|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА**   |  | | --- | | Институт информационных технологий | | Кафедра промышленной информатики | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ И ЛАБОРАТОРНЫМ РАБОТАМ** | | | | |
| **по дисциплине** | | | | |
| «Разработка баз данных» | | | | |
| **Студент группы** | ИНБО-08-22 Щёголев Н.А. |  |  |
|  | *(учебная группа, фамилия, имя, отчество студента)* |  | *(подпись студента)* |
| **Преподаватель** | Ст. преподаватель Кладов А.С. |  |  |
| *(должность, ученая степень, звание, фамилия, имя, отчество преподавателя) (подпись преподавателя)* | | | |

|  |  |
| --- | --- |
| Работа выполнена | «\_\_\_» \_\_\_\_\_\_\_\_ 2024 г. |
| «Зачтено» | «\_\_\_» \_\_\_\_\_\_\_\_ 2024 г. |
|  |  |

Москва 2024

СОДЕРЖАНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА 1 4](#_Toc31481)

[1.1 Создание базы данных и таблиц 4](#_Toc23289)

[ПРАКТИЧЕСКАЯ РАБОТА 2 10](#_Toc19536)

[2.1 Выборка и сортировка данных 10](#_Toc9970)

[2.2 Метасимволы оператора LIKE 13](#_Toc8973)

[2.3 Изменение данных в таблице 14](#_Toc745)

[ПРАКТИЧЕСКАЯ РАБОТА 3 18](#_Toc15928)

[3.1 Создание БД в PostgreSQL 18](#_Toc13653)

[3.2 Перенос БД на другой сервер 24](#_Toc26581)

[3.3 Data Modifying Language 31](#_Toc4091)

[3.4 Создание триггеров и функций 36](#_Toc770)

[ПРАКТИЧЕСКАЯ РАБОТА 4 39](#_Toc10292)

[4.1 Оконные функции 39](#_Toc15291)

[4.2 Агрегатные функции 43](#_Toc28166)

[4.3 Ранжирующие функции 48](#_Toc12574)

[4.4 Функции смещения 52](#_Toc18272)

[ВЫВОДЫ 56](#_Toc6951)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 57](#_Toc31898)

ПРАКТИЧЕСКАЯ РАБОТА 1

1.1 Создание базы данных и таблиц

На основе спроектированной базы данных создадим новую БД в MySQL Server.



Рисунок 1.1 - Физическая модель БД Автосервиса

Напишем код для создания базы данных и таблиц и вставим его в консоль. Посмотрим список баз данных с помощью команды show databases.

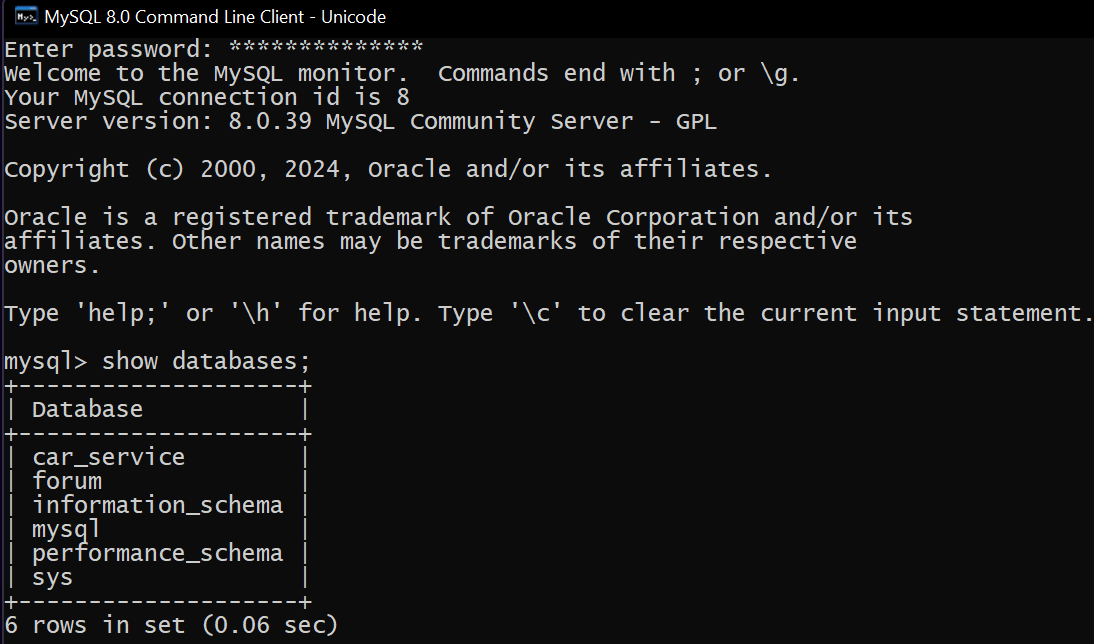


Рисунок 1.2 - Список баз данных

Теперь подключимся к базе данных car\_service, затем посмотрим список таблиц с помощью команды show tables.

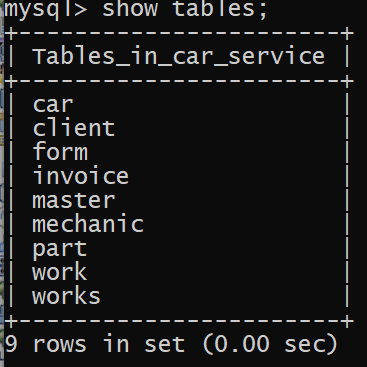


Рисунок 1.3 - Список всех таблиц БД

В ответе видим названия наших таблиц. Теперь посмотрим описание для каждой из таблиц.

