

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра информатики

ГРУДИНИН Михаил Артемович

БАЗА ДАННЫХ ДЛЯ КАРШЕРИНГА

Проектная работа

студента 1 курса дневного отделения, группы 23.Б16

Научный руководитель:
Шевнин Лев Ярославович

Работа предоставлена на кафедру

«11» мая 2024 г.

Санкт-Петербург
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ГЛАВА 1. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ	6
1.1 ER-Диаграмма	6
1.2 Описание	6
1.3 Связи	7
1.4 Итоги	8
ГЛАВА 2. СОЗДАНИЕ БАЗЫ ДАННЫХ И РАБОТА С ДАННЫМИ	9
2.1 Создание базы данных при помощи SQL запроса	9
2.2 Создание таблиц	9
2.3 Заполнение таблиц	12
2.4 Изменение данных таблицы	14
2.5 Удаление данных таблицы	14
2.6 Итоги	16
ГЛАВА 3. РАБОТА С ЗАПРОСАМИ SQL	18
3.1 Подготовка таблиц	18
3.2 Запросы на однотабличную выборку данных с различными условиями	19
3.3 Запросы на многотабличную выборку данных с различными условиями	20
3.4 Запросы с параметром	21
3.5 Запросы на группировку данных	21
3.6 Запросы на создание вычисляемых полей с датой и временем	22
3.7 Запросы на создание таблицы	23
3.8 Параметрический запрос на добавление одной записи	24
3.9 Параметрический запрос на добавление нескольких записей	25
3.10 Параметрический запрос на удаление записей	26
3.11 Параметрический запрос на обновление полей записей	28
3.12 Вложенные запросы	29
3.13 Итоги	31
ГЛАВА 4. ПРОДВИНУТЫЕ ЗАПРОСЫ	33
4.1 Вложенные запросы	33
4.2 Индексация столбцов	36
4.3 Ограничение первичных и внешних ключей	37
4.4 Итоги	39
ГЛАВА 5. Функции и триггеры	40
5.1 Триггер для установки статуса операции	40
5.2 Триггер для обновления данных при добавлении новых строк или изменении количества	42
5.3 Триггер для проверки ограничения на количество	44

5.4 Реализация вычислительной функции	47
5.5 Итоги	48
ЗАКЛЮЧЕНИЕ	50
СПИСОК ЛИТЕРАТУРЫ	51

ВВЕДЕНИЕ

Каршеринг - это технология предоставления услуг по аренде автомобилей. С каждым годом каршеринг становится все более популярным как удобный и экономичный способ получения доступа к автомобилю без необходимости его покупки. Этот рост востребованности требует эффективного управления данными для обеспечения оперативного и безопасного предоставления услуг. То есть, для эффективной работы системы каршеринга необходима надежная и хорошо структурированная база данных, которая будет хранить информацию об автомобилях, клиентах, арендах, тарифах и многом другом [1]. Такая база данных должна обеспечивать быстрый доступ к необходимым данным и их целостность.

Таким образом, **целью** работы стала разработка базы данных, которая способна эффективно управлять необходимыми для работы каршеринговой компании данными с использованием СУБД PostgreSQL и графического клиента pgAdmin4.

Для достижения цели были сформулированы и решены **задачи**:

1. Спроектировать базу данных: определить ее структуру, включая сущности (таблицы), их атрибуты (поля) и связи между ними. Необходимо учесть различные аспекты, такие как типы данных, первичные и внешние ключи для обеспечения целостности;
2. Реализовать базу данных: создать ее на основе спроектированной схемы с использованием PostgreSQL. Это включает в себя создание таблиц, определение правил связей и установку индексов для оптимизации запросов;
3. Реализовать SQL запросы для управления данными: выполнить операции добавления, изменения и удаления данных в базе. Необходимо также сформировать запросы для выборки данных с различными условиями;
4. Внедрить функциональные требования: разработать триггеры, функции

и хранимые процедуры для автоматизации определенных процессов.

ГЛАВА 1. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

1.1 ER-Диаграмма

В работе по проектированию базы данных для построения ее архитектуры использовали самостоятельно созданную ER-диаграмму, которая представлена на рисунке 1 [2].

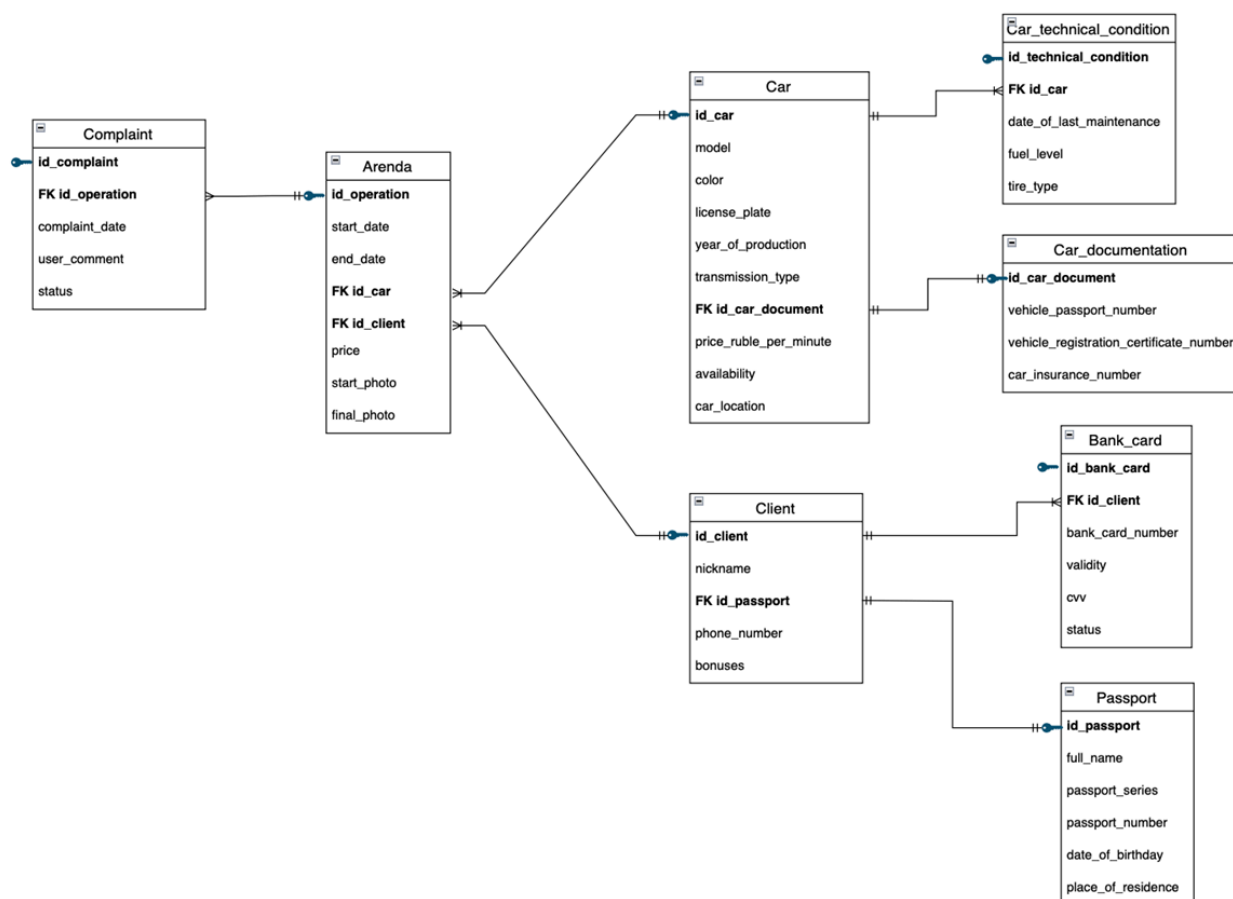


Рисунок 1 — ER-Диаграмма архитектуры базы данных

1.2 Описание

В ней существует таблица *Arenda*, в которой хранятся данные о конкретной поездке. Она связана с другими таблицами:

1. *Complaint* – здесь данные о жалобах пользователей;
2. *Car* – в ней информация об арендованном автомобиле;
3. *Client* – содержит данные о пользователе, который арендовал машину.

В свою очередь таблица *Cars* связана с:

1. *Car_technical_condition* – информацией о техническом состоянии машины;
2. *Car_documentation* – данными о документах автомобиля.

И, как мы видим из диаграммы, таблица *Clients* также связана с:

1. *Passport* – таблицей, в которой хранится информация о паспортных данных пользователя;
2. *Bank_card* – таблицей, содержащей данные о привязанных картах пользователя.

1.3 Связи

В базе данных (рисунок 1) существует два типа связей: «один к одному» и «один ко многим».

Связи «один ко многим»:

1. *Arenda - Complaint* – в процессе одной аренды могло произойти несколько жалоб;
2. *Car - Arenda* – одна и та же машина могла быть арендована много раз;
3. *Car - Car_technical_condition* – одной конкретной машине может соответствовать несколько технических состояний (то есть, некая история состояний машины);
4. *Client - Arenda* – один и тот же пользователь мог произвести процесс аренды много раз;
5. *Client - Bank_card* – у одного конкретного пользователя может быть привязано несколько банковских карт.

Связи «один к одному»:

1. *Car - Documentation* – одной конкретной машине соответствуют одни

конкретные документы;

2. *Client - Passport* – одному конкретному пользователю соответствует один конкретный паспорт.

1.4 Итоги

- Разработана диаграмма базы данных с 8 таблицами;
- Были описаны все таблицы базы данных;
- Определены связи между таблицами, установлены первичные и внешние ключи для обеспечения целостности данных.

ГЛАВА 2. СОЗДАНИЕ БАЗЫ ДАННЫХ И РАБОТА С ДАННЫМИ

2.1 Создание базы данных при помощи SQL запроса

Для создания базы данных выполнили следующий запрос:

CREATE DATABASE DatabaseBasicsSPbU2sem

Результат запроса представлен на рисунке 2.

