] integer	ФИО пассажира character varying (100)	Количество поездок bigint
1	1	Смирнова Анна Ивановна	7
2	2	Кузнецов Михаил Сергеевич	4
3	5	Соколова Мария Дмитриевна	3
4	3	Петрова Елена Викторовна	3
5	6	Лебедев Антон Павлович	3
6	4	Иванов Сергей Александрович	3
7	7	Морозова Ольга Николаевна	2
8	11	Ковалёв Дмитрий Викторович	2
9	9	Фёдорова Светлана Андреевна	1
10	10	Николаев Павел Константинович	1
11	8	Васильев Олег Игоревич	1
12	12	Зайцева Екатерина Павловна	0

1.4. Вывести данные пассажира, который воспользовался услугами таксопарка максимальное число раз.

```
SELECT p.passenger_id, p.name AS "ФИО пассажира",  
COUNT(o.order_id) AS "Количество поездок"  
FROM passengers p  
JOIN orders o ON p.passenger_id = o.passenger_id  
GROUP BY p.passenger_id, p.name  
ORDER BY "Количество поездок" DESC  
LIMIT 1;
```

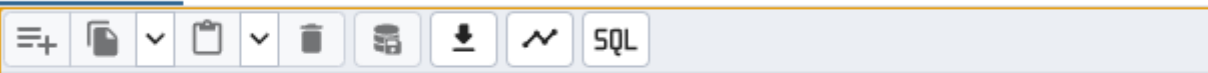
Результат:

Data Output Messages Notifications			
			
	passenger_id [PK] integer	ФИО пассажира character varying (100)	Количество поездок bigint
1	1	Смирнова Анна Ивановна	7

1.5. Вывести данные о водителе, который ездит на самом дорогом автомобиле.

```
SELECT e.employee_id, e.full_name AS "ФИО водителя",  
c.state_number AS "Госномер", c.cost AS "Стоимость авто"  
FROM employees e  
JOIN orders o ON e.employee_id = o.driver_id  
JOIN cars c ON o.car_id = c.car_id  
WHERE c.cost = (SELECT MAX(cost) FROM cars)  
LIMIT 1;
```

Результат:

Data Output Messages Notifications				
				
	employee_id integer	ФИО водителя character varying (100)	Госномер character varying (9)	Стоимость авто numeric (10,2)
1	2	Петров Пётр Петрович	K456MH50	65000.00

1.6. Вывести данные пассажира, который всегда ездит с одним и тем же водителем.

```
SELECT p.passenger_id, p.name AS "ФИО пассажира"
FROM passengers p
JOIN orders o ON p.passenger_id = o.passenger_id
GROUP BY p.passenger_id, p.name
HAVING COUNT(DISTINCT o.driver_id) = 1;
```

Результат:

	passenger_id [PK] integer	ФИО пассажира character varying (100)
1	2	Кузнецов Михаил Сергеевич
2	3	Петрова Елена Викторовна
3	4	Иванов Сергей Александрович
4	5	Соколова Мария Дмитриевна
5	6	Лебедев Антон Павлович
6	8	Васильев Олег Игоревич
7	9	Фёдорова Светлана Андреевна
8	10	Николаев Павел Константинович
9	11	Ковалёв Дмитрий Викторович

1.7. Какие автомобили имеют пробег больше среднего пробега для своей марки?

```
WITH avg_mileage_by_brand AS (
    SELECT m.brand_id, AVG(c.mileage) AS avg_mileage
    FROM cars c
    JOIN models m ON c.model_id = m.model_id
    GROUP BY m.brand_id
)
SELECT c.car_id, c.state_number AS "Госномер", b.brand_name
AS "Марка", c.mileage AS "Пробег, км", ROUND(amb.avg_mileage)
AS "Средний пробег по марке, км"
FROM cars c
JOIN models m ON c.model_id = m.model_id
JOIN brands b ON m.brand_id = b.brand_id
JOIN avg_mileage_by_brand amb ON m.brand_id = amb.brand_id
WHERE c.mileage > amb.avg_mileage;
```


Результат:

Data Output Messages Notifications				
	employee_id integer	ФИО водителя character varying (100)	Госномер character varying (9)	Стоимость авто numeric (10,2)
1	2	Петров Пётр Петрович	K456MH50	65000.00

2. Представления

Выполнить запросы на создание представлений согласно индивидуальному заданию, часть 3.