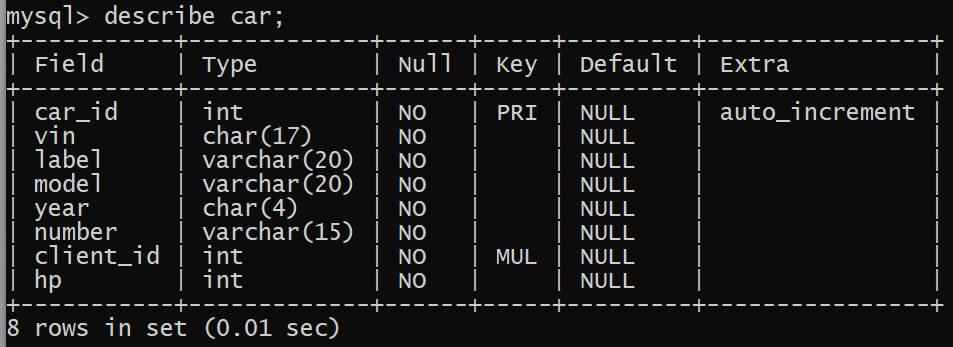


Рисунок 1.4 - Описание таблицы car

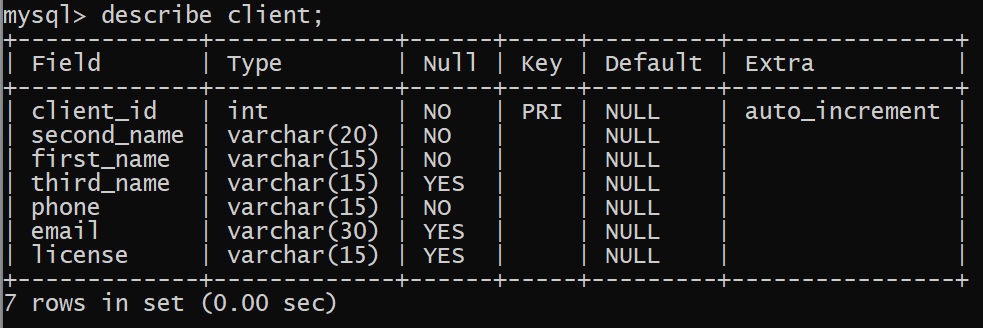


Рисунок 1.5 - Описание таблицы client

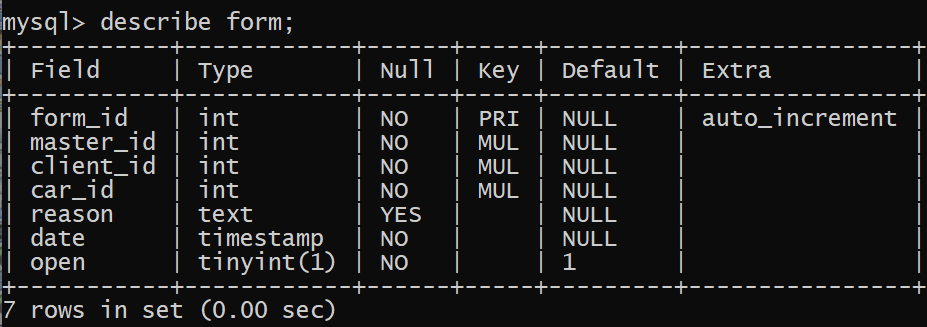


Рисунок 1.6 - Описание таблицы form (заявка)

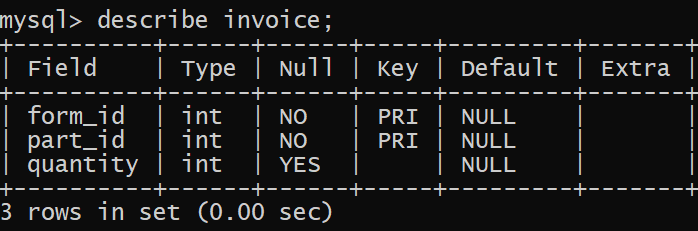


Рисунок 1.7 - Описание таблицы invoice (накладная)