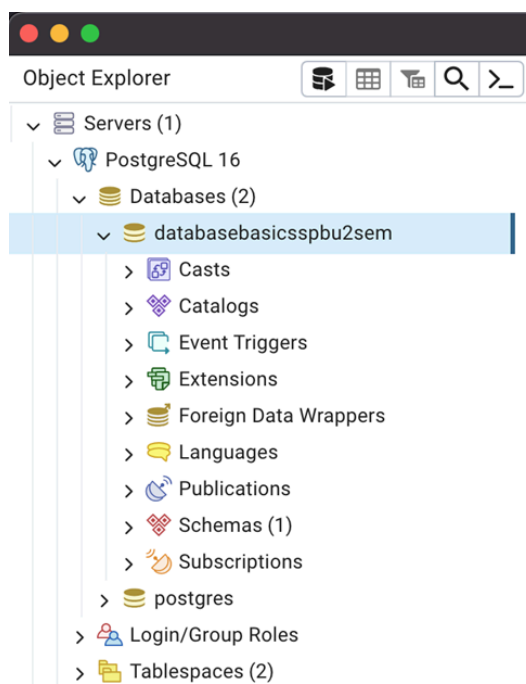


Рисунок 2 - Создание базы данных

2.2 Создание таблиц

Приступили к созданию таблиц в SQL. Ограничения **NOT NULL** на позиции с **PRIMARY KEY** не добавляли, так как в PostgreSQL **PRIMARY KEY** уже включает в себя проверку на **NOT NULL**.

Зная, что тип данных **SERIAL** поддерживает как автоинкрементирование, так и ручное указание id, выполнили следующие запросы [3]:

```
CREATE TABLE Car_documentation (  
    id_car_document SERIAL PRIMARY KEY,  
    vehicle_passport_number SMALLSERIAL,
```

```

    vehicle_registration_certificate_number SMALLSERIAL,
    car_insurance_number SMALLSERIAL
);

CREATE TABLE Car (
    id_car SERIAL PRIMARY KEY,
    model VARCHAR(30),
    color VARCHAR(20),
    license_plate VARCHAR(15),
    year_of_production SMALLSERIAL,
    transmission_type VARCHAR(20),
    id_car_document SERIAL REFERENCES Car_documentation(id_car_document),
    price_ruble_per_minute SMALLSERIAL,
    availability BOOLEAN,
    car_location VARCHAR(50)
);

CREATE TABLE Car_technical_condition (
    id_technical_condition SERIAL PRIMARY KEY,
    id_car SERIAL REFERENCES Car(id_car),
    date_of_last_maintenance DATE,
    fuel_level SMALLSERIAL,
    tire_type VARCHAR
);

CREATE TABLE Passport (
    id_passport SERIAL PRIMARY KEY,
    full_name VARCHAR,
    passport_series SMALLSERIAL,
    passport_number SERIAL,
    date_of_birthday DATE,
    place_of_residence VARCHAR
);

CREATE TABLE Client (
    id_client SERIAL PRIMARY KEY,

```

```

nickname VARCHAR(30),
id_passport SERIAL REFERENCES Passport(id_passport),
phone_number BIGSERIAL,
bonuses SMALLSERIAL
);

CREATE TABLE Bank_card (
    id_bank_card SERIAL PRIMARY KEY,
    id_client SERIAL REFERENCES Client(id_client),
    bank_card_number BIGSERIAL,
    validity CHAR(5),
    cvv SMALLSERIAL,
    status BOOLEAN
);

CREATE TABLE Arenda (
    id_operation SERIAL PRIMARY KEY,
    start_date TIMESTAMP,
    end_date TIMESTAMP,
    id_car SERIAL REFERENCES Car(id_car),
    id_client SERIAL REFERENCES Client(id_client),
    price SERIAL,
    start_photo VARCHAR,
    final_photo VARCHAR
);

CREATE TABLE Complaint (
    id_complaint SERIAL PRIMARY KEY,
    id_operation SERIAL REFERENCES Arenda(id_operation),
    complaint_date TIMESTAMP,
    user_comment VARCHAR,
    status BOOLEAN
);

```

Результат запроса представлен на рисунке 3.

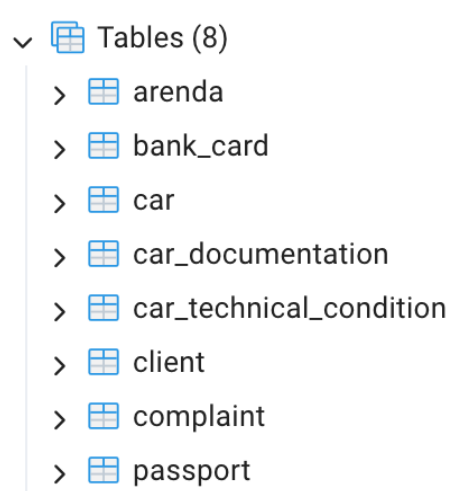


Рисунок 3 – Создание таблиц

2.3 Заполнение таблиц

Для заполнения созданных таблиц выполнили следующий запрос:

```
INSERT INTO Car_documentation(id_car_document, vehicle_passport_number,
vehicle_registration_certificate_number, car_insurance_number) VALUES
(1, 1234, 5678, 9012),
(2, 2109, 8765, 4321),
(3, 5423, 1265, 1238);
```

```
INSERT INTO Car(id_car, model, color, license_plate, year_of_production,
transmission_type, id_car_document, price_ruble_per_minute, availability, car_location)
VALUES
(1, 'Alfa Romeo 164', 'Зеленый', 'p287mx74', 1991, 'МКПП', 1, 100, true, 'с.
Chelyabinsk, st. Degtyareva 1'),
(2, 'Toyota Carina 2', 'Серый', 'a913ea74', 1991, 'МКПП', 2, 90, true, 'с.
Chelyabinsk, st. Degtyareva 2'),
(3, 'Lifan x50', 'Синий', 'e653pc774', 2016, 'АКПП', 3, 8, false, 'с. Chelyabinsk, st.
Degtyareva 3');
```

```
INSERT INTO Car_technical_condition(id_technical_condition, id_car,
date_of_last_maintenance, fuel_level, tire_type) VALUES
```

(1, 1, '2021-01-14', 20, 'Зимняя'),
 (2, 2, '2020-07-09', 40, 'Летняя'),
 (3, 3, '2024-12-01', 90, 'Зимняя'),
 (4, 1, '2025-01-14', 30, 'Зимняя');

Результаты запроса представлен на рисунках 4, 5, 6.

1 **SELECT** * **FROM** Car

Data Output Messages Notifications

	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying (15)	year_of_production smallint	transmis characte
1	1	Alfa Romeo 164	Зеленый	p287mx74	1991	МКПП
2	2	Toyota Carina 2	Серый	a913ea74	1991	МКПП
3	3	Lifan x50	Синий	e653pc774	2016	АКПП

Рисунок 4 –Заполнение таблицы Car

1 **SELECT** * **FROM** Car_documentation

Data Output Messages Notifications

	id_car_document [PK] integer	vehicle_passport_number smallint	vehicle_registration_certificate_number smallint	car_insurance_nu smallint
1	1	1234	5678	
2	2	2109	8765	
3	3	5423	1265	

Рисунок 5 – Заполнение таблицы Car_documentation

1 **SELECT** * **FROM** Car_technical_condition

Data Output Messages Notifications

	id_technical_condition [PK] integer	id_car integer	date_of_last_maintenance date	fuel_level smallint	tire_type character varying
1	1	1	2021-01-14	20	Зимняя
2	2	2	2020-07-09	40	Летняя
3	3	3	2024-12-01	90	Зимняя
4	4	1	2025-01-14	30	Зимняя

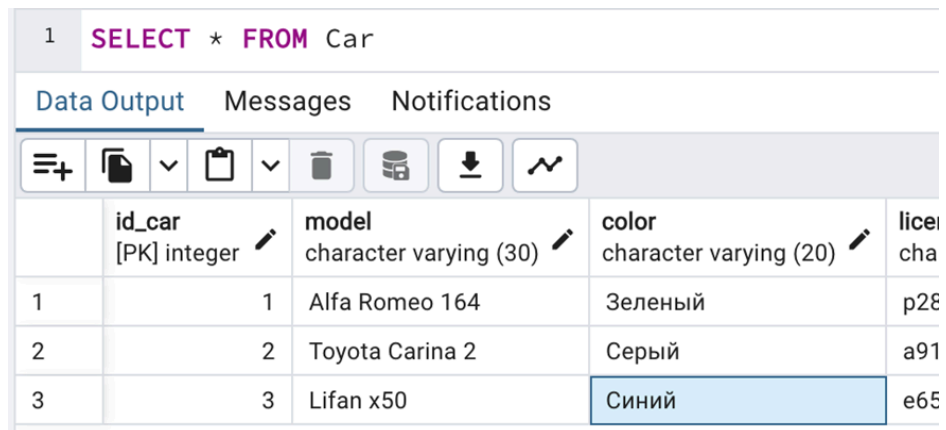
Рисунок 6 – Заполнение таблицы Car_yechnical_condition

2.4 Изменение данных таблицы

Чтобы изменить данные таблицы, выполнили запрос:

```
UPDATE car SET color = 'Черный' WHERE id_car = 3
```

Его результат, а так же вариант «до» представлены на рисунках 7, 8.

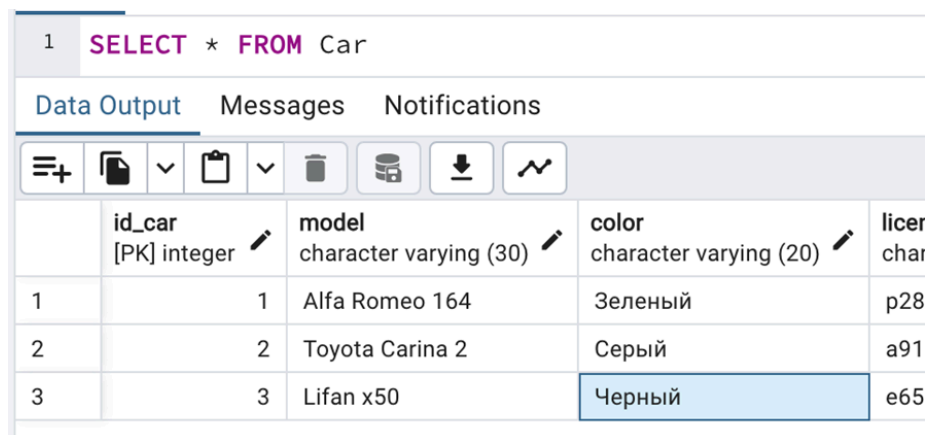


1 **SELECT** * **FROM** Car

Data Output Messages Notifications

	id_car [PK] integer	model character varying (30)	color character varying (20)	licen cha
1	1	Alfa Romeo 164	Зеленый	p28
2	2	Toyota Carina 2	Серый	a91
3	3	Lifan x50	Синий	e65

Рисунок 7 – Результат «до» запроса



1 **SELECT** * **FROM** Car

Data Output Messages Notifications

	id_car [PK] integer	model character varying (30)	color character varying (20)	licen char
1	1	Alfa Romeo 164	Зеленый	p28
2	2	Toyota Carina 2	Серый	a91
3	3	Lifan x50	Черный	e65

Рисунок 8 – Результат «после» запроса

2.5 Удаление данных таблицы

Чтобы удалить данные таблицы, выполнили запросы:

```
DELETE FROM public.car_technical_condition
```

```
WHERE public.car_technical_condition.id_technical_condition = '3';
```

```
DELETE FROM public.car
```

WHERE public.car.id_car = '3';

DELETE FROM public.car_documentation

WHERE public.car_documentation.id_car_document = '3';

Результаты «до» и «после» представлены на рисунках 9-14.