2.1. Создать представление: содержащее сведения о незанятых на данный момент водителях.

```
CREATE OR REPLACE VIEW free_drivers AS
SELECT
    e.employee_id,
    e.full_name AS "ФИО водителя",
    e.phone AS "Телефон",
    e.address AS "Адрес"
FROM employees e
JOIN current_positions cp ON e.employee_id = cp.employee_id
JOIN positions p ON cp.position_id = p.position_id
WHERE p.position = 'Водитель'
AND e.employee_id NOT IN (
    SELECT o.driver_id
    FROM orders o
    WHERE CURRENT_TIMESTAMP BETWEEN o.pickup_time AND
o.dropoff_time
)
ORDER BY e.full_name;
```

Вывод:

```
SELECT * FROM free_drivers;
```

Результат:

Data Output Messages Notifications				
SQL				
	employee_id integer	ФИО водителя character varying (100)	Телефон character varying (11)	Адрес character varying (255)
1	7	Егоров Павел Андреевич	79007778899	Москва, ул. Новый Арбат, д. 10
2	1	Иванов Иван Иванович	79001112233	Москва, ул. Ленина, д. 10
3	4	Козлов Дмитрий Сергеевич	79004445566	Москва, ул. Тверская, д. 5
4	5	Морозов Сергей Викторович	79005556677	Москва, ул. Арбат, д. 15
5	2	Петров Пётр Петрович	79002223344	Москва, ул. Мира, д. 20
6	9	Смирнов Артём Викторович	79009900112	Москва, ул. Волгоградский проспект, д. 8
7	6	Фёдоров Михаил Павлович	79006667788	Москва, ул. Кутузовский проспект, д. 25

2.2. Создать представление: зарплата всех водителей за вчерашний день.

```
CREATE OR REPLACE VIEW driver_salaries_yesterday AS
SELECT
    e.employee_id,
    e.full_name AS "ФИО водителя",
    COALESCE(SUM(o.distance * t.base_price_per_km), 0) AS
"Зарплата за 27.04.2025, руб."
FROM employees e
JOIN current_positions cp ON e.employee_id = cp.employee_id
JOIN positions p ON cp.position_id = p.position_id
LEFT JOIN orders o ON e.employee_id = o.driver_id
AND DATE(o.call_date) = CURRENT_DATE - INTERVAL '1 day'
LEFT JOIN car_tariffs ct ON o.car_id = ct.car_id
AND ct.start_date <= o.call_date
AND (ct.end_date IS NULL OR ct.end_date >= o.call_date)
LEFT JOIN tariffs t ON ct.tariff_id = t.tariff_id
WHERE p.position = 'Водитель'
GROUP BY e.employee_id, e.full_name
ORDER BY "Зарплата за 27.04.2025, руб." DESC;
```

Вывод:

```
SELECT * FROM driver_salaries_yesterday;
```

Результат:

Data Output Messages Notifications			
<div><div><div>≡+</div><div></div><div>▼</div><div></div><div>▼</div><div></div><div></div><div></div><div></div><div>SQL</div></div></div>			
	employee_id integer	ФИО водителя character varying (100)	Зарплата за 27.04.2025, руб. numeric
1	2	Петров Пётр Петрович	50.0000
2	1	Иванов Иван Иванович	48.7500
3	4	Козлов Дмитрий Сергеевич	20.0000
4	9	Смирнов Артём Викторович	0
5	5	Морозов Сергей Викторович	0
6	6	Фёдоров Михаил Павлович	0
7	7	Егоров Павел Андреевич	0

- Иванов Иван ($\text{driver_id}=1$): 2 поездки ($10.0 \text{ км} + 9.5 \text{ км}$) \times 2.50 руб/км (тариф «Стандарт») = 48.75 руб.
- Петров Пётр ($\text{driver_id}=2$): 1 поездка (12.5 км) \times 4.00 руб/км (тариф «Премиум») = 50.00 руб.
- Козлов Дмитрий ($\text{driver_id}=4$): 1 поездка (8.0 км) \times 2.50 руб/км (тариф «Стандарт») = 20.00 руб.
- Остальные водители (Морозов, Фёдоров, Егоров, Смирнов): 0 руб.

3. Выполнить запросы на модификацию данных (INSERT, UPDATE, DELETE) с использованием подзапросов (составить самостоятельно).