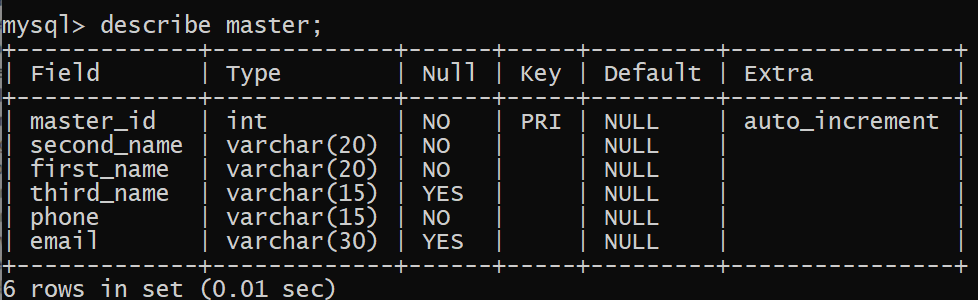


Рисунок 1.8 - Описание таблицы master

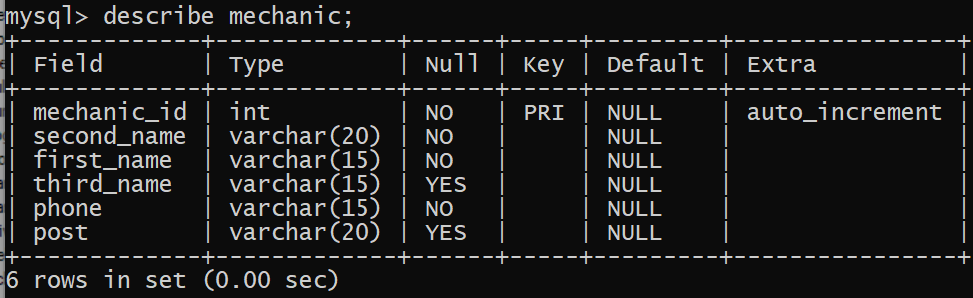


Рисунок 1.9 - Описание таблицы mechanic

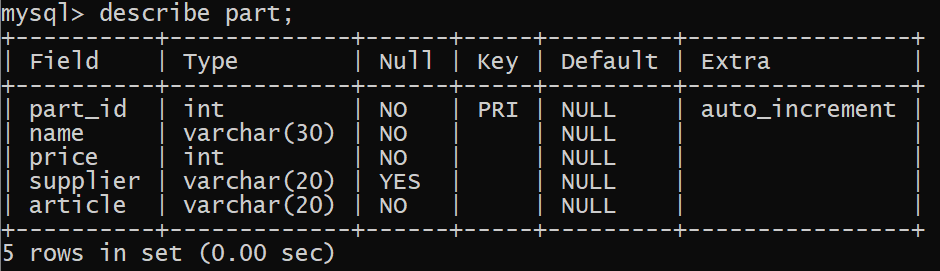


Рисунок 1.10 - Описание таблицы part

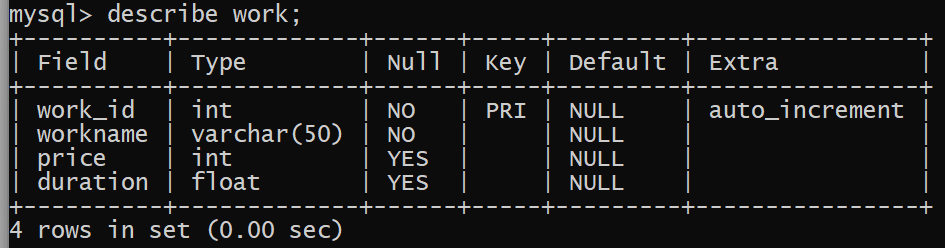


Рисунок 1.11 - Описание таблицыwork

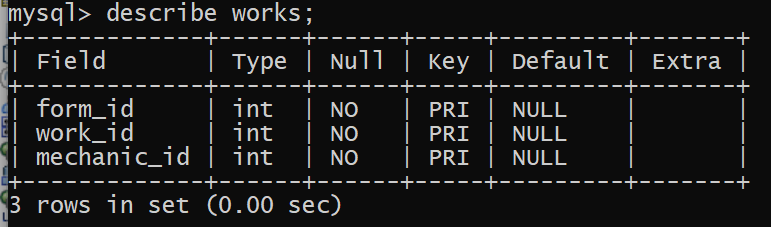


Рисунок 1.12 - Описание таблицыworks

Внесем данные в таблицу при помощи команды INSERT.

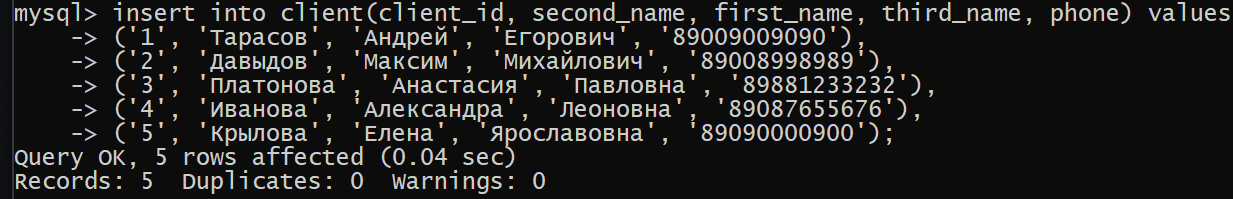


Рисунок 1.13 - Заполнение таблицы client

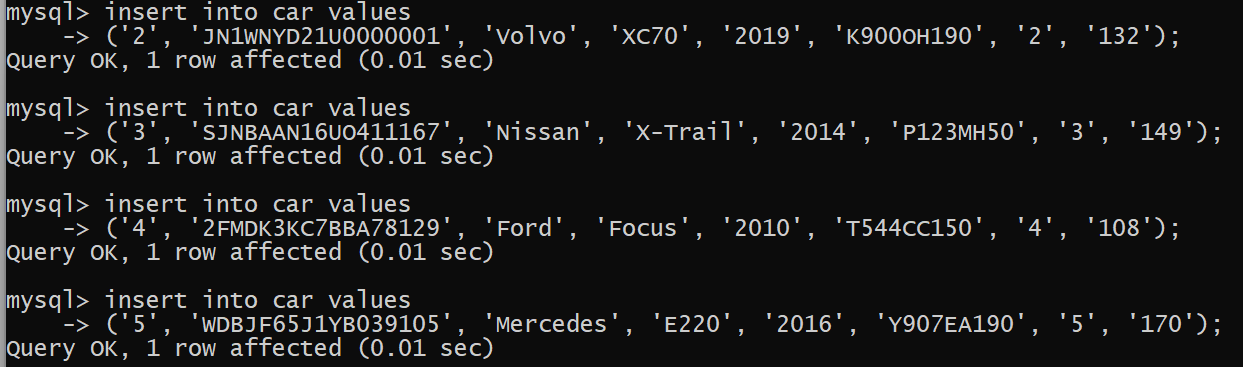


Рисунок 1.14 - Заполнение таблицы car

Аналогичным образом вставляем значения в каждую из наших таблиц. После того, как все значения будут вставлены, выведем их на экран при помощи команды SELECT \*.

