

**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 Клименков Владислав Максимович
Факультет прикладной информатики
Группа К3241
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

1 Цель работы

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

2 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.

3 Схема базы данных (ЛР 3)

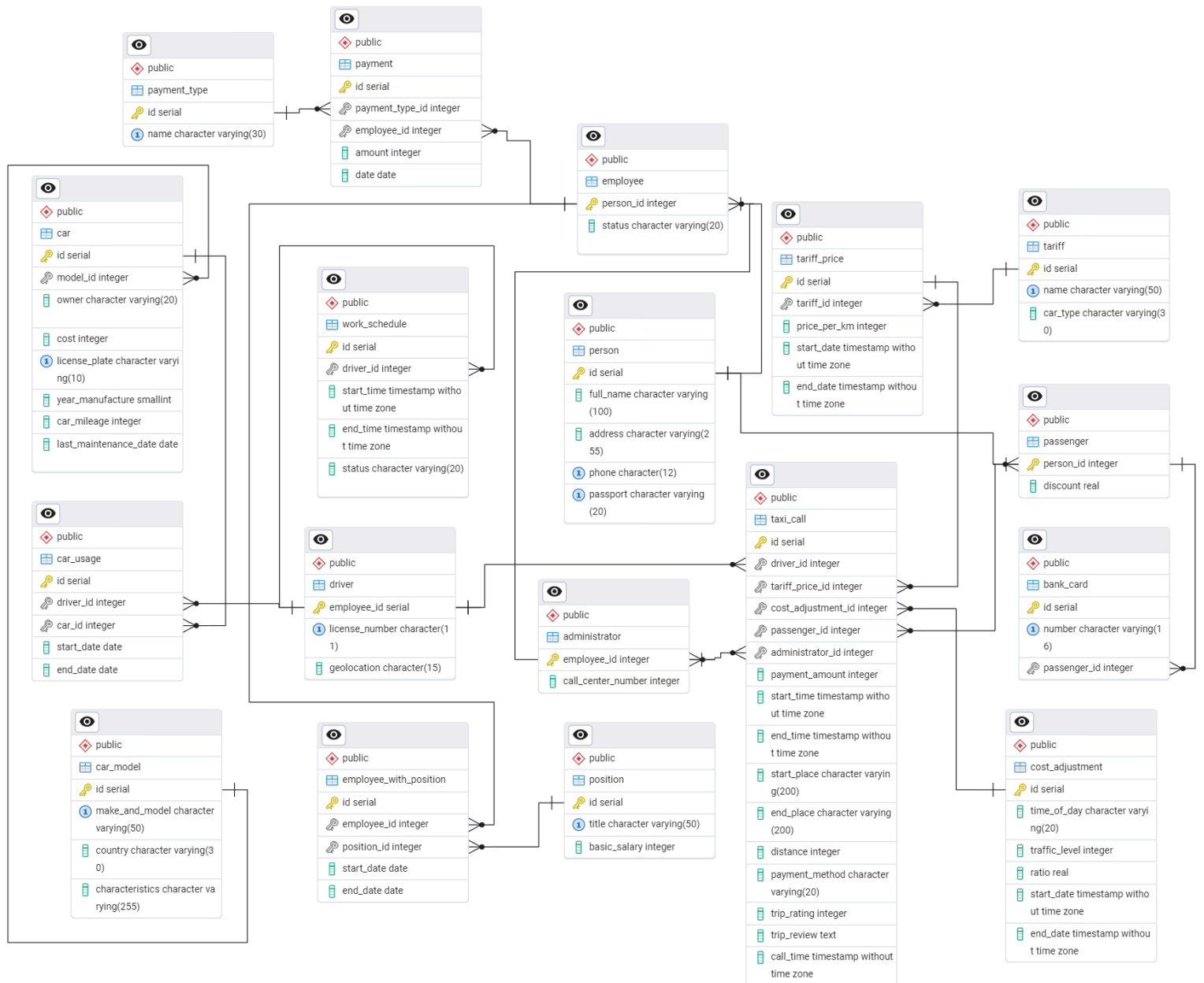


Рисунок 1 – ERD-схема базы данных

4 Выполнение

4.1 Запросы к базе данных

Запрос №1

Описание запроса:

Вывести данные о водителе, который чаще всех доставляет пассажиров на заданную улицу.