3.1. INSERT: Добавление нового тарифа для автомобилей с высоким пробегом

Формулировка: создать новый тариф «Высокий пробег» и применить его к автомобилям, у которых пробег превышает средний пробег по их марке, начиная с текущей даты (28 апреля 2025).

Логика:

- Создаём новый тариф «Высокий пробег» с ценой 3.50 руб/км.
- Используем подзапрос для определения автомобилей, у которых пробег выше среднего по их марке.
- Добавляем записи в car_tariffs для этих автомобилей с start_date = 28 апреля 2025.

```
-- Создаём новый тариф
INSERT INTO tariffs (tariff_name, base_price_per_km, adjustment_factors,
correction_factors)
VALUES ('Высокий пробег', 3.50, 'Для автомобилей с пробегом выше среднего', 1.15)
RETURNING tariff_id;

-- Добавляем тариф для автомобилей с высоким пробегом
INSERT INTO car_tariffs (car_id, tariff_id, start_date, end_date)
SELECT
    c.car_id,
    (SELECT tariff_id FROM tariffs WHERE tariff_name = 'Высокий пробег') AS
tariff_id,
    '2025-04-28' AS start_date,
    NULL AS end_date
FROM cars c
JOIN models m ON c.model_id = m.model_id
WHERE c.mileage > (
    SELECT AVG(c2.mileage)
    FROM cars c2
    JOIN models m2 ON c2.model_id = m2.model_id
    WHERE m2.brand_id = m.brand_id
);
```

Результат:

```
-- Проверка после выполнения
SELECT c.car_id, c.state_number AS "Госномер", c.mileage AS "Пробег, км",
t.tariff_name AS "Тариф", ct.start_date AS "Дата начала"
FROM cars c
LEFT JOIN car_tariffs ct ON c.car_id = ct.car_id
LEFT JOIN tariffs t ON ct.tariff_id = t.tariff_id
WHERE ct.end_date IS NULL
ORDER BY c.car_id;
```

	car_id integer	Госномер character varying (9)	Пробег, км integer	Тариф character varying (50)	Дата начала timestamp without time zone
1	1	A123BC77	300000	Стандарт	2024-01-01 00:00:00
2	2	K456MH50	170000	Высокий пробег	2025-04-28 00:00:00
3	2	K456MH50	170000	Премиум	2024-01-01 00:00:00
4	3	T789H023	280000	Высокий пробег	2025-04-28 00:00:00
5	3	T789H023	280000	Эконом	2024-01-01 00:00:00
6	4	M234PC78	260000	Стандарт	2024-01-01 00:00:00
7	5	E567KT97	230000	Премиум	2024-01-01 00:00:00
8	6	H890YX66	320000	Эконом	2024-01-01 00:00:00
9	6	H890YX66	320000	Высокий пробег	2025-04-28 00:00:00
10	7	O321EK47	190000	Стандарт	2024-01-01 00:00:00
11	8	P654AM25	270000	Комфорт	2024-01-01 00:00:00
12	8	P654AM25	270000	Высокий пробег	2025-04-28 00:00:00
13	9	C987X016	140000	Премиум	2024-01-01 00:00:00
14	10	Y432TM34	290000	Стандарт	2024-01-01 00:00:00
15	11	B765KP86	250000	Эконом	2024-01-01 00:00:00
16	11	B765KP86	250000	Высокий пробег	2025-04-28 00:00:00
17	12	X210CE52	200000	Эконом	2024-01-01 00:00:00

3.2. UPDATE: Увеличение цены тарифа для популярных водителей

Формулировка: увеличить цену за километр на 10% для тарифов, применённых к автомобилям водителей, которые выполнили более 5 поездок на ул. Ленина.

Логика:

- Определяем водителей, у которых более 5 поездок с конечным адресом «ул. Ленина» (подзапрос).
- Находим автомобили, связанные с этими водителями через заказы.