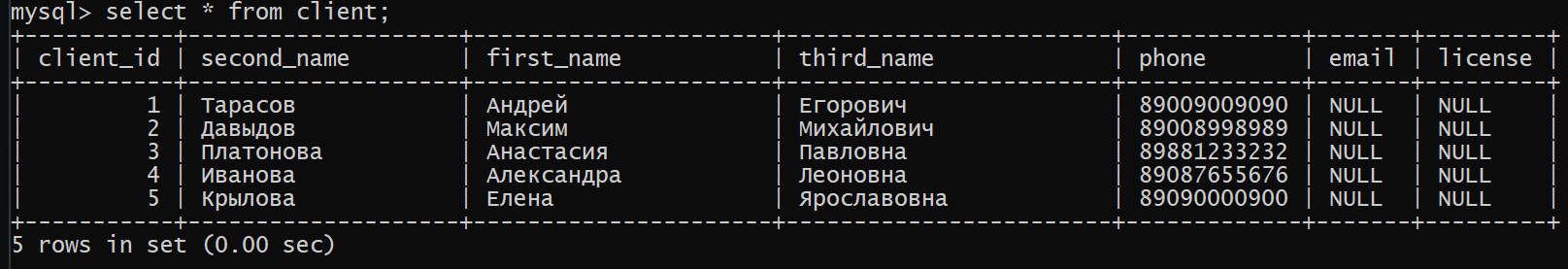


Рисунок 1.15 - Данные из таблицы client

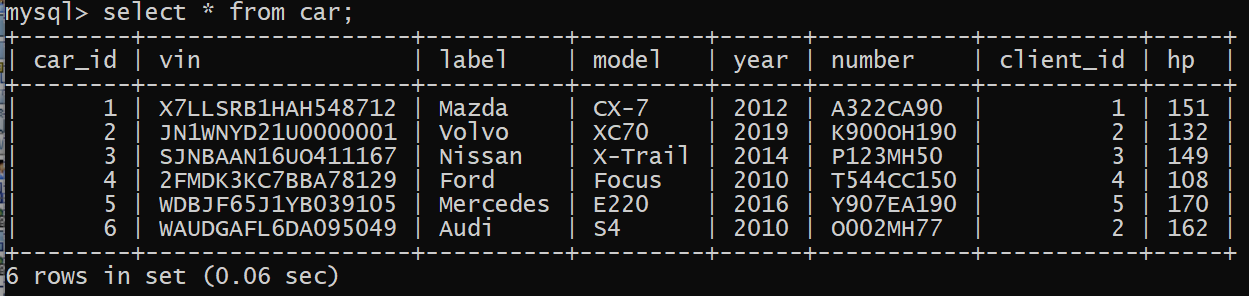


Рисунок 1.16 - Данные из таблицы car

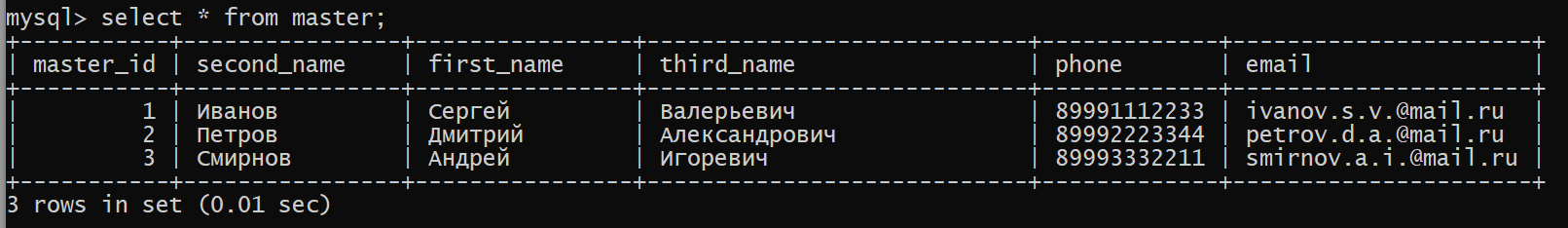


Рисунок 1.17 - Данные из таблицы master

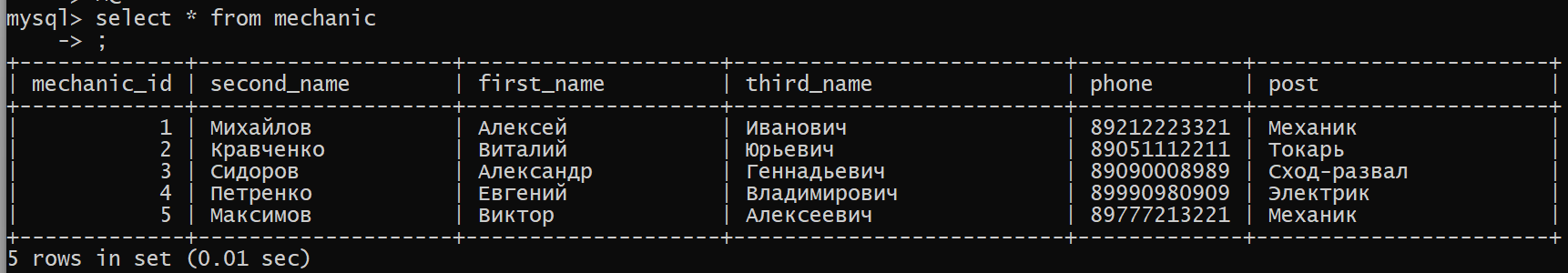


Рисунок 1.18 - Данные из таблицы mechanic

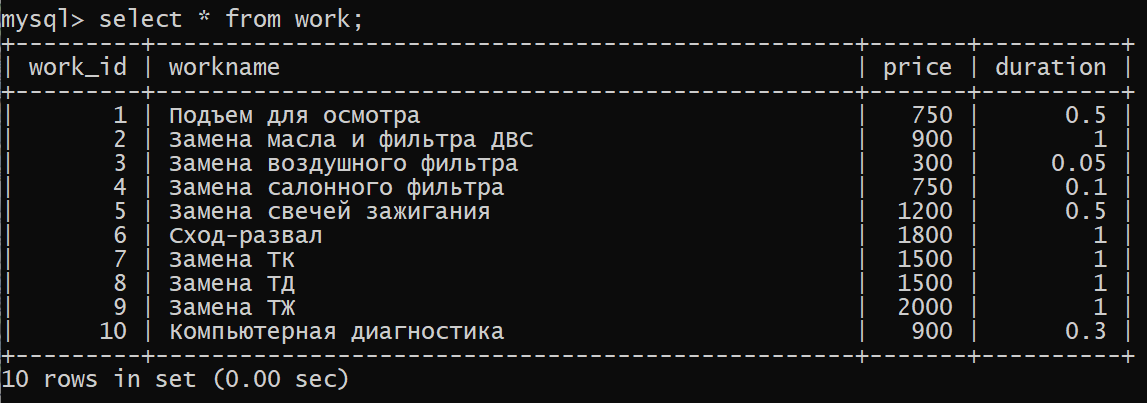


Рисунок 1.19 - Данные из таблицы work

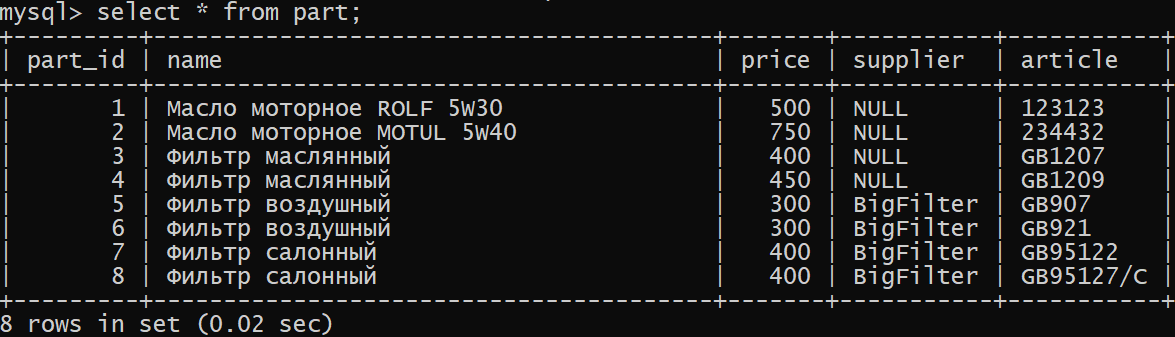


Рисунок 1.20 - Данные из таблицы part

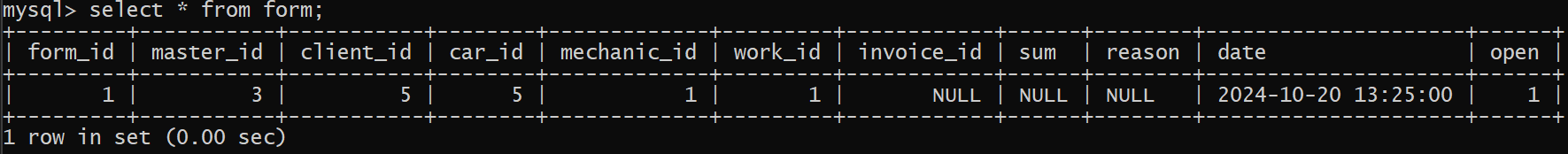