SQL-код запроса:

```
WITH delivery_counts AS (
```

```

SELECT
    d.employee_id,
    p.full_name,
    COUNT(tc.id) AS delivery_count
FROM
    driver d
JOIN
    taxi_call tc ON d.employee_id = tc.driver_id
JOIN
    public.person p ON d.employee_id = p.id
WHERE
    tc.end_place LIKE '%Восточная%'
GROUP BY
    d.employee_id, p.full_name
)

SELECT
    employee_id,
    full_name,
    delivery_count
FROM
    delivery_counts
WHERE
    delivery_count = (SELECT MAX(delivery_count) FROM delivery_counts)
ORDER BY
    employee_id;

```

Скриншот выполнения запроса:

Query

Query History

```

5      COUNT(tc.id) AS delivery_count
6  FROM
7      driver d
8  JOIN
9      taxi_call tc ON d.employee_id = tc.driver_id
10 JOIN
11     public.person p ON d.employee_id = p.id
12 WHERE
13     tc.end_place LIKE '%Восточная%'
14 GROUP BY
15     d.employee_id, p.full_name
16 )
17
18 SELECT
19     employee_id,
20     full_name,
21     delivery_count
22 FROM
23     delivery_counts
24 WHERE
25     delivery_count = (SELECT MAX(delivery_count) FROM delivery_counts)
26 ORDER BY
27     employee_id;

```

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗑️

📥

📥

📈

SQL

	employee_id integer	full_name character varying (100)	delivery_count bigint
1	148	Леонтий Викторович Лобанов	5

Запрос №2

Описание запроса:

Вывести данные об автомобилях, которые имеют пробег более 250 тысяч. километров и которые не проходили ТО в текущем году.

SQL-код запроса:

```

SELECT *
FROM car
WHERE car_mileage > 250000
      AND last_maintenance_date < DATE_TRUNC('year', CURRENT_DATE);

```

Скриншот выполнения запроса:

Query

Query History

1

SELECT *

2

FROM car

3

WHERE car_mileage > 250000

4

AND last_maintenance_date < DATE_TRUNC('year', CURRENT_DATE);

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	id [PK] integer	model_id integer	owner character varying (20)	cost integer	license_plate character varying (10)	year_manufacture smallint	car_mileage integer	last_maintenance_date date
1	7	7	водитель	1382909	B658XY52	2013	263856	2022-09-26
2	23	3	компания	2176125	P852AM74	2016	278241	2016-07-14
3	52	6	компания	5663179	C102BY98	2009	260000	2024-07-07
4	40	4	компания	9509437	B251TH67	2018	321879	2019-02-20
5	56	3	водитель	2947205	Y1870P96	2007	278125	2013-10-27

Запрос №3

Описание запроса:

Сколько раз каждый пассажир воспользовался услугами таксопарка?

SQL-код запроса:

```
SELECT
    tc.passenger_id,
    p.full_name,
    COUNT(tc.id) AS trip_count
FROM
    taxi_call tc
JOIN
    person p ON tc.passenger_id = p.id
GROUP BY
    tc.passenger_id, p.full_name
ORDER BY
    trip_count DESC;
```

Скриншот выполнения запроса:

Query		Query History	
1	SELECT		
2	tc.passenger_id,		
3	p.full_name,		
4	COUNT(tc.id) AS trip_count		
5	FROM		
6	taxi_call tc		
7	JOIN		
8	person p ON tc.passenger_id = p.id		
9	GROUP BY		
10	tc.passenger_id, p.full_name		
11	ORDER BY		
12	trip_count DESC;		

Data Output		Messages	Notifications
<div> <div>≡</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑</div> <div>🗄</div> <div>⬇</div> <div>📈</div> <div>SQL</div> </div>			
	passenger_id integer	full_name character varying (100)	trip_count bigint
1	273	Алина Вячеславовна Капустина	10
2	241	Агата Львовна Гущина	9
3	256	Сергеева Нинель Ивановна	8
4	271	Рубен Еремеевич Щукин	8
5	246	Фёкла Евгеньевна Русакова	8
6	285	Эммануил Денисович Капустин	8
7	260	Шубин Гремислав Венедиктович	8
8	243	Лидия Тимофеевна Горшкова	8
9	284	Марина Дмитриевна Петрова	7
10	276	Борисов Любомир Ефстафьевич	7
Total rows: 100 Query complete 00:00:00.135			

Запрос №4

Описание запроса:

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

SQL-код запроса:

```
WITH trip_counts AS (
    SELECT
        tc.passenger_id,
        p.full_name,
        COUNT(tc.id) AS trip_count
    FROM
        taxi_call tc
        JOIN
            person p ON tc.passenger_id = p.id
    GROUP BY
        tc.passenger_id, p.full_name
    ORDER BY
        trip_count DESC
)
```

```

JOIN
    person p ON tc.passenger_id = p.id
GROUP BY
    tc.passenger_id, p.full_name
)

SELECT
    passenger_id,
    full_name,
    trip_count
FROM
    trip_counts
WHERE
    trip_count = (SELECT MAX(trip_count) FROM trip_counts)
ORDER BY
    passenger_id;

```

Скриншот выполнения запроса:

The screenshot displays a SQL query execution interface. The top section shows the query text, which is a multi-part query using a CTE to find the passenger with the highest number of trips. The bottom section shows the results of the query, which is a single row representing the passenger with the highest trip count.