- Обновляем base_price_per_km в таблице tariffs для тарифов, применённых к этим автомобилям.

```
UPDATE tariffs
SET base_price_per_km = base_price_per_km * 1.10
WHERE tariff_id IN (
    SELECT ct.tariff_id
    FROM car_tariffs ct
    JOIN orders o ON ct.car_id = o.car_id
    JOIN employees e ON o.driver_id = e.employee_id
    WHERE o.end_address LIKE '%ул. Ленина%'
    AND e.employee_id IN (
        SELECT driver_id
        FROM orders
        WHERE end_address LIKE '%ул. Ленина%'
        GROUP BY driver_id
        HAVING COUNT(*) > 5
    )
    GROUP BY ct.tariff_id
);
```

Результат:

-- Проверка после выполнения

```
SELECT t.tariff_id, t.tariff_name AS "Название тарифа", t.base_price_per_km AS
"Цена за км, руб."
FROM tariffs t
ORDER BY t.tariff_id;
```

Data Output Messages Notifications			
	tariff_id [PK] integer	Название тарифа character varying (50)	Цена за км, руб. numeric (10,2)
1	1	Стандарт	2.75
2	2	Премиум	4.00
3	3	Эконом	2.00
4	4	Комфорт	3.00
5	5	Высокий пробег	3.50

3.3. DELETE: Удаление заказов с низкой выручкой

Формулировка: удалить заказы, выполненные автомобилями с тарифом «Эконом», выручка которых (расстояние × цена за км) ниже средней выручки по всем заказам с этим тарифом.

Логика:

- Находим заказы, выполненные автомобилями с тарифом «Эконом» (tariff_id=3).
- Вычисляем выручку для каждого заказа как distance * base_price_per_km.
- Используем подзапрос для расчёта средней выручки по заказам с тарифом «Эконом».
- Удаляем заказы, где выручка ниже средн

```
DELETE FROM orders
WHERE order_id IN (
SELECT o.order_id
FROM orders o
JOIN car_tariffs ct ON o.car_id = ct.car_id
JOIN tariffs t ON ct.tariff_id = t.tariff_id
WHERE t.tariff_name = 'Эконом'
AND o.distance * t.base_price_per_km < (
SELECT AVG(o2.distance * t2.base_price_per_km)
FROM orders o2
JOIN car_tariffs ct2 ON o2.car_id = ct2.car_id
JOIN tariffs t2 ON ct2.tariff_id = t2.tariff_id
WHERE t2.tariff_name = 'Эконом'
)
);
```

Результат:

```
-- Проверка до выполнения
SELECT o.order_id, o.car_id, o.distance AS "Расстояние, км", t.tariff_name AS
"Тариф",
       o.distance * t.base_price_per_km AS "Выручка, руб."
FROM orders o
JOIN car_tariffs ct ON o.car_id = ct.car_id
JOIN tariffs t ON ct.tariff_id = t.tariff_id
WHERE t.tariff_name = 'Эконом'
ORDER BY o.order_id;
```

Data Output

Messages

Notifications

SQL

	order_id integer	car_id integer	Расстояние, км numeric (10,2)	Тариф character varying (50)	Выручка, руб. numeric
1	8	6	10.00	Эконом	20.0000
2	13	6	7.00	Эконом	14.0000
3	18	6	8.00	Эконом	16.0000
4	22	11	10.50	Эконом	21.0000
5	27	6	9.50	Эконом	19.0000
6	29	11	11.50	Эконом	23.0000

-- Проверка после выполнения

```

SELECT o.order_id, o.car_id, o.distance AS "Расстояние, км", t.tariff_name AS
"Тариф",
       o.distance * t.base_price_per_km AS "Выручка, руб."
FROM orders o
JOIN car_tariffs ct ON o.car_id = ct.car_id
JOIN tariffs t ON ct.tariff_id = t.tariff_id
WHERE t.tariff_name = 'Эконом'
ORDER BY o.order_id;

```

Data Output

Messages

Notifications

☰

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	order_id integer	car_id integer	Расстояние, км numeric (10,2)	Тариф character varying (50)	Выручка, руб. numeric
1	8	6	10.00	Эконом	20.0000
2	22	11	10.50	Эконом	21.0000
3	27	6	9.50	Эконом	19.0000
4	29	11	11.50	Эконом	23.0000

4. Создание индексов

Выполнить запросы без индекса и создать планы запросов. Выполнить создание индексов. Выполнить запросы с индексами и создать планы запросов. Сравнить время выполнения запросов. Удалить индексы. В отчете отразить все команды, время запросов без индекса и с индексами.