1

SELECT * FROM Car

Data Output

Messages

Notifications

≡+

▼

▼

	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varyi
1	1	Alfa Romeo 164	Зеленый	p287mx74
2	2	Toyota Carina 2	Серый	a913ea74
3	3	Lifan x50	Черный	e653pc774

Рисунок 9 – Результат «до» таблицы Car

1

SELECT * FROM Car

2

Data Output

Messages

Notifications

	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying
1	1	Alfa Romeo 164	Зеленый	p287mx74
2	2	Toyota Carina 2	Серый	a913ea74

Рисунок 10 – Результат «после» таблицы Car

1

SELECT * FROM Car_documentation

Data Output

Messages

Notifications

id_car_document

[PK] integer

vehicle_passport_number

smallint

vehicle_registration_certificate_number

smallint

1

1

1234

56

2

2

2109

87

3

3

5423

12

Рисунок 11 – Результат «до» таблицы Car_documentation

1

SELECT * FROM Car_documentation

2

Data Output

Messages

Notifications

	id_car_document [PK] integer	vehicle_passport_number smallint	vehicle_registration_certificate_number smallint
1	1	1234	567
2	2	2109	876

Рисунок 12 – Результат «после» таблицы Car_documentation

1

SELECT * FROM Car_technical_condition

Data Output

Messages

Notifications

id_technical_condition

[PK] integer

id_car

integer

date_of_last_maintenance

date

fuel_level

smallint

tire_type

character varying

1

1

1

2021-01-14

20

Зимняя

2

2

2

2020-07-09

40

Летняя

3

3

3

2024-12-01

90

Зимняя

4

4

1

2025-01-14

30

Зимняя

Рисунок 13 – Результат «до» таблицы Car_technical_condition

Query

Query History

1

SELECT * FROM Car_technical_condition

2

Data Output

Messages

Notifications

≡+

	id_technical_condition [PK] integer	id_car integer	date_of_last_maintenance date	fuel_level smallint	tire_type character varying
1	1	1	2021-01-14	20	Зимняя
2	2	2	2020-07-09	40	Летняя
3	4	1	2025-01-14	30	Зимняя

Рисунок 14 – Результат «после» таблицы Car_technical_condition

2.6 Итоги

- Создана база данных на основе спроектированной диаграммы;
- Реализованы SQL-запросы для добавления, изменения и удаления данных в таблицах;

- Сформирован отчет с результатами работы, включая скриншоты и листинги SQL-запросов.

ГЛАВА 3. РАБОТА С ЗАПРОСАМИ SQL

3.1 Подготовка таблиц

Чтобы заполнить все оставшиеся пустые таблицы, выполнили следующий запрос:

```
INSERT INTO Passport (id_passport, full_name, passport_series, passport_number,  
date_of_birthday, place_of_residence)
```

```
VALUES (1, 'Александр Водила', 1234, 567890, '1990-05-15', 'Moscow'),  
        (2, 'Леха Буржуй', 5678, 901234, '1985-10-20', 'Saint Petersburg'),  
        (3, 'Анатолий Колесо', 9876, 543210, '1995-03-25', 'Novosibirsk');
```

```
INSERT INTO Client (id_client, nickname, id_passport, phone_number, bonuses)
```

```
VALUES (1, 'client1', 1, 89024562198, 50),  
        (2, 'client2', 2, 82323423523, 100),  
        (3, 'client3', 3, 81233454566, 75);
```

```
INSERT INTO Bank_card (id_bank_card, id_client, bank_card_number, validity, cvv,  
status)
```

```
VALUES (1, 1, 1234567890123456, '12/25', 123, false),  
        (2, 1, 9876543210987654, '06/24', 456, true),  
        (3, 2, 1111222233334444, '09/23', 789, true);
```

```
INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price,  
start_photo, final_photo)
```

```
VALUES (1, '2024-03-01 08:00:00', '2024-03-05 08:00:00', 1, 1, 500, 'start_photo1.jpg',  
'final_photo1.jpg'),  
        (2, '2024-03-10 14:00:00', '2024-03-15 14:00:00', 2, 2, 1000, 'start_photo2.jpg',  
'final_photo2.jpg'),  
        (3, '2024-04-01 09:00:00', '2024-04-05 09:00:00', 1, 2, 1500, 'start_photo3.jpg',  
'final_photo3.jpg');
```

```

INSERT INTO Complaint (id_complaint, id_operation, complaint_date, user_comment,
status)
VALUES (1, 1, '2024-02-28 10:00:00', 'Кто-то руль погрыз', true),
      (2, 2, '2024-03-15 12:30:00', 'Машину влево ведет и ваще не едет короче отстой',
true),
      (3, 3, '2024-04-05 09:45:00', 'Машина грязная лобовое треснуто еще и ДПС в
наркологию увезли', false);

```

3.2 Запросы на однотабличную выборку данных с различными условиями

Запрос и результат запроса представлен на рисунках 15, 16, 17.

1 **SELECT** * **FROM** Car **WHERE** color = 'Зеленый';

Data Output Messages Notifications

	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character va
1	1	Alfa Romeo 164	Зеленый	p287mx74

Рисунок 15 - Выбор данных из таблицы Car

1 **SELECT** id_technical_condition **FROM** Car_technical_condition **WHERE** date_of_last_maintenance > DATE '1999-01-14';

Data Output Messages Notifications

	id_technical_condition [PK] integer
1	1
2	2
3	4

Рисунок 16 - Выбор данных из таблицы Car_technical_condition

1 **SELECT** model **FROM** Car **WHERE** id_car = 1;

Data Output Messages Notifications

	model character varying (30)
1	Alfa Romeo 164

Рисунок 17 - Выбор данных из таблицы Car

3.3 Запросы на многотабличную выборку данных с различными условиями

Выполнили запрос, представленный на рисунке 18, для выбора клиентов и их паспортных данных.

1 SELECT Client.*, Passport.* 2 FROM Client 3 INNER JOIN Passport ON Client.id_passport = Passport.id_passport;		
Data Output Messages Notifications		
	id_client integer	nickname character varying (30)
1	1	client1
2	2	client2
3	3	client3
	id_passport integer	phone_number bigint
1	1	89024562198
2	2	82323423523
3	3	81233454566
	bonuses smallint	
1	50	
2	100	
3	75	
	id_passport integer	full_name character varying
1	1	Александр Водила
2	2	Леха Буржуй
3	3	Анатолий Колесо
	passport_series smallint	passport_number integer
1	1234	567890
2	5678	901234
3	9876	543210
	date_of_birthday date	place_of_residence character varying
1	1990-05-15	Moscow
2	1985-10-20	Saint Petersburg
3	1995-03-25	Novosibirsk

Рисунок 18 - Выбор данных из таблиц Client и Passport

Пояснение к запросу на рисунке 18:

Мы выбирали данные о клиентах из таблицы Client вместе с данными их паспортов из таблицы Passport. Использовали оператор **INNER JOIN**, чтобы объединить таблицу Client с таблицей Passport. Условие Client.id_passport = Passport.id_passport указывает на то, какие строки должны быть объединены: строки, у которых значение столбца id_passport в таблице Client совпадает со значением столбца id_passport в таблице Passport.

Выполнили запрос, представленный на рисунке 19, для выбора информации об арендах с моделью и цветом автомобиля. Ход решения задачи о выборке аналогичен предыдущему.

Query Query History		
1 SELECT Arenda.*, Car.model, Car.color 2 FROM Arenda 3 INNER JOIN Car ON Arenda.id_car = Car.id_car;		
Data Output Messages Notifications		
	id_operation integer	start_date timestamp without time zone
1	1	2024-03-01 08:00:00
2	2	2024-03-10 14:00:00
3	3	2024-04-01 09:00:00
	end_date timestamp without time zone	id_car integer
1	2024-03-05 08:00:00	1
2	2024-03-15 14:00:00	2
3	2024-04-05 09:00:00	1
	id_client integer	price integer
1	1	500
2	2	1000
3	2	1500
	start_photo character varying	final_photo character varying
1	start_photo1.jpg	final_photo1.jpg
2	start_photo2.jpg	final_photo2.jpg
3	start_photo3.jpg	final_photo3.jpg
	model character varying (30)	color character varying (20)
1	Alfa Romeo 164	Зеленый
2	Toyota Carina 2	Серый
3	Alfa Romeo 164	Зеленый

Рисунок 19 - Выбор данных из таблиц Car и Arenda

3.4 Запросы с параметром

Запросы и результат запросов представлен на рисунках 20, 21.

1 **SELECT** * **FROM** Client **WHERE** nickname = 'client1';

Data Output Messages Notifications

	id_client [PK] integer	nickname character varying (30)	id_passport integer	phone_number bigint	bonuses smallint
1	1	client1	1	89024562198	50

Рисунок 20 - Выбор данных из таблицы Client

1 **SELECT** * **FROM** Arenda **WHERE** id_client = 2;

Data Output Messages Notifications

	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying
1	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg
2	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg

Рисунок 21 - Выбор данных из таблицы Arenda

3.5 Запросы на группировку данных

Выполнили следующий запрос, чтобы добавить еще одну операцию аренды:

```
INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price,  
start_photo, final_photo)  
VALUES (4, '2024-06-10 15:00:00', '2024-06-10 18:00:00', 2, 1, 1000, 'start_photo4.jpg',  
'final_photo4.jpg');
```

Выполнили запрос, представленный на рисунке 22, для подсчета количества аренд для каждого пользователя и отсортировали их по убыванию.