Query:

```

1 WITH trip_counts AS (
2     SELECT
3         tc.passenger_id,
4         p.full_name,
5         COUNT(tc.id) AS trip_count
6     FROM
7         taxi_call tc
8     JOIN
9         person p ON tc.passenger_id = p.id
10    GROUP BY
11        tc.passenger_id, p.full_name
12 )
13
14 SELECT
15     passenger_id,
16     full_name,
17     trip_count
18 FROM
19     trip_counts
20 WHERE
21     trip_count = (SELECT MAX(trip_count) FROM trip_counts)
22 ORDER BY
23     passenger_id;

```

Data Output:

	passenger_id integer	full_name character varying (100)	trip_count bigint
1	273	Алина Вячеславовна Капустина	10

Total rows: 1 Query complete 00:00:00.138

Запрос №5

Описание запроса:

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

SQL-код запроса:

```
SELECT
    cu.driver_id,
    p.full_name,
    c.cost,
    cu.start_date,
    cu.end_date
FROM
    car c
JOIN
    car_usage cu ON c.id = cu.car_id
    AND (cu.end_date IS NULL OR cu.end_date > CURRENT_DATE)
JOIN
    person p ON cu.driver_id = p.id
WHERE
    c.cost = (
        SELECT MAX(cost)
        FROM car
        JOIN car_usage cu2 ON car.id = cu2.car_id
        WHERE cu2.end_date IS NULL OR cu2.end_date > CURRENT_DATE
    );
```

Скриншот выполнения запроса:


```

FROM
    taxi_call
GROUP BY
    passenger_id
HAVING
    COUNT(DISTINCT driver_id) = 1
)
GROUP BY
    tc.passenger_id, p1.full_name, tc.driver_id, p2.full_name
ORDER BY
    trip_count;

```

Скриншот выполнения запроса:

Query		Query History				
1	SELECT					
2	tc.passenger_id,					
3	p1.full_name AS passenger_name,					
4	tc.driver_id,					
5	p2.full_name AS driver_name,					
6	COUNT(tc.id) AS trip_count					
7	FROM					
8	taxi_call tc					
9	JOIN					
10	person p1 ON tc.passenger_id = p1.id					
11	JOIN					
12	person p2 ON tc.driver_id = p2.id					
13	WHERE					
14	tc.passenger_id IN (
15	SELECT					
16	passenger_id					
17	FROM					
18	taxi_call					
19	GROUP BY					
20	passenger_id					
21	HAVING					
Data Output		Messages				
	passenger_id integer	passenger_name character varying (100)	driver_id integer	driver_name character varying (100)	trip_count bigint	
1	202	Евфросиния Максимовна Калинина	109	Котова Василиса Львовна	11	
2	212	Богданова Александра Аскольдовна	139	Алевтина Феликсовна Аксенова	1	
3	228	Кузьмин Елисей Игоревич	129	Костин Владимир Евсеевич	1	
4	235	Анна Эльдаровна Горбачева	117	Панфилов Эраст Дорофеевич	1	

Запрос №7

Описание запроса:

Какие автомобили имеют пробег больше среднего пробега для своей марки.

SQL-код запроса:

```

WITH average_mileage AS (
    SELECT
        c.model_id,

```

```

        AVG(c.car_mileage) AS avg_mileage
FROM
    car c
GROUP BY
    c.model_id
)

SELECT
    c.id,
    cm.make_and_model,
    c.car_mileage,
    am.avg_mileage
FROM
    public.car c
JOIN
    public.car_model cm ON c.model_id = cm.id
JOIN
    average_mileage am ON c.model_id = am.model_id
WHERE
    c.car_mileage > am.avg_mileage;

```

Скриншот выполнения запроса:

Query		Query History	
1	WITH average_mileage AS (
2	SELECT		
3	c.model_id,		
4	AVG(c.car_mileage) AS avg_mileage		
5	FROM		
6	car c		
7	GROUP BY		
8	c.model_id		
9)		
10			
11	SELECT		
12	c.id,		
13	cm.make_and_model,		
14	c.car_mileage,		
15	am.avg_mileage		
16	FROM		
17	public.car c		
18	JOIN		
19	public.car_model cm ON c.model_id = cm.id		
20	JOIN		
Data Output		Messages	Notifications
	id integer	make_and_model character varying (50)	car_mileage integer
			avg_mileage numeric
1	40	Hyundai Elantra	321879
2	23	Volkswagen Golf	278241
3	56	Volkswagen Golf	278125
4	78	Subaru Outback	274698
5	7	Kia Sportage	263856

4.2 Представления

Представление №1

Описание представления:

Создать представление, содержащее сведения о незанятых на данный момент водителях.