Запрос 1: Водитель, чаще всех доставляющий на ул. Ленина

Формулировка: найти водителя, который чаще всех доставляет пассажиров на ул. Ленина.

```
EXPLAIN ANALYZE
SELECT e.employee_id, e.full_name AS "ФИО водителя", COUNT(*) AS "Количество поездок"
FROM orders o
JOIN employees e ON o.driver_id = e.employee_id
WHERE o.end_address LIKE '%ул. Ленина%'
GROUP BY e.employee_id, e.full_name
ORDER BY "Количество поездок" DESC
LIMIT 1;
```

Результат:

Data Output		Messages	Notifications
<div></div>			
	QUERY PLAN text		
1	Limit (cost=3.59..3.59 rows=1 width=230) (actual time=0.062..0.063 rows=1 loops=1)		
2	-> Sort (cost=3.59..3.59 rows=2 width=230) (actual time=0.061..0.062 rows=1 loops=1)		
3	Sort Key: (count(*)) DESC		
4	Sort Method: top-N heapsort Memory: 25kB		
5	-> HashAggregate (cost=3.56..3.58 rows=2 width=230) (actual time=0.054..0.056 rows=4 loops=1)		
6	Group Key: e.employee_id		
7	Batches: 1 Memory Usage: 24kB		
8	-> Hash Join (cost=1.04..3.48 rows=15 width=222) (actual time=0.038..0.049 rows=16 loops=1)		
9	Hash Cond: (o.driver_id = e.employee_id)		
10	-> Seq Scan on orders o (cost=0.00..2.38 rows=15 width=4) (actual time=0.019..0.027 rows=16 ...)		
11	Filter: ((end_address)::text ~~ '%ул. Ленина%':text)		
12	Rows Removed by Filter: 16		
13	-> Hash (cost=1.02..1.02 rows=2 width=222) (actual time=0.014..0.014 rows=10 loops=1)		
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB		
15	-> Seq Scan on employees e (cost=0.00..1.02 rows=2 width=222) (actual time=0.009..0.010 ...)		
16	Planning Time: 0.159 ms		
17	Execution Time: 0.254 ms		

Создадим индексы для запроса:

Фильтр WHERE o.end_address LIKE '%ул. Ленина%' требует поиска по строке с маской. Обычный B-дерево индекс неэффективен для LIKE с начальным %. Вместо этого можно использовать триграммный индекс (с расширением pg_trgm), который оптимизирует поиск по подстроке.





Соединение o.driver_id = e.employee_id выиграет от индекса на orders.driver_id.

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;

-- Индекс для LIKE-поиска по end_address
CREATE INDEX idx_orders_end_address_trgm ON orders USING GIN (end_address
gin_trgm_ops);

-- Индекс для соединения по driver_id
CREATE INDEX idx_orders_driver_id ON orders (driver_id);
```

Результат:

Data Output		Messages	Notifications
   			
	QUERY PLAN		
	text		
1	Limit (cost=3.62..3.62 rows=1 width=230) (actual time=0.055..0.056 rows=1 loops=1)		
2	-> Sort (cost=3.62..3.62 rows=2 width=230) (actual time=0.054..0.055 rows=1 loops=1)		
3	Sort Key: (count(*)) DESC		
4	Sort Method: top-N heapsort Memory: 25kB		
5	-> HashAggregate (cost=3.59..3.61 rows=2 width=230) (actual time=0.048..0.049 rows=4 loops=1)		
6	Group Key: e.employee_id		
7	Batches: 1 Memory Usage: 24kB		
8	-> Hash Join (cost=1.04..3.51 rows=16 width=222) (actual time=0.030..0.041 rows=16 loops=1)		
9	Hash Cond: (o.driver_id = e.employee_id)		
10	-> Seq Scan on orders o (cost=0.00..2.40 rows=16 width=4) (actual time=0.015..0.022 rows=16 ...)		
11	Filter: ((end_address)::text ~~ '%ул. Ленина%':text)		
12	Rows Removed by Filter: 16		
13	-> Hash (cost=1.02..1.02 rows=2 width=222) (actual time=0.010..0.011 rows=10 loops=1)		
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB		
15	-> Seq Scan on employees e (cost=0.00..1.02 rows=2 width=222) (actual time=0.005..0.006 ...)		
16	Planning Time: 1.698 ms		
17	Execution Time: 0.085 ms		

Можно видеть, что время выполнения запроса сократилось с 0.254 мс до 0.085 мс.

Запрос 2: Пассажир, едущий с одним водителем

Формулировка: вывести данные пассажира, который всегда ездит с одним и тем же водителем.