1	SELECT id_client, COUNT(*) AS rental_count
2	FROM Arenda
3	GROUP BY id_client
4	ORDER BY rental_count DESC;

Data Output		Messages	Notifications
	id_client integer	rental_count bigint	
1	2	2	
2	1	2	

Рисунок 22 - Подсчет количества аренд пользователей

Выполнили запрос, представленный на рисунке 23, для подсчета средней цены аренды для каждого автомобиля.

1	SELECT id_car, AVG(price) AS avg_price
2	FROM Arenda
3	GROUP BY id_car;

Data Output		Messages	Notifications
	id_car integer	avg_price numeric	
1	2	1000.0000000000000000	
2	1	1000.0000000000000000	

Рисунок 23 - Подсчёт средней цены аренда для каждого автомобиля

3.6 Запросы на создание вычисляемых полей с датой и временем

Выполнили запрос, представленный на рисунке 24, для создания поля с информацией о продолжительности аренды.

1	SELECT id_operation, start_date, end_date, end_date - start_date AS rental_duration			
2	FROM Arenda;			

Data Output	Messages	Notifications
-------------	----------	---------------

--	--	--	--	--

	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	rental_duration interval
1	1	2024-03-01 08:00:00	2024-03-05 08:00:00	4 days
2	2	2024-03-10 14:00:00	2024-03-15 14:00:00	5 days
3	3	2024-04-01 09:00:00	2024-04-05 09:00:00	4 days
4	4	2024-06-10 15:00:00	2024-06-10 18:00:00	03:00:00

Рисунок 24 - Создание поля с информацией о продолжительности аренды

3.7 Запросы на создание таблицы

Выполнили следующий запрос для создания таблицы Client_violations со штрафами и нарушениями во время аренды:

```
CREATE TABLE Client_violations (
  id_client_violations SERIAL PRIMARY KEY,
  id_operation SERIAL REFERENCES Arenda(id_operation),
  violation_description TEXT,
  fine_amount SERIAL,
  status BOOLEAN
);
```

Выполнили следующий запрос для заполнения этой таблицы данными:

```
INSERT INTO Client_violations (id_client_violations, id_operation,
violation_description, fine_amount, status)
VALUES (1, 1, 'Поцарапан бампер', 15000, true),
(2, 1, 'Превышение скорости', 5000, false);
```

Arenda - *Client_violations* - связь один ко многим, так как в течении одной аренды можно получить несколько штрафов.

Теперь наша база данных выглядит следующим образом, представленным на рисунке 25.

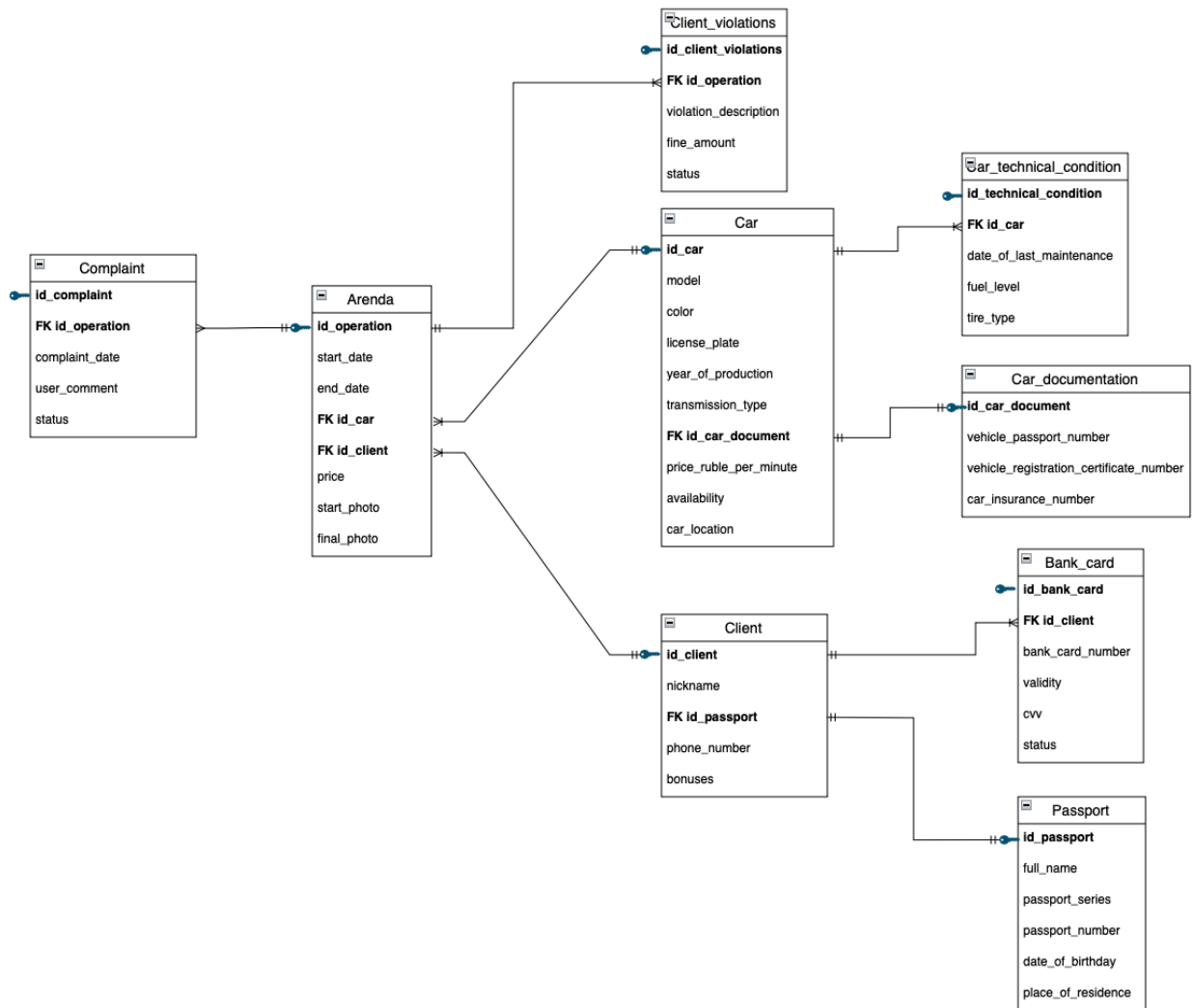


Рисунок 25 - Обновленная архитектура базы данных

3.8 Параметрический запрос на добавление одной записи

Выполнили следующие запросы:

```
INSERT INTO Car_documentation(id_car_document, vehicle_passport_number,
vehicle_registration_certificate_number, car_insurance_number) VALUES (3, 5423,
1265, 1238);
```

```
PREPARE insert_car (INT, VARCHAR(30), VARCHAR(20), VARCHAR(15), INT,
VARCHAR(20), INT, INT, BOOLEAN, VARCHAR(50)) AS
```



```
INSERT INTO Car (id_car, model, color, license_plate, year_of_production,  
transmission_type, id_car_document, price_ruble_per_minute, availability,  
car_location)
```

```
VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10);
```

```
EXECUTE insert_car(3, 'ВАЗ 2108', 'Красный', 'в442pp174', 1990, 'МКПП', 3, 10,  
true, 'г.Петергоф ул.Ботаническая д.70 к.4');
```

```
INSERT INTO Car_technical_condition(id_technical_condition, id_car,  
date_of_last_maintenance, fuel_level, tire_type) VALUES (3, 3, '2024-12-01', 90,  
'Зимняя');
```

3.9 Параметрический запрос на добавление нескольких записей

Выполнили следующие запросы:

```
DEALLOCATE insert_car; -- Удалить существующее подготовленное  
выражение
```

```
INSERT INTO Car_documentation(id_car_document, vehicle_passport_number,  
vehicle_registration_certificate_number, car_insurance_number) VALUES  
(4, 2433, 3265, 5238),  
(5, 1433, 7265, 5138);
```

```
PREPARE insert_car(INT, VARCHAR(30), VARCHAR(20), VARCHAR(15), INT,  
VARCHAR(20), INT, INT, BOOLEAN, VARCHAR(50)) AS
```

```
INSERT INTO Car (id_car, model, color, license_plate, year_of_production,  
transmission_type, id_car_document, price_ruble_per_minute, availability,  
car_location)
```

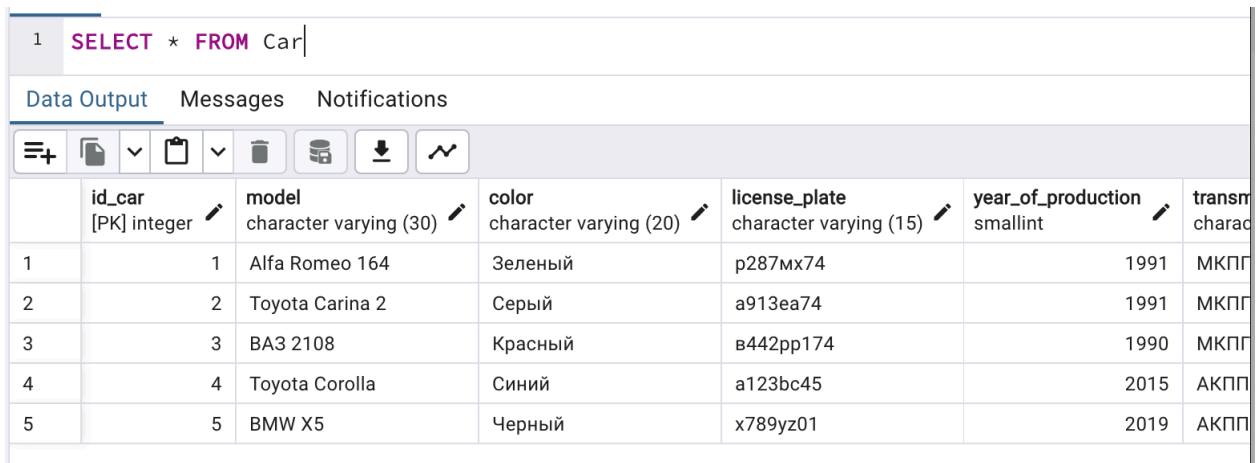
```
VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10);
```

```
EXECUTE insert_car(4, 'Toyota Corolla', 'Синий', 'a123bc45', 2015, 'АКПП', 2,  
15, true, 'г.Санкт-Петербург ул.Невский проспект, д.1');
```

```
EXECUTE insert_car(5, 'BMW X5', 'Черный', 'x789yz01', 2019, 'АКПП', 3, 20,  
false, 'г.Москва ул.Тверская, д.15');
```

```
INSERT INTO Car_technical_condition(id_technical_condition, id_car,
date_of_last_maintenance, fuel_level, tire_type) VALUES
(5, 4, '2024-11-01', 90, 'Зимняя'),
(6, 5, '2024-12-01', 60, 'Зимняя');
```

Результат запросов представлен на рисунке 26.