SQL-код представления:

```
CREATE OR REPLACE VIEW available_drivers AS
SELECT
    p.id,
    p.full_name,
    ws.start_time,
    ws.end_time
FROM
    work_schedule ws
JOIN
    person p ON ws.driver_id = p.id
WHERE
    NOW() BETWEEN ws.start_time AND ws.end_time
    AND ws.status = 'вышел';
```

Скриншоты создания и отображения представления:

The screenshot displays a database query editor interface. At the top, there are two tabs: 'Query' (selected) and 'Query History'. The 'Query' tab shows a SQL statement with line numbers 1 through 13. The statement is: `CREATE OR REPLACE VIEW available_drivers AS SELECT p.id, p.full_name, ws.start_time, ws.end_time FROM work_schedule ws JOIN person p ON ws.driver_id = p.id WHERE NOW() BETWEEN ws.start_time AND ws.end_time AND ws.status = 'вышел';`. Below the query editor, there are three tabs: 'Data Output', 'Messages' (selected), and 'Notifications'. The 'Messages' tab shows the message 'CREATE VIEW' and a status message: 'Query returned successfully in 563 msec.'

Query

Query History

1

SELECT * FROM public.available_drivers

2

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

	<div>id</div> <div>integer</div> <div>🔒</div>	<div>full_name</div> <div>character varying (100)</div> <div>🔒</div>	<div>start_time</div> <div>timestamp without time zone</div> <div>🔒</div>	<div>end_time</div> <div>timestamp without time zone</div> <div>🔒</div>
2	101	Дмитрий Федосеевич Фадеев	2025-04-21 08:00:00	2025-04-21 20:00:00
3	102	Проход Федосеевич Корнилов	2025-04-21 08:00:00	2025-04-21 20:00:00
4	103	Глафира Аскольдовна Ситникова	2025-04-21 08:00:00	2025-04-21 20:00:00
5	104	Панкрат Ааронович Колобов	2025-04-21 08:00:00	2025-04-21 20:00:00
6	105	Устинков Тит Александрович	2025-04-21 08:00:00	2025-04-21 20:00:00

Представление №2

Описание представления:

Создать представление с зарплатой всех водителей за вчерашний день.

SQL-код представления:

```
CREATE OR REPLACE VIEW driver_salary_yesterday AS
SELECT
    pay.employee_id,
    per.full_name,
    payt.name AS payment_type_name,
    pay.date,
    pay.amount
FROM
    payment pay
JOIN
    driver d ON pay.employee_id = d.employee_id
JOIN
    person per ON d.employee_id = per.id
JOIN
    payment_type payt ON pay.payment_type_id = payt.id
WHERE
    payt.name = 'Зарплата'
    AND pay.date = CURRENT_DATE - INTERVAL '1 day';
```

Скриншоты создания и отображения представления:

Query Query History

1 **CREATE OR REPLACE VIEW** driver_salary_yesterday **AS**
2 **SELECT**
3 pay.employee_id,
4 per.full_name,
5 payt.name **AS** payment_type_name,
6 pay.date,
7 pay.amount
8 **FROM**
9 payment pay
10 **JOIN**
11 driver d **ON** pay.employee_id = d.employee_id
12 **JOIN**
13 person per **ON** d.employee_id = per.id
14 **JOIN**
15 payment_type payt **ON** pay.payment_type_id = payt.id
16 **WHERE**
17 payt.name = 'Зарплата'
18 **AND** pay.date = **CURRENT_DATE** - **INTERVAL** '1 day';

Data Output Messages Notifications

CREATE VIEW

Query returned successfully in 286 msec.