```
EXPLAIN ANALYZE
SELECT p.passenger_id, p.name AS "ФИО пассажира"
FROM passengers p
JOIN orders o ON p.passenger_id = o.passenger_id
GROUP BY p.passenger_id, p.name
HAVING COUNT(DISTINCT o.driver_id) = 1;
```

Результат:

Data Output		Messages	Notifications
<div></div>			
	QUERY PLAN text		
1	GroupAggregate (cost=4.28..4.55 rows=1 width=222) (actual time=0.064..0.069 rows=8 loops=1)		
2	Group Key: p.passenger_id		
3	Filter: (count(DISTINCT o.driver_id) = 1)		
4	Rows Removed by Filter: 2		
5	-> Sort (cost=4.28..4.36 rows=32 width=226) (actual time=0.058..0.060 rows=32 loops=1)		
6	Sort Key: p.passenger_id, o.driver_id		
7	Sort Method: quicksort Memory: 27kB		
8	-> Hash Join (cost=1.04..3.48 rows=32 width=226) (actual time=0.038..0.047 rows=32 loops=1)		
9	Hash Cond: (o.passenger_id = p.passenger_id)		
10	-> Seq Scan on orders o (cost=0.00..2.32 rows=32 width=8) (actual time=0.018..0.021 rows=32 loop...)		
11	-> Hash (cost=1.02..1.02 rows=2 width=222) (actual time=0.013..0.013 rows=12 loops=1)		
12	Buckets: 1024 Batches: 1 Memory Usage: 9kB		
13	-> Seq Scan on passengers p (cost=0.00..1.02 rows=2 width=222) (actual time=0.007..0.009 row...)		
14	Planning Time: 0.175 ms		
15	Execution Time: 0.101 ms		

Создадим индексы для запроса:

- Соединение `p.passenger_id = o.passenger_id` требует индекса на `orders.passenger_id`.

- Условие COUNT(DISTINCT o.driver_id) в HAVING работает с driver_id, нужен аналогичный индекс как и у Запроса 1.

```
CREATE INDEX idx_orders_passenger_id ON orders (passenger_id);
```

```
-- Индекс для соединения по driver_id
```

```
CREATE INDEX idx_orders_driver_id ON orders (driver_id);
```

Результат:

Data Output		Messages	Notifications
<div> </div>			
	QUERY PLAN text		
1	GroupAggregate (cost=4.28..4.55 rows=1 width=222) (actual time=0.034..0.037 rows=8 loops=1)		
2	Group Key: p.passenger_id		
3	Filter: (count(DISTINCT o.driver_id) = 1)		
4	Rows Removed by Filter: 2		
5	-> Sort (cost=4.28..4.36 rows=32 width=226) (actual time=0.030..0.031 rows=32 loops=1)		
6	Sort Key: p.passenger_id, o.driver_id		
7	Sort Method: quicksort Memory: 27kB		
8	-> Hash Join (cost=1.04..3.48 rows=32 width=226) (actual time=0.016..0.022 rows=32 loops=1)		
9	Hash Cond: (o.passenger_id = p.passenger_id)		
10	-> Seq Scan on orders o (cost=0.00..2.32 rows=32 width=8) (actual time=0.006..0.008 rows=32 loop...)		
11	-> Hash (cost=1.02..1.02 rows=2 width=222) (actual time=0.006..0.006 rows=12 loops=1)		
12	Buckets: 1024 Batches: 1 Memory Usage: 9kB		
13	-> Seq Scan on passengers p (cost=0.00..1.02 rows=2 width=222) (actual time=0.003..0.004 row...)		
14	Planning Time: 0.144 ms		
15	Execution Time: 0.055 ms		

Можно видеть, что время выполнения запроса сократилось с 0.101 мс до 0.055 мс.

Вывод

В процессе выполнения лабораторной работы созданы запросы на выборку и модификацию данных с подзапросами, включая добавление, обновление и удаление записей. Разработаны представления и индексы для оптимизации запросов, проведено сравнение времени их выполнения с индексами и без. Выявлено, что индексация эффективна для больших данных, хотя на малом объёме эффект ограничен. Работа укрепила навыки проектирования базы данных, анализа производительности и оптимизации SQL-запросов.