	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying (15)	year_of_production smallint	transm charac
1	1	Alfa Romeo 164	Зеленый	p287mx74	1991	МКПП
2	2	Toyota Carina 2	Серый	a913ea74	1991	МКПП
3	3	ВАЗ 2108	Красный	v442pp174	1990	МКПП
4	4	Toyota Corolla	Синий	a123bc45	2015	АКПП
5	5	BMW X5	Черный	x789yz01	2019	АКПП

Рисунок 26 - Параметрический запрос на добавление нескольких записей в таблицу Car

3.10 Параметрический запрос на удаление записей

Для удаления нескольких записей выполнили следующий запрос:

```
DELETE FROM Car_technical_condition WHERE id_technical_condition IN (5, 6);
```

```
PREPARE delete_car(INT) AS
```

```
DELETE FROM Car WHERE id_car = $1;
```

```
EXECUTE delete_car(4);
```

```
EXECUTE delete_car(5);
```

```
DELETE FROM Car_documentation WHERE id_car_document IN (4, 5);
```

Результат “до” представлен на рисунках 27, 28, 29.

1

SELECT * FROM Car_technical_condition

Data Output

Messages

Notifications

id_technical_condition

[PK] integer

id_car

integer

date_of_last_maintenance

date

fuel_level

smallint

tire_type

character varying

1

1

2021-01-14

20

Зимняя

2

2

2020-07-09

40

Летняя

3

4

2025-01-14

30

Зимняя

4

3

2024-12-01

90

Зимняя

5

5

2024-11-01

90

Зимняя

6

6

2024-12-01

60

Зимняя

Рисунок 27 - Результат «до» запроса таблицы Car_technical_condition

1SELECT * FROM Car

Data Output

Messages

Notifications

	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying (15)	year_of_production smallint	tran cha
1	1	Alfa Romeo 164	Зеленый	p287mx74	1991	МК
2	2	Toyota Carina 2	Серый	a913ea74	1991	МК
3	3	BA3 2108	Красный	в442pp174	1990	МК
4	4	Toyota Corolla	Синий	a123bc45	2015	АК
5	5	BMW X5	Черный	x789yz01	2019	АК

Рисунок 28 - Результат «до» запроса таблицы Car

1SELECT * FROM Car_documentation

Data Output

Messages

Notifications

	id_car_document [PK] integer	vehicle_passport_number smallint	vehicle_registration_certificate_number smallint	car_insurance_number smallint
1	1	1234	5678	9012
2	2	2109	8765	4321
3	3	5423	1265	1238
4	4	2433	3265	5238
5	5	1433	7265	5138

Рисунок 29 - Результат «до» запроса таблицы Car_documentation

Результат “после” представлен на рисунках 30, 31, 32.

1 SELECT * FROM Car_technical_condition						
Data Output Messages Notifications						
<div> <div>≡</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>						
	id_technical_condition [PK] integer	id_car integer	date_of_last_maintenance date	fuel_level smallint	tire_type character varying	
1	1	1	2021-01-14	20	Зимняя	
2	2	2	2020-07-09	40	Летняя	
3	4	1	2025-01-14	30	Зимняя	
4	3	3	2024-12-01	90	Зимняя	

Рисунок 30 - Результат «после» запроса таблицы Car_technical_condition

1 SELECT * FROM Car					
Data Output Messages Notifications					
<div> <div>≡</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>					
	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying (15)	year_of_production smallint
1	1	Alfa Romeo 164	Зеленый	p287mx74	
2	2	Toyota Carina 2	Серый	a913ea74	
3	3	BA3 2108	Красный	v442pp174	

Рисунок 31 - Результат «после» запроса таблицы Car

1 SELECT * FROM Car_documentation				
Data Output Messages Notifications				
<div> <div>≡</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>				
	id_car_document [PK] integer	vehicle_passport_number smallint	vehicle_registration_certificate_number smallint	car_insurance_number smallint
1	1	1234	5678	9012
2	2	2109	8765	4321
3	3	5423	1265	1238

Рисунок 32 - Результат «после» запроса таблицы Car_documentation

3.11 Параметрический запрос на обновление полей записей

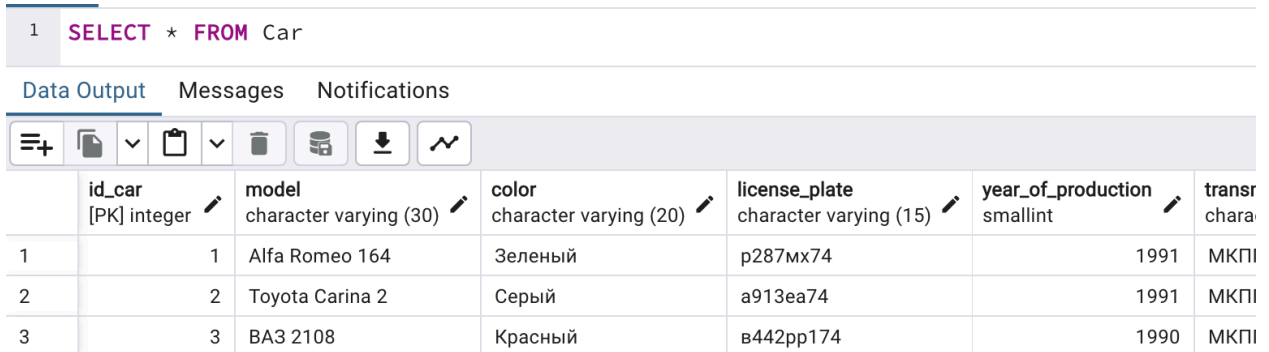
Для обновления нескольких записей выполнили следующий запрос:

```
PREPARE update_car(INT, VARCHAR(30), VARCHAR(20), INT) AS
UPDATE Car SET model = $2, color = $3, year_of_production = $4 WHERE
id_car = $1;
```

EXECUTE update_car(2, 'Mercedes-Benz w124', 'Синий', 1990);

EXECUTE update_car(3, 'BA3 21083', 'Бежевый', 1991);

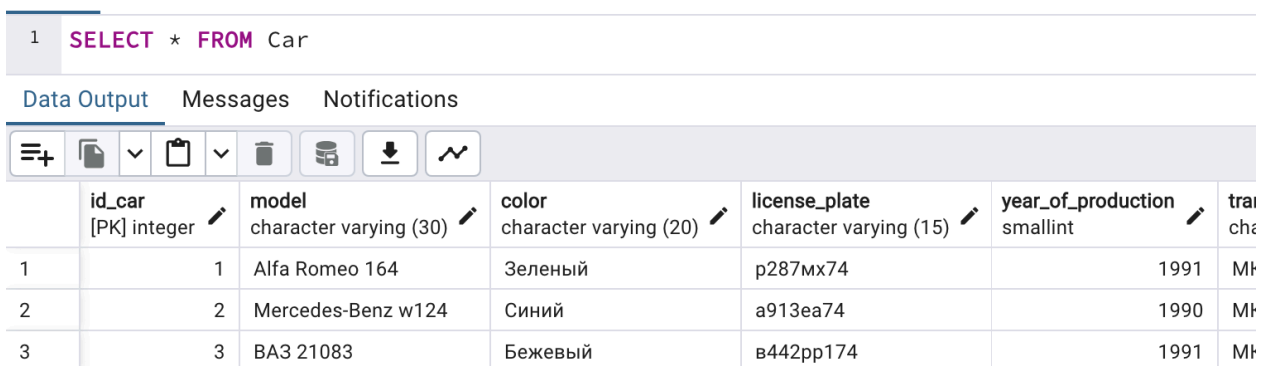
Результат “до” представлен на рисунке 33.



1 SELECT * FROM Car						
Data Output Messages Notifications						
	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying (15)	year_of_production smallint	trans chara
1	1	Alfa Romeo 164	Зеленый	p287mx74	1991	МКП
2	2	Toyota Carina 2	Серый	a913ea74	1991	МКП
3	3	BA3 2108	Красный	в442pp174	1990	МКП

Рисунок 33 - Результат «до» запроса таблицы Car

Результат “после” представлен на рисунке 34.



1 SELECT * FROM Car						
Data Output Messages Notifications						
	id_car [PK] integer	model character varying (30)	color character varying (20)	license_plate character varying (15)	year_of_production smallint	trans chara
1	1	Alfa Romeo 164	Зеленый	p287mx74	1991	М
2	2	Mercedes-Benz w124	Синий	a913ea74	1990	М
3	3	BA3 21083	Бежевый	в442pp174	1991	М

Рисунок 34 - Результат после запроса таблицы Car

3.12 Вложенные запросы

Выполнили следующий запрос для добавления дополнительной операции:

INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price, start_photo, final_photo)

VALUES (5, '2024-06-10 15:00:00', '2024-06-10 18:00:00', 2, 1, 500, 'start_photo5.jpg', 'final_photo5.jpg');

Выполнили запрос, представленный на рисунке 35, для выбора клиентов, у которых количество аренд больше среднего.

```
1 SELECT * FROM Client
2 WHERE id_client IN (
3     SELECT id_client FROM Arenda
4     GROUP BY id_client
5     HAVING COUNT(*) > (
6         SELECT AVG(rental_count) FROM (
7             SELECT id_client, COUNT(*) AS rental_count FROM Arenda GROUP BY id_client
8         ) AS avg_rentals
9     )
10 );
```

Data Output Messages Notifications

	id_client [PK] integer	nickname character varying (30)	id_passport integer	phone_number bigint	bonuses smallint
1	1	client1	1	89024562198	50

Рисунок 35 - Выбор клиентов, у которых количество аренд больше среднего

Пояснения к запросу:

1. Внешний запрос:

Внешний запрос выбирает всех клиентов из таблицы Client, удовлетворяющих определенному условию:

```
SELECT * FROM Client
WHERE id_client IN (
    -- Вложенный запрос
);
```

2. Вложенный запрос:

В этом вложенном запросе сначала происходит группировка записей из таблицы Arenda по id_client, затем с помощью выражения COUNT(*) подсчитывается количество аренд для каждого клиента. Затем с помощью

выражения **HAVING** выбираются только те клиенты, у которых количество аренд превышает среднее значение.

```
SELECT id_client FROM Arenda
```

```
GROUP BY id_client
```

```
HAVING COUNT(*) > (
```

```
-- Вложенный запрос
```

```
)
```

3. Вложенный подзапрос:

В этом вложенном подзапросе мы сначала реализуем внутренний запрос **SELECT id_client, COUNT(*) AS rental_count FROM Arenda GROUP BY id_client**, который подсчитывает количество аренд для каждого клиента и группирует результаты по **id_client**. Затем внешний запрос **SELECT AVG(rental_count) FROM (...) AS avg_rentals** вычисляет среднее значение количества аренд для всех клиентов:

```
SELECT AVG(rental_count) FROM (
```

```
    SELECT id_client, COUNT(*) AS rental_count FROM Arenda GROUP BY  
id_client
```

```
) AS avg_rentals
```

3.13 Итоги

- Написаны запросы на выборку данных с различными условиями;
- Реализованы запросы с параметрами и запросы на группировку данных;
- Созданы запросы для вычисляемых полей с датой и временем;
- Реализован запрос на создание таблицы;
- Разработаны параметрические запросы на добавление одной и нескольких записей;
- Написаны запросы на удаление и обновление данных, а также вложенные запросы для сложных операций.