Query Query History

1 **SELECT** * **FROM** public.driver_salary_yesterday
2

Data Output Messages Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

	employee_id integer	full_name character varying (100)	payment_type_name character varying (30)	date date	amount integer
1	101	Дмитрий Федосеевич Фадеев	Зарплата	2025-04-20	1904
2	102	Проход Федосеевич Корнилов	Зарплата	2025-04-20	4434
3	103	Глафира Аскольдовна Ситникова	Зарплата	2025-04-20	9781
4	104	Панкрат Ааронович Колобов	Зарплата	2025-04-20	2607
5	105	Устинов Тит Арсеньевич	Зарплата	2025-04-20	7112
6	106	Никодим Харлампьевич Гаврилов	Зарплата	2025-04-20	2059

4.3 Запросы на модификацию данных

INSERT-запрос

Описание запроса:

Выплатить премию в 100000 рублей самым долго работающим сотрудникам.

SQL-код запроса:

```
INSERT INTO payment (payment_type_id, employee_id, amount, date)
```

```

SELECT
    2,
    employee_id,
    100000,
    CURRENT_DATE
FROM
    employee_with_position
WHERE
    end_date IS NULL OR end_date > CURRENT_DATE
    AND start_date = (SELECT MIN(start_date)
                      FROM employee_with_position
                      WHERE end_date IS NULL OR end_date > CURRENT_DATE);

```

Скриншоты выполнения запроса:

Query

Query History

1

▼

SELECT * FROM public.payment

2

ORDER BY id DESC

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	id [PK] integer	payment_type_id integer	employee_id integer	amount integer	date date
1	305	1	200	4716	2025-04-20
2	304	1	199	4073	2025-04-20
3	303	1	198	3499	2025-04-20

Query	Query History
1	▼ INSERT INTO payment (payment_type_id, employee_id, amount, date)
2	SELECT
3	2,
4	employee_id,
5	100000,
6	CURRENT_DATE
7	FROM
8	employee_with_position
9	WHERE
10	end_date IS NULL OR end_date > CURRENT_DATE
11	AND start_date = (SELECT MIN (start_date)
12	FROM employee_with_position
13	WHERE end_date IS NULL OR end_date > CURRENT_DATE);

Data Output	Messages	Notifications
INSERT 0 1		
Query returned successfully in 116 msec.		

Query

Query History

1

▼

SELECT * FROM public.payment

2

ORDER BY id DESC

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	id [PK] integer	payment_type_id integer	employee_id integer	amount integer	date date
1	306	2	82	100000	2025-04-21
2	305	1	200	4716	2025-04-20
3	304	1	199	4073	2025-04-20

UPDATE-запрос

Описание запроса:

Обновить на текущую все даты технического обслуживания для всех автомобилей компании с заданной маркой, которые используются хотя бы одним водителем.

SQL-код запроса:

```
UPDATE car
SET last_maintenance_date = CURRENT_DATE
WHERE owner = 'компания'
AND model_id = 2
AND id IN (
  SELECT car_id
  FROM car_usage
  WHERE end_date IS NULL OR end_date > CURRENT_DATE
);
```

Скриншоты выполнения запроса:

Query

Query History

1

SELECT *

FROM public.car

2

ORDER BY

last_maintenance_date

DESC

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	id [PK] integer	model_id integer	owner character varying (20)	cost integer	license_plate character varying (10)	year_manufacture smallint	car_mileage integer	last_maintenance_date date
1	35	4	компания	6956567	C332E077	2014	38532	2025-03-22
2	5	9	водитель	5381433	C3940K90	2024	76057	2025-03-13
3	59	3	водитель	3550666	C730TH24	2008	79441	2025-02-13
4	78	9	компания	4576045	T920KY07	2024	274698	2025-02-07

QueryQuery History

1

▼

UPDATE car

2

SET last_maintenance_date = CURRENT_DATE

3

WHERE owner = 'компания'

4

AND model_id = 2

5

AND id IN (

6

SELECT car_id

7

FROM car_usage

8

WHERE end_date IS NULL OR end_date > CURRENT_DATE

9

);

Data Output

Messages

Notifications

UPDATE 2

Query returned successfully in 55 msec.

QueryQuery History

1

▼

SELECT * FROM public.car

2

ORDER BY last_maintenance_date DESC

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🔍

📥

📤

📶

SQL

	id [PK] integer	model_id integer	owner character varying (20)	cost integer	license_plate character varying (10)	year_manufacture smallint	car_mileage integer	last_maintenance_date date
1	85	2	компания	3929318	B272YC22	2021	48192	2025-04-21
2	92	2	компания	9897920	C340KP78	2023	25511	2025-04-21
3	35	4	компания	6956567	C332EO77	2014	38532	2025-03-22
4	5	9	водитель	5381433	C394OK90	2024	76057	2025-03-13

DELETE-запрос

Описание запроса:

Удалить всех пассажиров, у которых нет привязанной банковской карты и которые не совершили ни одного вызова такси.

SQL-код запроса:

```
DELETE FROM passenger
WHERE person_id NOT IN (
    SELECT passenger_id FROM bank_card
)
AND person_id NOT IN (
    SELECT passenger_id FROM taxi_call
);
```

Скриншоты выполнения запроса:

Query

Query History

1

▼

SELECT * FROM public.passenger

2

ORDER BY person_id DESC

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	person_id [PK] integer	discount real
1	379	0.4
2	378	0.2
3	377	0.3
4	376	0.4
5	375	0.1

Query		Query History
1	▼	DELETE FROM passenger
2		WHERE person_id NOT IN (
3		SELECT passenger_id FROM bank_card
4)
5		AND person_id NOT IN (
6		SELECT passenger_id FROM taxi_call
7);
Data Output		Messages
		DELETE 50
Query returned successfully in 199 msec.		

Query

Query History

1

▼

SELECT * FROM public.passenger

2

ORDER BY person_id DESC

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼



🗑️

🗄️

⬇️

📈

SQL

	person_id [PK] integer 	discount real 
1	300	0.4
2	299	0.5
3	298	0.3
4	297	0.4
5	296	0.1

4.4 Создание индексов

Простой индекс

SQL-код запроса:

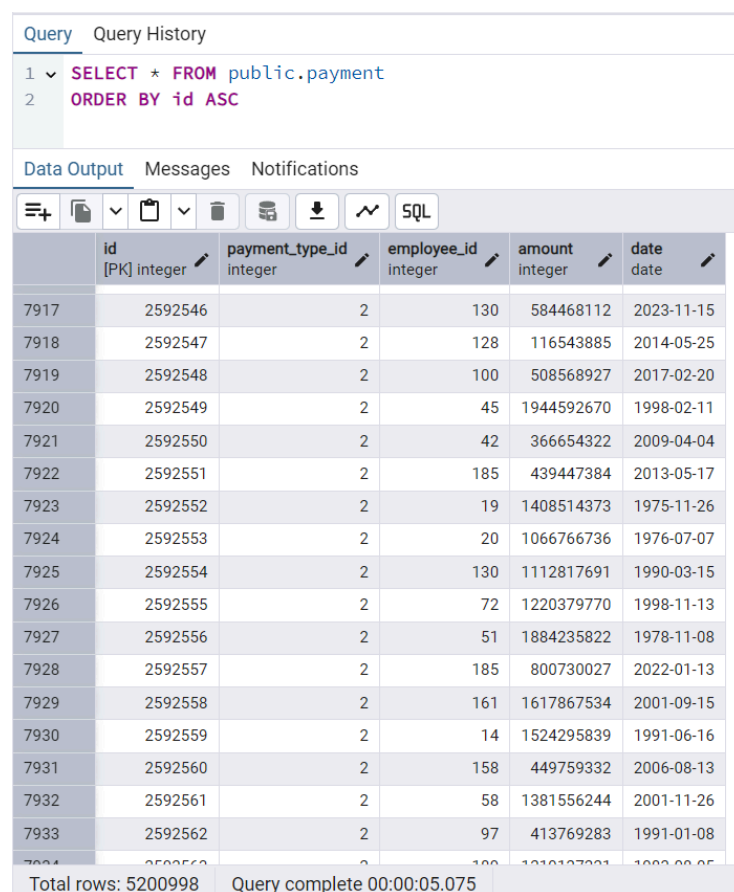
```
SELECT * FROM payment WHERE amount > 1999999000  
ORDER BY amount DESC;
```

SQL-код индекса:

```
CREATE INDEX idx_amount ON payment(amount);
```

Скриншоты работы с индексом:

Чтобы нагляднее увидеть влияние индекса заполним таблицу payment большим количеством записей (больше 5 000 000) с различными значениями в столбце amount (от 1 до 2 000 000 000):



The screenshot shows a database query interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:
1 SELECT * FROM public.payment
2 ORDER BY id ASC
Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table of results. The table has six columns: 'id' (PK integer), 'payment_type_id' (integer), 'employee_id' (integer), 'amount' (integer), and 'date' (date). The table contains 20 rows of data, with 'id' values ranging from 7917 to 7933. At the bottom of the table, there is a summary row: 'Total rows: 5200998' and 'Query complete 00:00:05.075'.