ГЛАВА 4. ПРОДВИНУТЫЕ ЗАПРОСЫ

4.1 Вложенные запросы

Выполнили запрос, представленный на рисунке 36, для выборки машин с наименьшим количеством топлива.

```
1 SELECT c.id_car, c.model, t.fuel_level
2 FROM Car c
3 JOIN Car_technical_condition t ON c.id_car = t.id_car
4 WHERE t.fuel_level = (
5     SELECT MIN(fuel_level)
6     FROM Car_technical_condition
7 );
8
```

Data Output Messages Notifications

	id_car integer	model character varying (30)	fuel_level smallint
1	1	Alfa Romeo 164	20

Рисунок 36 - Выборка машин с наименьшим количеством топлива

Пояснения к запросу на рисунке 36:

Этот запрос использует вложенный **SELECT** для выбора минимального значения уровня топлива в таблице Car_technical_condition, а затем основной **SELECT** выбирает id_car и model из таблицы Car с этим минимальным уровнем топлива [4].

Выполнили запрос, представленный на рисунке 37, для проверки.

```
1 select * from car_technical_condition
```

Data Output Messages Notifications

	id_technical_condition [PK] integer	id_car integer	date_of_last_maintenance date	fuel_level smallint	tire_type character varying
1	1	1	2021-01-14	20	Зимняя
2	2	2	2020-07-09	40	Летняя
3	4	1	2025-01-14	30	Зимняя
4	3	3	2024-12-01	90	Зимняя

Рисунок 37 - Данные таблицы Car_technical_condition

Выполнили следующий запрос, представленный на рисунке 38, для выборки клиентов, у которых есть активные жалобы:

```
1 SELECT nickname
2 FROM Client
3 WHERE id_client IN (
4     SELECT DISTINCT a.id_client
5     FROM Arenda a
6     JOIN Complaint cm ON a.id_operation = cm.id_operation
7     WHERE cm.status = TRUE
8 );
9
```

Data Output Messages Notifications

	nickname character varying (30) 🔒
1	client2
2	client1

Рисунок 38 - Выборка клиентов с активными жалобами

Пояснение к запросу, представленному выше:

Этот запрос использует вложенный SELECT для выбора уникальных (для этого используем **DISTINCT**) идентификаторов клиентов (id_client), у которых есть активные жалобы (status = TRUE), а затем основной SELECT выбирает клиентов с этими идентификаторами.

Выполнили запрос, представленный на рисунке 39, для проверки.

```
1 SELECT a.id_client, c.status
2 FROM Complaint c
3 JOIN Arenda a ON c.id_operation = a.id_operation;
4
```

Data Output Messages Notifications

	id_client integer 🔒	status boolean 🔒
1	1	true
2	2	true
3	2	false

Рисунок 39 - Пользователи и статусы их жалоб

Выполнили запрос, представленный на рисунке 40, для выборки машины, которая чаще всего использовалась в аренде.

```
1 SELECT id_car, model
2 FROM Car
3 WHERE id_car IN (
4     SELECT id_car
5     FROM Arenda
6     GROUP BY id_car
7     ORDER BY COUNT(*) DESC
8     LIMIT 1
9 );
```

Data Output Messages Notifications

	id_car [PK] integer	model character varying (30)
1	2	Mercedes-Benz w124

Рисунок 40 - Выборка машины, которая чаще всего использовалась в аренде

Пояснения к запросу, представленному выше:

Во вложенном запросе мы сначала группируем аренды по `id_car` и подсчитываем количество аренд для каждой машины (`COUNT(*)`). Затем мы сортируем результаты по убыванию количества аренд (`COUNT(*) DESC`) и выбираем только первую запись (машина с наибольшим количеством аренд) с помощью `LIMIT 1`.

В основном запросе мы выбираем идентификатор машины `id_car` и `model` только для той машины, которая была выбрана в результате вложенного запроса.

Выполнили запрос, представленный на рисунке 41, для проверки.

1select * from arenda

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer
1	1	2024-03-01 08:00:00	2024-03-05 08:00:00	1
2	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2
3	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1
4	4	2024-06-10 15:00:00	2024-06-10 18:00:00	2
5	5	2024-06-10 15:00:00	2024-06-10 18:00:00	2

Рисунок 41 - Данные таблицы Arenda

4.2 Индексация столбцов

Запрос и результат запроса представлен на рисунке 42.

1

CREATE INDEX idx_car ON Car (model, license_plate, transmission_type, price_ruble_per_minute, availability, car_location);

2

CREATE INDEX idx_client ON Client (nickname, bonuses);

3

CREATE INDEX idx_passport ON Passport (full_name);

4

CREATE INDEX idx_car_technical_condition ON Car_technical_condition (fuel_level);

5

6

SELECT *

7

FROM Passport INDEX(idx_passport)

8

WHERE full_name = 'Александр Водила';

Data Output

Messages

Notifications

	idx_passport integer	full_name character varying	passport_series smallint	passport_number integer	date_of_birthday date	place_of_residence character varying
1	1	Александр Водила	1234	567890	1990-05-15	Moscow

Рисунок 42 - Создание индексов

Пояснения к запросу, представленному на рисунке 41:

В PostgreSQL индексы - это структуры данных, которые ускоряют процесс поиска и доступа к данным в таблицах базы данных. Индексы создаются с целью оптимизации производительности запросов SELECT, когда таблица содержит большое количество данных. Обычно создаются индексы на конкретные столбцы или комбинации столбцов, которые часто используются в запросах SELECT для ускорения выполнения этих запросов.

Это позволяет СУБД быстро находить соответствующие строки в таблице, используя индекс, вместо полного сканирования таблицы. Рекомендуется избегать создания индексов на всю таблицу, если только это действительно необходимо для вашего конкретного случая.

В PostgreSQL, как и в большинстве СУБД, индекс можно создать на всю таблицу, но это обычно не имеет смысла. Индекс на всю таблицу (полный индекс) создает отображение каждой строки таблицы в структуру индекса, что может занимать большой объем памяти и замедлять операции вставки, обновления и удаления данных. Создание индексов может увеличить немного время выполнения операций добавления, обновления и удаления данных, так как PostgreSQL должен поддерживать актуальность индексов в соответствии с изменениями данных. Также не стоит создавать слишком много индексов, так как это может привести к увеличению использования дискового пространства и ухудшению производительности вставки и обновления данных.

4.3 Ограничение первичных и внешних ключей

Выполнили следующий запрос для ограничения первичных ключей:

```
ALTER TABLE Car_documentation
```

```
ADD CONSTRAINT pk_car_documentation PRIMARY KEY (id_car_document);
```

```
ALTER TABLE Car
```

```
ADD CONSTRAINT pk_car PRIMARY KEY (id_car);
```

```
ALTER TABLE Car_technical_condition
```

```
ADD CONSTRAINT pk_car_technical_condition PRIMARY KEY  
(id_technical_condition);
```

```
ALTER TABLE Passport
```

```
ADD CONSTRAINT pk_passport PRIMARY KEY (id_passport);
```

ALTER TABLE Client

ADD CONSTRAINT pk_client **PRIMARY KEY** (id_client);

ALTER TABLE Bank_card

ADD CONSTRAINT pk_bank_card **PRIMARY KEY** (id_bank_card);

ALTER TABLE Arenda

ADD CONSTRAINT pk_arenda **PRIMARY KEY** (id_operation);

ALTER TABLE Complaint

ADD CONSTRAINT pk_complaint **PRIMARY KEY** (id_complaint);

ALTER TABLE Client_violations

ADD CONSTRAINT pk_client_violations **PRIMARY KEY** (id_client_violations);

Выполнили следующий запрос для ограничения внешних ключей:

ALTER TABLE Car

ADD CONSTRAINT fk_car_car_documentation **FOREIGN KEY** (id_car_document)

REFERENCES Car_documentation(id_car_document);

ALTER TABLE Car_technical_condition

ADD CONSTRAINT fk_car_technical_condition_car **FOREIGN KEY** (id_car)

REFERENCES Car(id_car);

ALTER TABLE Client

ADD CONSTRAINT fk_client_passport **FOREIGN KEY** (id_passport)

REFERENCES Passport(id_passport);

ALTER TABLE Bank_card

ADD CONSTRAINT fk_bank_card_client **FOREIGN KEY** (id_client)

REFERENCES Client(id_client);

ALTER TABLE Arenda

ADD CONSTRAINT fk_arenda_car **FOREIGN KEY** (id_car) **REFERENCES**

Car(id_car),

ADD CONSTRAINT fk_arenda_client **FOREIGN KEY** (id_client) **REFERENCES**

Client(id_client);

ALTER TABLE Complaint

ADD CONSTRAINT fk_complaint_arenda **FOREIGN KEY** (id_operation)

REFERENCES Arenda(id_operation);

ALTER TABLE Client_violations

ADD CONSTRAINT fk_client_violations_arenda **FOREIGN KEY** (id_operation)

REFERENCES Arenda(id_operation);

4.4 Итоги

- Реализованы вложенные запросы;
- Произведена индексация столбцов для оптимизации запросов;
- Написаны инструкции по добавлению ограничений первичных и внешних ключей для обеспечения целостности данных.