	id [PK] integer	payment_type_id integer	employee_id integer	amount integer	date date
7917	2592546	2	130	584468112	2023-11-15
7918	2592547	2	128	116543885	2014-05-25
7919	2592548	2	100	508568927	2017-02-20
7920	2592549	2	45	1944592670	1998-02-11
7921	2592550	2	42	366654322	2009-04-04
7922	2592551	2	185	439447384	2013-05-17
7923	2592552	2	19	1408514373	1975-11-26
7924	2592553	2	20	1066766736	1976-07-07
7925	2592554	2	130	1112817691	1990-03-15
7926	2592555	2	72	1220379770	1998-11-13
7927	2592556	2	51	1884235822	1978-11-08
7928	2592557	2	185	800730027	2022-01-13
7929	2592558	2	161	1617867534	2001-09-15
7930	2592559	2	14	1524295839	1991-06-16
7931	2592560	2	158	449759332	2006-08-13
7932	2592561	2	58	1381556244	2001-11-26
7933	2592562	2	97	413769283	1991-01-08
Total rows: 5200998 Query complete 00:00:05.075					

Проверим скорость выполнения запроса без индекса:

Query	Query History
1	SELECT * FROM payment WHERE amount > 1999999000
2	ORDER BY amount DESC;

Data Output	Messages	Explain	Notifications
Successfully run. Total query runtime: 447 msec. 1 rows affected.			

Среднее время выполнения запроса 400-500 мс.

Теперь рассмотрим раздел Explain:

Query	Query History
1	SELECT * FROM payment WHERE amount > 1999999000
2	ORDER BY amount DESC;

Data Output	Messages	Explain	Notifications
<div> <div>Graphical</div> <div>Analysis</div> <div>Statistics</div> </div> <div> <div>🔍</div> <div>🔄</div> <div>🔍</div> <div>📄</div> </div> <div> <div>payment</div> <div>Sort</div> <div>Gather Merge</div> </div>			

Как можно заметить, использовался механизм “Gather Merge”, позволяющий разбить вычисления на несколько параллельных процессов, а потом объединить их результаты. Это позволяет увеличить скорость выполнения запроса.

Теперь создадим индекс для столбца “amount” (используем DESC, так как в оптимизированном запросе используется “ORDER BY ... DESC”):

Query	Query History
1	CREATE INDEX idx_amount ON payment (amount DESC);

Data Output	Messages	Notifications
CREATE INDEX		
Query returned successfully in 10 secs 560 msec.		

Выполним тот же запрос, но на этот раз при созданном индексе:


Query	Query History
1 SELECT * FROM payment WHERE amount > 1999999000	
2 ORDER BY amount DESC ;	

Data Output	Messages	Notifications
Successfully run. Total query runtime: 117 msec.		
1 rows affected.		

Среднее время выполнение запроса снизилось до 100-200 мс. Разница заметна, но не настолько ощутима, так как и без индекса выполнение запроса происходит достаточно быстро (возможно, в том числе и за счёт параллельных вычислений).

Проверим раздел Explain:

Query	Query History
1 SELECT * FROM payment WHERE amount > 1999999000	
2 ORDER BY amount DESC ;	

Data Output	Messages	Explain X	Notifications
<div> <div>Graphical</div> <div>Analysis</div> <div>Statistics</div> </div> <div> <div>🔍</div> <div>🔄</div> <div>🔍</div> <div>📄</div> </div> <div>  <p>idx_amount</p> </div>			

Как можно заметить, в этот раз в качестве плана использовался созданный нами индекс, что и позволило увеличить скорость выполнения.

Теперь удалим индекс:

Query	Query History
1 DROP INDEX idx_amount;	

Data Output	Messages	Notifications
DROP INDEX		
Query returned successfully in 277 msec.		

Итоговое время:

Без индекса: 400-500 мс в среднем.

С индексом: 100-200 мс в среднем.

Составной индекс

SQL-код запроса:

```
SELECT \*  
FROM car  
WHERE year_manufacture >= 2024 AND car_mileage <= 1000  
ORDER BY cost DESC;
```

SQL-код индекса:

```
CREATE INDEX idx_car_year_cost_mileage ON car (year_manufacture, cost DESC,  
car_mileage);
```

Скриншоты работы с индексом:

Заполним таблицу car большим количеством случайно сгенерированных данных (более 3 000 000 строк):

Query

Query History

Scratch Pad

1 SELECT * FROM public.car

2 ORDER BY id ASC

Data Output

Messages

Notifications

Showing rows: 4001 to 5000

Page No: 5

of 3172

	id [PK] integer	model_id integer	owner character varying (20)	cost integer	license_plate character varying (10)	year_manufacture smallint	car_mileage integer	last_maintenance_date date
4025	28300	9	водитель	613093333	H907MA53	2010	634297265	2012-08-24
4026	28301	10	водитель	1962676401	T636BM86	2013	1892920619	2022-05-07
4027	28302	10	компания	1812865634	Y353HA64	2023	949069231	2025-01-23
4028	28303	7	компания	72552105	C359AC69	2022	1372765714	2022-07-31
4029	28304	3	водитель	83045586	E023KM37	2008	1095699721	2013-10-02
4030	28305	6	водитель	1361350116	O647BK14	2016	926631688	2020-07-29
4031	28306	5	водитель	165583030	A409HY41	2005	1999289732	2025-04-02
4032	28307	8	водитель	1285614800	B070MT40	2022	904264260	2024-04-11
4033	28308	3	компания	1092822876	P7700E27	2014	844305736	2020-05-08
4034	28309	1	компания	1929673042	H650CO22	2015	1717642328	2018-01-02
4035	28310	9	компания	930351680	O598TH71	2024	59073262	2025-02-27
4036	28311	5	водитель	395279509	C489YK41	2017	423310574	2023-04-11
4037	28312	10	водитель	1146998613	C186AH42	2007	1249103259	2014-03-01
4038	28313	1	водитель	1448838653	T055EP41	2007	559719177	2008-04-30
4039	28314	1	компания	1561782572	O038OH28	2014	777731887	2022-01-14
4040	28315	8	водитель	1835844178	K792BA84	2009	1242220226	2013-06-12
4041	28316	10	водитель	304399128	K256MT43	2013	461160931	2019-04-04
4042	28317	4	водитель	859172261	E224YP81	2023	624838464	2023-12-09

Total rows: 3171984

Query complete 00:00:29.969

CRLF

Ln 2, Col 17

Проверим скорость выполнения запроса без индекса:

Query
Query History

1 ▼ **SELECT** *
2 **FROM** car
3 **WHERE** year_manufacture >= 2024 **AND** car_mileage <= 10000
4 **ORDER BY** cost **DESC**;

Data Output
Messages
Notifications

Successfully run. Total query runtime: 435 msec.

1 rows affected.

Среднее время выполнения 400-500 мс.

Рассмотрим раздел Explain:

Query
Query History

1 ▼ **SELECT** *
2 **FROM** car
3 **WHERE** year_manufacture >= 2024 **AND** car_mileage <= 10000
4 **ORDER BY** cost **DESC**;

Data Output
Messages
Explain ×
Notifications

Graphical
Analysis
Statistics

🔍
🔄
🔍
📄

car

Sort

Gather Merge

Теперь создадим составной индекс для столбцов “year_manufacture”, “cost DESC” и “car_mileage”:

Query
Query History

1 **CREATE INDEX** idx_car_year_cost_mileage **ON** car (year_manufacture, cost **DESC**, car_mileage);

Data Output
Messages
Notifications

CREATE INDEX

Query returned successfully in 21 secs 644 msec.

Проверим скорость работы запроса при созданном составном индексе:

Query	Query History
1	SELECT *
2	FROM car
3	WHERE year_manufacture >= 2024 AND car_mileage <= 10000
4	ORDER BY cost DESC;

Data Output	Messages	Notifications
Successfully run. Total query runtime: 124 msec.		
1 rows affected.		

Среднее время выполнения снизилось до 100-200 мс.

Рассмотрим раздел Explain:

Query	Query History
1	SELECT *
2	FROM car
3	WHERE year_manufacture >= 2024 AND car_mileage <= 10000
4	ORDER BY cost DESC;

Data Output	Messages	Explain	Notifications
<div>Graphical Analysis Statistics</div> <div> <pre> graph LR A[idx_car_year_cost_mileage] --> B[Sort] </pre> </div>			

Как можно заметить, созданный индекс применяется при выполнении запроса.

Теперь удалим индекс:

Query	Query History
1	DROP INDEX idx_car_year_cost_mileage;

Data Output	Messages	Notifications
DROP INDEX		
Query returned successfully in 55 msec.		

Итоговое время:

Без индекса: 400-500 мс в среднем.

С индексом: 100-200 мс в среднем.

5 Выводы

В рамках данной работы удалось выполнить все поставленные практические задачи:

- Создать запросы и представления на выборку данных к базе данных PostgreSQL (согласно индивидуальному заданию лабораторной работы №2, часть 2 и 3).
- Составить 3 запроса на модификацию данных (INSERT, UPDATE, DELETE) с использованием подзапросов.
- 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.

При работе с индексами удалось выяснить, что при большом количестве обрабатываемых данных простые и составные индексы способны ускорить выполнение определённых SQL-команд. В рамках работы удалось добиться ускорения с примерно 400-500 мс до 100-200 мс на запрос как для простого, так и для составного индексов.