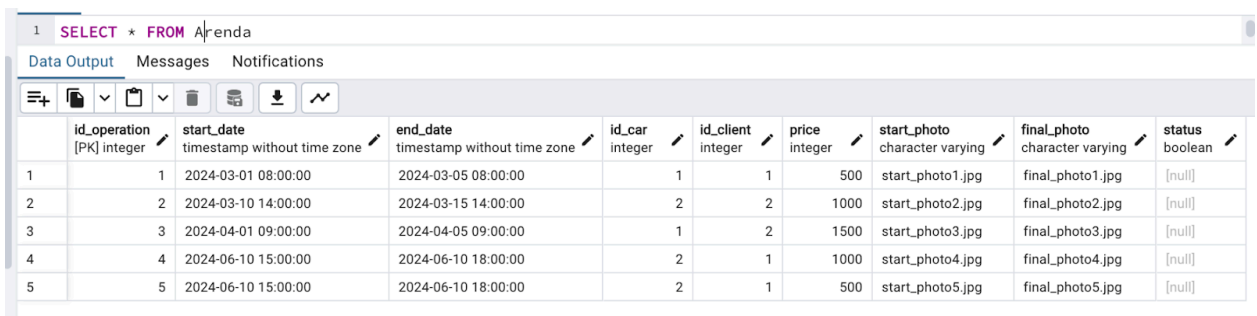
ГЛАВА 5. Функции и триггеры

5.1 Триггер для установки статуса операции

Пусть столбец “status” в таблице “Arenda” отвечает за статус операции аренды (например, “true” - аренда завершена, “false” - в процессе), выполнили следующий запрос для создания столбца:

```
ALTER TABLE Arenda ADD COLUMN status BOOLEAN;
```

Результат запроса представлен на рисунке 43.



1	SELECT * FROM Arenda									
Data Output Messages Notifications										
	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying	status boolean	
1	1	2024-03-01 08:00:00	2024-03-05 08:00:00	1	1	500	start_photo1.jpg	final_photo1.jpg	[null]	
2	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg	[null]	
3	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg	[null]	
4	4	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	[null]	
5	5	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	500	start_photo5.jpg	final_photo5.jpg	[null]	

Рисунок 43 - Добавление столбца status в таблицу Arenda

Выполнили следующий запрос для создания триггера для установки статуса операции:

```
CREATE OR REPLACE FUNCTION set_operation_status()
```

```
RETURNS TRIGGER AS $$
```

```
BEGIN
```

```
-- Проверяем даты, чтобы определить статус
```

```
IF NEW.end_date < CURRENT_TIMESTAMP THEN
```

```
    NEW.status = true; -- Если дата окончания прошла, ставим статус
```

```
"готово"
```

```
ELSE
```

```
    NEW.status = false; -- Иначе ставим статус "в процессе"
```

```
END IF;
```

```
RETURN NEW;
```


END;

\$\$ LANGUAGE plpgsql;

CREATE TRIGGER operation_status_trigger

BEFORE INSERT OR UPDATE ON Arenda

FOR EACH ROW

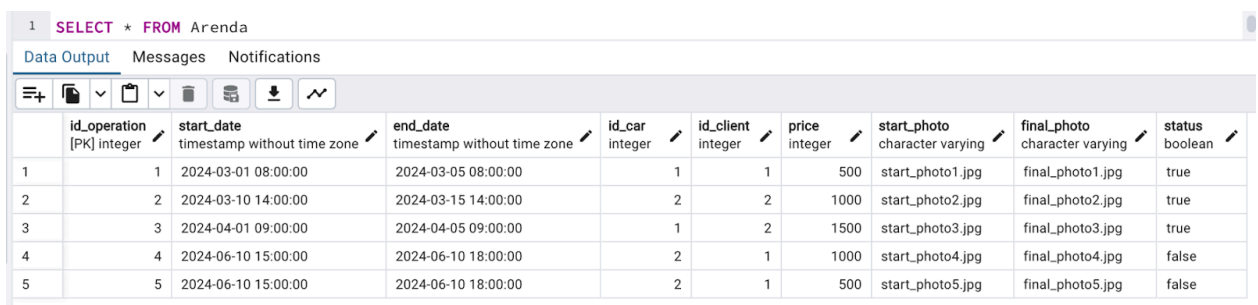
EXECUTE FUNCTION set_operation_status();

Пусть все операции, которые были до создания колонки, будут завершены, выполнили следующий запрос:

UPDATE Arenda

SET status = true;

Заметили, что PostgreSQL автоматически заполняет колонку status, в зависимости от end_date, выполнили запрос, представленный на рисунке 44.



	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying	status boolean
1	1	2024-03-01 08:00:00	2024-03-05 08:00:00	1	1	500	start_photo1.jpg	final_photo1.jpg	true
2	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg	true
3	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg	true
4	4	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	false
5	5	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	500	start_photo5.jpg	final_photo5.jpg	false

Рисунок 44 - Данные таблицы Arenda

Выполнили следующий запрос для добавления нескольких операций:

INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price, start_photo, final_photo)

VALUES (6, '2024-06-10 15:00:00', '2024-06-10 18:00:00', 2, 1, 1000, 'start_photo4.jpg', 'final_photo4.jpg'),

(7, '2024-06-10 15:00:00', '2024-01-01 18:00:00', 2, 1, 1000, 'start_photo4.jpg', 'final_photo4.jpg');

Запрос и результат запроса представлен на рисунке 45.

1 **SELECT** * **FROM** Arenda

Data Output Messages Notifications

	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying	status boolean
1	1	2024-03-01 08:00:00	2024-03-05 08:00:00	1	1	500	start_photo1.jpg	final_photo1.jpg	true
2	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg	true
3	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg	true
4	4	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	false
5	5	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	500	start_photo5.jpg	final_photo5.jpg	false
6	6	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	false
7	7	2024-06-10 15:00:00	2024-01-01 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	true

Рисунок 45 - Данные таблицы Arenda

5.2 Триггер для обновления данных при добавлении новых строк или изменении количества

Выполнили следующий запрос:

ALTER TABLE Client

ADD COLUMN number_of_rentals **INTEGER DEFAULT 0**;

Результат запроса представлен на рисунке 46.

1 **SELECT** * **FROM** Client

Data Output Messages Notifications

	id_client [PK] integer	nickname character varying (30)	id_passport integer	phone_number bigint	bonuses smallint	number_of_rentals integer
1	1	client1	1	89024562198	50	0
2	2	client2	2	82323423523	100	0
3	3	client3	3	81233454566	75	0

Рисунок 46 - Данные таблицы Client

Выполнили следующий запрос для заполнения созданной колонки количеством операций каждого пользователя при помощи таблицы Arenda:

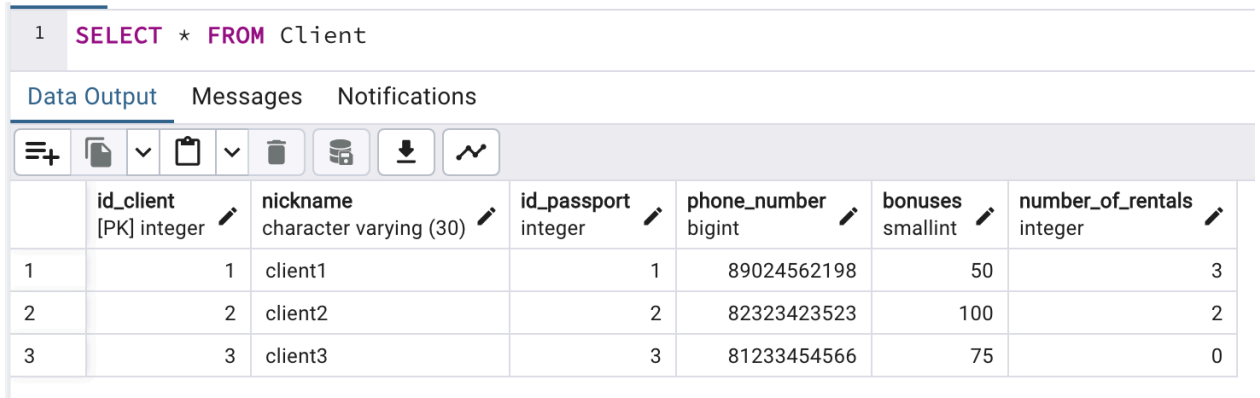
UPDATE Client

```

SET number_of_rentals = (
    SELECT COUNT(*)
    FROM Arenda
    WHERE Arenda.id_client = Client.id_client
);

```

Результат запроса представлен на рисунке 47.



	id_client [PK] integer	nickname character varying (30)	id_passport integer	phone_number bigint	bonuses smallint	number_of_rentals integer
1	1	client1	1	89024562198	50	3
2	2	client2	2	82323423523	100	2
3	3	client3	3	81233454566	75	0

Рисунок 47 - Данные таблицы Client

Выполнили следующий запрос для создания триггера:

```

CREATE OR REPLACE FUNCTION update_rental_count()
RETURNS TRIGGER AS $$
BEGIN
    -- Обновляем количество аренд у клиента, связанного с данной операцией
    UPDATE Client
    SET number_of_rentals = (
        SELECT COUNT(*)
        FROM Arenda
        WHERE id_client = NEW.id_client
    )
    WHERE id_client = NEW.id_client;

    RETURN NEW;
END;

```

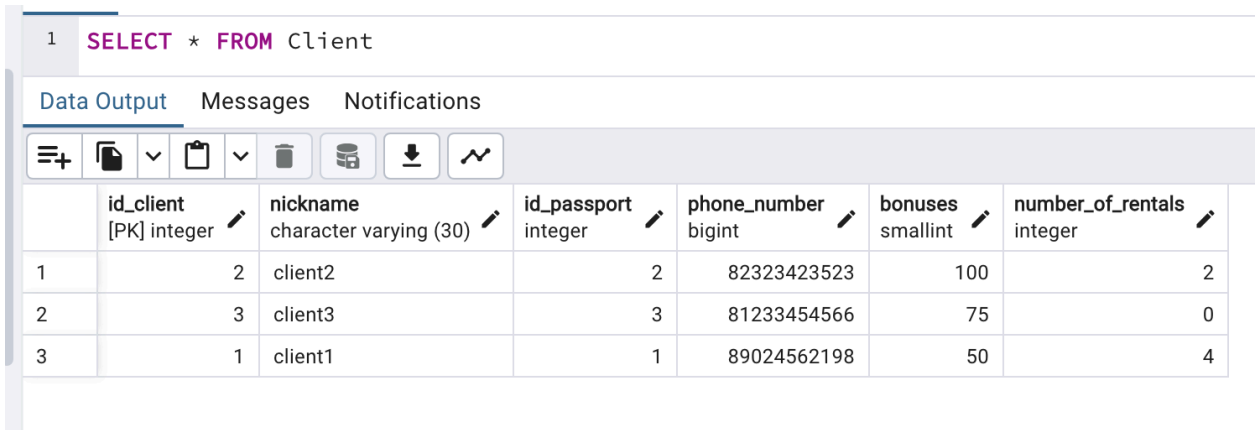
```
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER rental_update_trigger  
AFTER INSERT OR DELETE ON Arenda  
FOR EACH ROW  
EXECUTE FUNCTION update_rental_count();
```

Выполнили следующий запрос для добавления строки с операцией пользователя с `id_client = 1` в таблицу `Arenda`:

```
INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price,  
start_photo, final_photo)  
VALUES (8, '2024-06-10 15:00:00', '2024-06-10 18:00:00', 2, 1, 1000, 'start_photo8.jpg',  
'final_photo8.jpg');
```

Посмотрели на рисунок 48 и заметили, что таблица `Client` обновилась автоматически.



	id_client [PK] integer	nickname character varying (30)	id_passport integer	phone_number bigint	bonuses smallint	number_of_rentals integer
1	2	client2	2	82323423523	100	2
2	3	client3	3	81233454566	75	0
3	1	client1	1	89024562198	50	4

Рисунок 48 - Данные таблицы `Client`

5.3 Триггер для проверки ограничения на количество

Пусть `client1` завершил все свои аренды. Для этого выполнили следующий запрос:

```
UPDATE Arenda
```

SET end_date = '2024-01-10 18:00:00'

WHERE id_client = 1;

Результат “до” представлен на рисунке 49.

1

SELECT * FROM Arenda

Data Output

Messages

Notifications

	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying	status boolean
1	1	2024-03-01 08:00:00	2024-03-05 08:00:00	1	1	500	start_photo1.jpg	final_photo1.jpg	true
2	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg	true
3	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg	true
4	4	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	false
5	5	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	500	start_photo5.jpg	final_photo5.jpg	false
6	8	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo8.jpg	final_photo8.jpg	false

Рисунок 49 - Результат «до» запроса в таблице Arenda

Результат «после» представлен на рисунке 50.

1 SELECT * FROM Arenda										
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><div></div></div></div></div>										
	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying	status boolean	
1	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg	true	
2	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg	true	
3	1	2024-03-01 08:00:00	2024-01-10 18:00:00	1	1	500	start_photo1.jpg	final_photo1.jpg	true	
4	4	2024-06-10 15:00:00	2024-01-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	true	
5	5	2024-06-10 15:00:00	2024-01-10 18:00:00	2	1	500	start_photo5.jpg	final_photo5.jpg	true	
6	8	2024-06-10 15:00:00	2024-01-10 18:00:00	2	1	1000	start_photo8.jpg	final_photo8.jpg	true	

Рисунок 50 - Результат «после» запроса в таблице Arenda

Выполнили следующий запрос для создания триггера:

CREATE OR REPLACE FUNCTION check_active_rental()

RETURNS TRIGGER AS \$\$

DECLARE

active_rental_count **INTEGER**;

BEGIN

-- Проверяем количество активных аренд для данного клиента

SELECT COUNT(*)

INTO active_rental_count

FROM Arenda

```

WHERE id_client = NEW.id_client
AND status = false; -- Проверяем только незавершенные (в процессе) аренды

-- Если у клиента уже есть активная аренда, выбрасываем ошибку
IF active_rental_count > 0 THEN
    RAISE EXCEPTION 'Client already has an active rental. Cannot start another one.';
END IF;

-- Если проверка пройдена успешно, возвращаем NEW (разрешаем вставку)
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Создаем триггер BEFORE INSERT для таблицы Arenda
CREATE TRIGGER prevent_multiple_active_rentals
BEFORE INSERT ON Arenda
FOR EACH ROW
EXECUTE FUNCTION check_active_rental();

```

Выполнили следующий запрос для добавления еще одной аренды клиенту client1:

```

INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price,
start_photo, final_photo)
VALUES (9, '2024-06-10 15:00:00', '2024-06-10 18:00:00', 2, 1, 1000, 'start_photo9.jpg',
'final_photo9.jpg');

```

Результат запроса представлен на рисунке 51.

1 SELECT * FROM Arenda										
Data Output Messages Notifications										
<div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>										
	id_operation [PK] integer	start_date timestamp without time zone	end_date timestamp without time zone	id_car integer	id_client integer	price integer	start_photo character varying	final_photo character varying	status boolean	
1	2	2024-03-10 14:00:00	2024-03-15 14:00:00	2	2	1000	start_photo2.jpg	final_photo2.jpg	true	
2	3	2024-04-01 09:00:00	2024-04-05 09:00:00	1	2	1500	start_photo3.jpg	final_photo3.jpg	true	
3	1	2024-03-01 08:00:00	2024-01-10 18:00:00	1	1	500	start_photo1.jpg	final_photo1.jpg	true	
4	4	2024-06-10 15:00:00	2024-01-10 18:00:00	2	1	1000	start_photo4.jpg	final_photo4.jpg	true	
5	5	2024-06-10 15:00:00	2024-01-10 18:00:00	2	1	500	start_photo5.jpg	final_photo5.jpg	true	
6	8	2024-06-10 15:00:00	2024-01-10 18:00:00	2	1	1000	start_photo8.jpg	final_photo8.jpg	true	
7	9	2024-06-10 15:00:00	2024-06-10 18:00:00	2	1	1000	start_photo9.jpg	final_photo9.jpg	false	

Рисунок 51 - Данные таблицы Arenda

Попробовали выполнить запрос на добавление еще одной активной аренды пользователю client1, этот запрос представлен на рисунке 52. Получили ошибку, потому что у client1 уже есть незаконченная аренда.

1 INSERT INTO Arenda (id_operation, start_date, end_date, id_car, id_client, price, start_photo, final_photo)										
2 VALUES (10, '2024-06-10 15:00:00', '2024-06-10 18:00:00', 2, 1, 1000, 'start_photo10.jpg', 'final_photo10.jpg');										
Data Output Messages Notifications										
ERROR: Client already has an active rental. Cannot start another one. CONTEXT: PL/pgSQL function check_active_rental() line 14 at RAISE SQL state: P0001										

Рисунок 52 - Работа триггера

5.4 Реализация вычислительной функции

Выполнили следующий запрос для создания функции для вычисления стоимости аренды автомобиля:

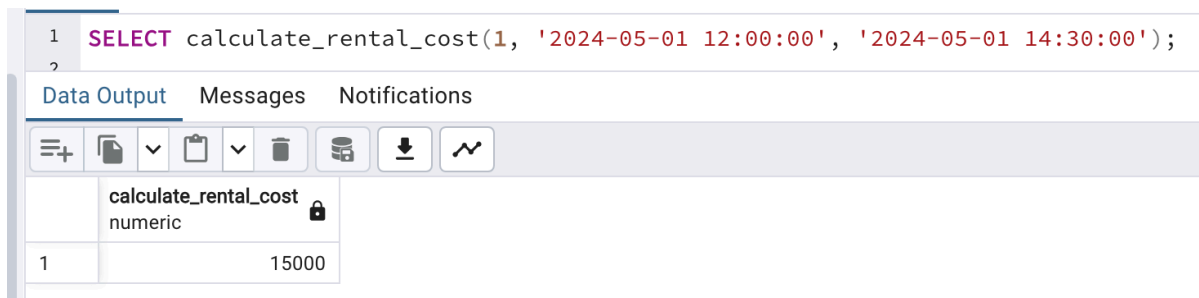
```
CREATE OR REPLACE FUNCTION calculate_rental_cost(
    car_id_param INT,
    start_date_param TIMESTAMP,
    end_date_param TIMESTAMP
)
RETURNS NUMERIC AS $$
DECLARE
    car_price_per_minute NUMERIC;
    rental_duration_minutes INT;
```

```

rental_cost NUMERIC;
BEGIN
  SELECT price_ruble_per_minute
  INTO car_price_per_minute
  FROM Car
  WHERE id_car = car_id_param;
  rental_duration_minutes := EXTRACT(EPOCH FROM (end_date_param -
start_date_param)) / 60;
  rental_cost := rental_duration_minutes * car_price_per_minute;
  RETURN rental_cost;
END;
$$ LANGUAGE plpgsql;

```

Пример вызова функции для вычисления стоимости аренды представлен на рисунке 53.



The screenshot shows a database client interface with a SQL editor at the top containing the query: `SELECT calculate_rental_cost(1, '2024-05-01 12:00:00', '2024-05-01 14:30:00');`. Below the editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, displaying a table with one row of results. The table has two columns: the first column contains the value '1' and the second column contains the value '15000'. Above the table, the function name 'calculate_rental_cost' and its return type 'numeric' are displayed.

calculate_rental_cost numeric	
1	15000

Рисунок 53 - Работа функции calculate_rental_cost

5.5 Итоги

- Разработаны триггеры для автоматического управления статусом данных в базе;
- Написаны триггеры для обновления данных при добавлении новых строк или изменении количества;
- Реализован триггер, проверяющий численное ограничение;
- Разработана вычислительная функция, принимающая аргументы и

возвращающая результат.

ЗАКЛЮЧЕНИЕ

Разработанный проект базы данных для системы каршеринга успешно реализован и все требования по этапам задания выполнены. Разработанная при помощи PostgreSQL база данных содержит необходимые таблицы, поля, связи, типы данных, а также реализует разнообразные запросы и функциональные требования.

Этот проект демонстрирует практическое применение знаний по базам данных и SQL на примере реальной системы каршеринга, что позволяет лучше понять принципы проектирования и управления данными в контексте реальных бизнес-процессов.

СПИСОК ЛИТЕРАТУРЫ

1. Петров П.П. Технологии реализации каршеринга с использованием PostgreSQL // Сборник научных трудов "Современные информационные технологии". — Москва, 2019. — С. 112-125.
2. Иванов И.И. Основы проектирования баз данных для каршеринга // Журнал "Информационные технологии". — 2020. — Т. 15, № 3. — С. 45-52.
3. PostgreSQL Documentation. Официальная документация по PostgreSQL. URL: <https://www.postgresql.org/docs/>
4. Силбершац А. И. Базы данных и СУБД: Курс лекций. — СПб.: Питер, 2017. — 416 с.