Рисунок 1.21 - Данные из таблицы form

ПРАКТИЧЕСКАЯ РАБОТА 2

2.1 Выборка и сортировка данных

Посмотрим выполненные работы для клиенты, объедив три таблицы и перечислив интересующие нас столбцы через запятую, все что идет после нам пока не интересно:

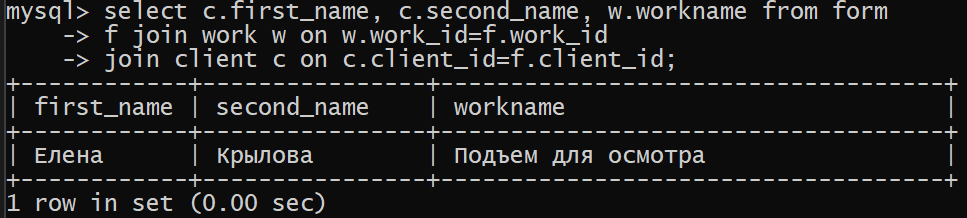


Рисунок 2.1 - Работы для клиента

Посмотрим все выполненные работы:

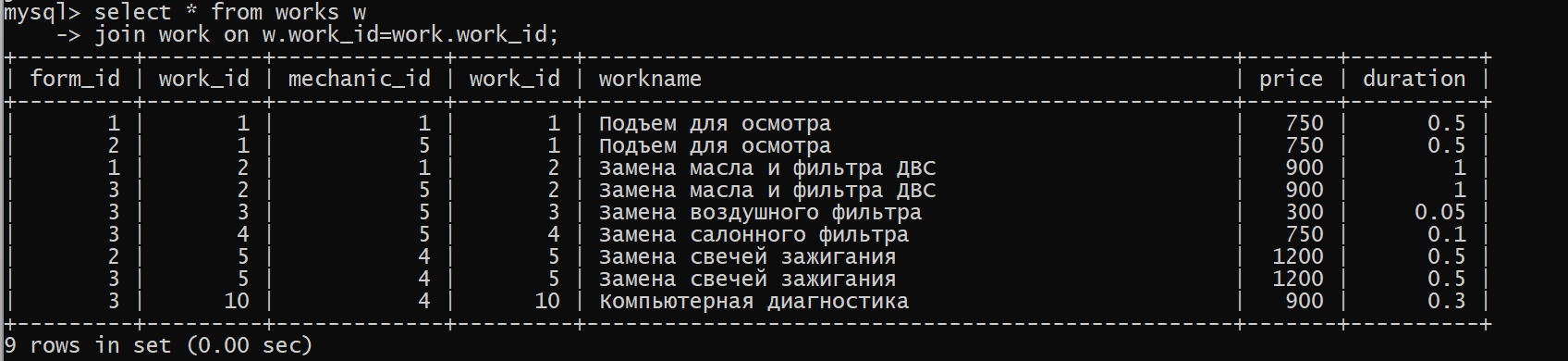


Рисунок 2.2 - Все выполненные работы

Если нужно посмотреть, например, какие работы выполнил нужный нам механик, используем оператор WHERE:

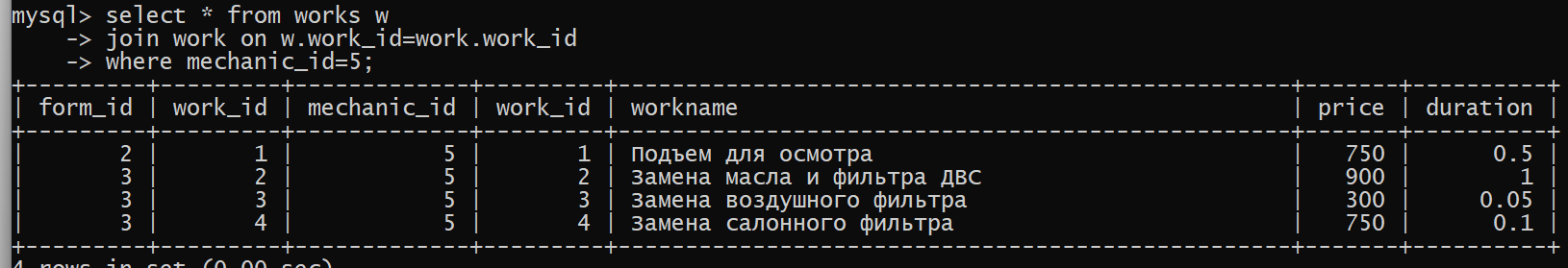


Рисунок 2.3 - Выполненные работы механиком с id=5

Аналогично, посмотрим запчасти нудной нам заявки:

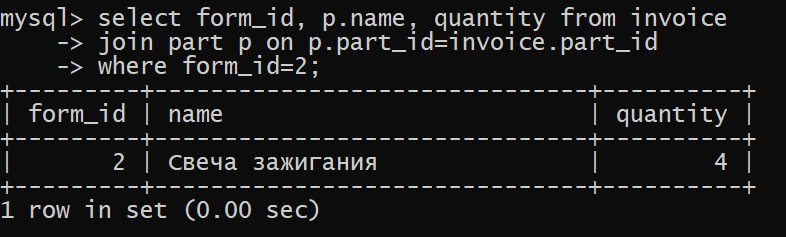


Рисунок 2.4 - Запчасти для заявки №2

Если мы хотим отсортивать данные по интересующему нас столбцу, то для этого в SQL существует ключевое слово ORDER BY после которого указывается имя столбца по которому будет происходить сортировка. По умолчанию сортировка идет по возрастанию, но это можно изменить, добавив ключевое слово DESC. Сортировку можно производить сразу по нескольким столбцам, указав нужные параметры через запятую, сначала отсортируется по первому параметру, кортежи с одинаковым первым параметром сортируются по второму параметру.

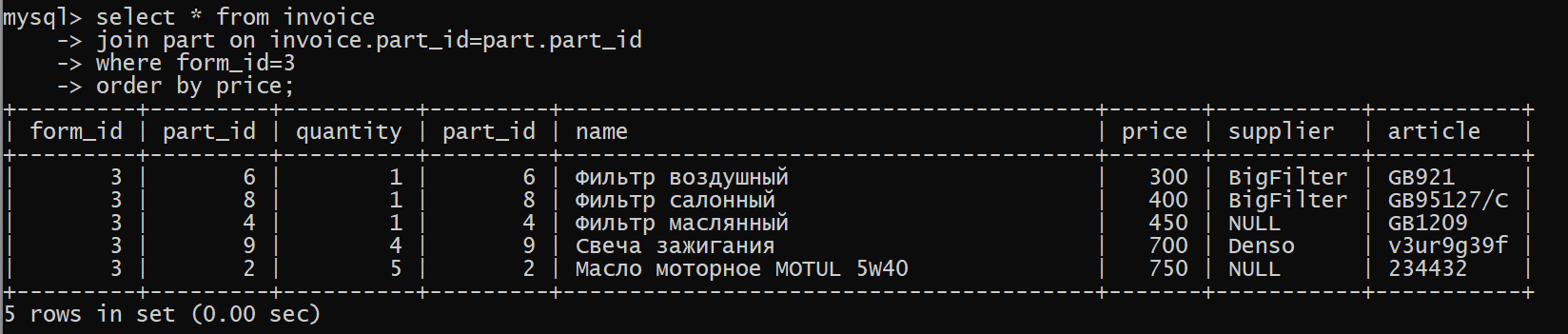


Рисунок 2.5 - Сортировка работ по возрастанию цены

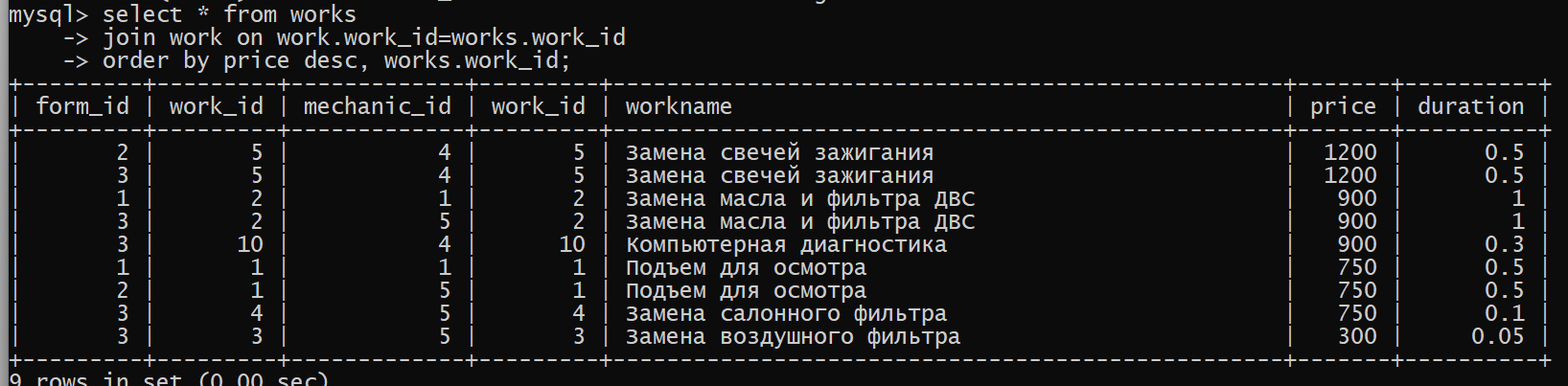


Рисунок 2.6 - Сортировка работ сначала по убыванию цены, потом по номеру заявки

ля поиска значений в указанном диапазоне используется оператор BETWEEN:

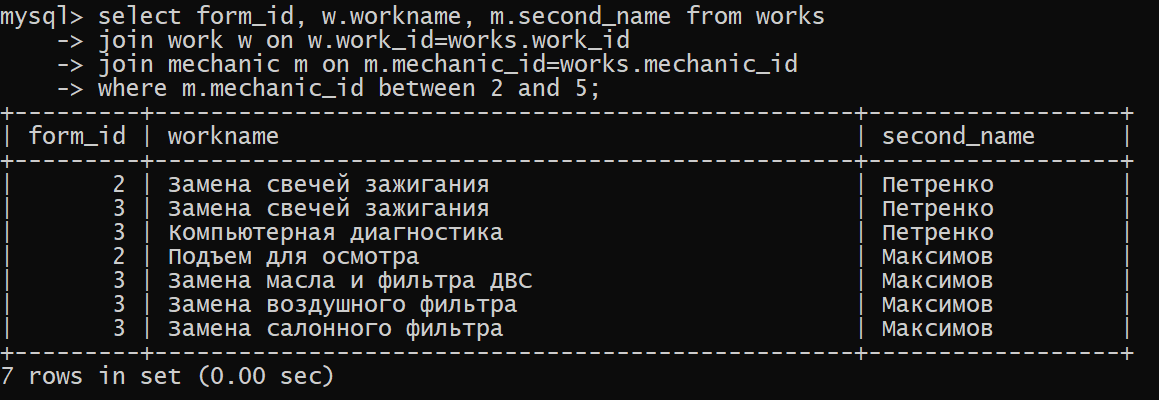


Рисунок 2.7 - Записи в диапазоне

Для отбора значений, не равных нулю, используется оператор IS NOT NULL:

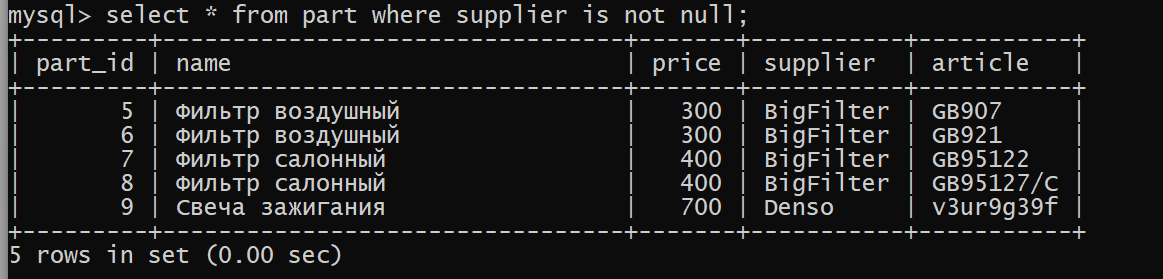


Рисунок 2.8 - Записи, где стобец не равен NULL

Для отбора значений, которые не равны указанным, используется оператор IN:

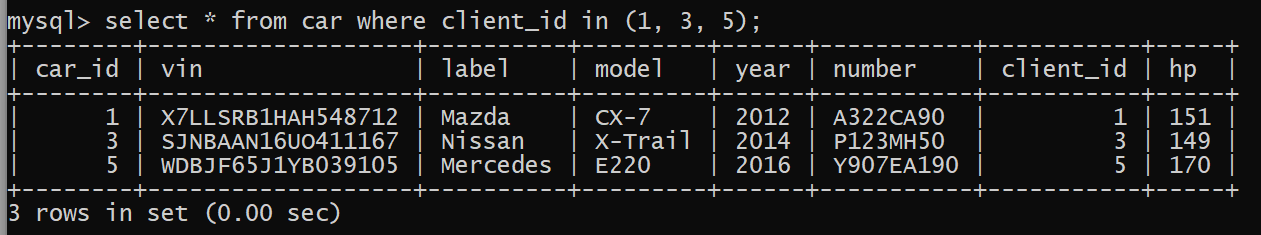


Рисунок 2.9 - Записи, значение принадлежит множеству значений

2.2 Метасимволы оператора LIKE

Поиск с использованием метасимволов может осуществляться только в текстовых полях.

Самый распространенный метасимвол — %. Он означает любые символы. Например, если надо найти записи, по конкретной дате.

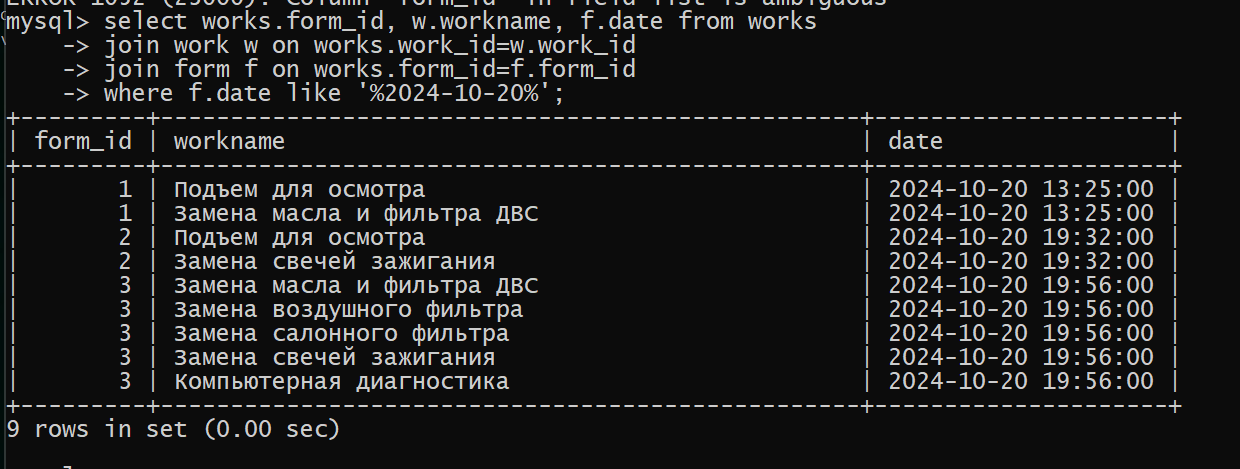
:

Рисунок 2.10 - Поиск с метасимволом '%'

Также можно искать записи, не содержащие комбинации символов:

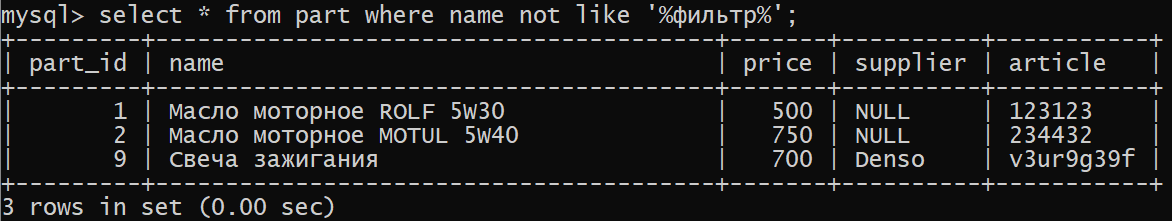
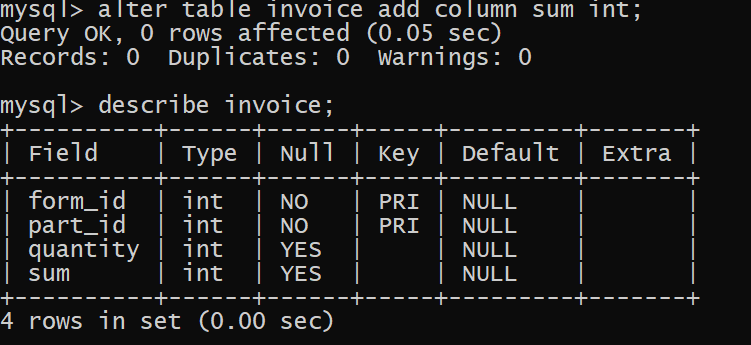
:

Рисунок 2.11 - Поиск с метасимволом '%'

2.3 Изменение данных в таблице

Для добавления столбцов в таблицу используется оператор ALTER TABLE — ADD COLUMN.



**Рисунок 2.12 - Добавление нового столбца в таблицу**

Для того, чтобы указать местоположение столбца используются ключевые слова: FIRST — новый столбец будет первым, и AFTER — указывает после какого столбца поместить новый.

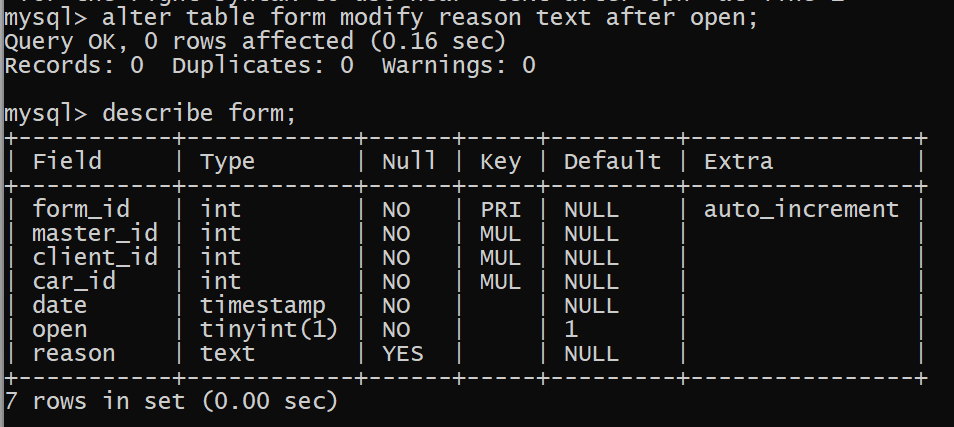


Рисунок 2.13 - Добавление еще одного столбца

А теперь установим новые значения по условию:

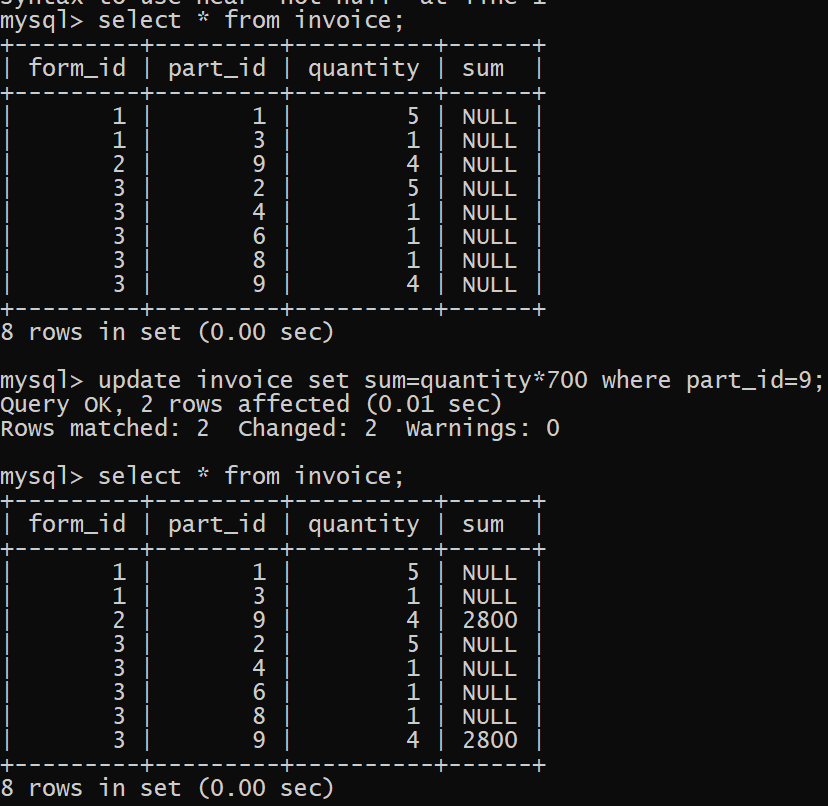


Рисунок 2.14 - Результат обновления

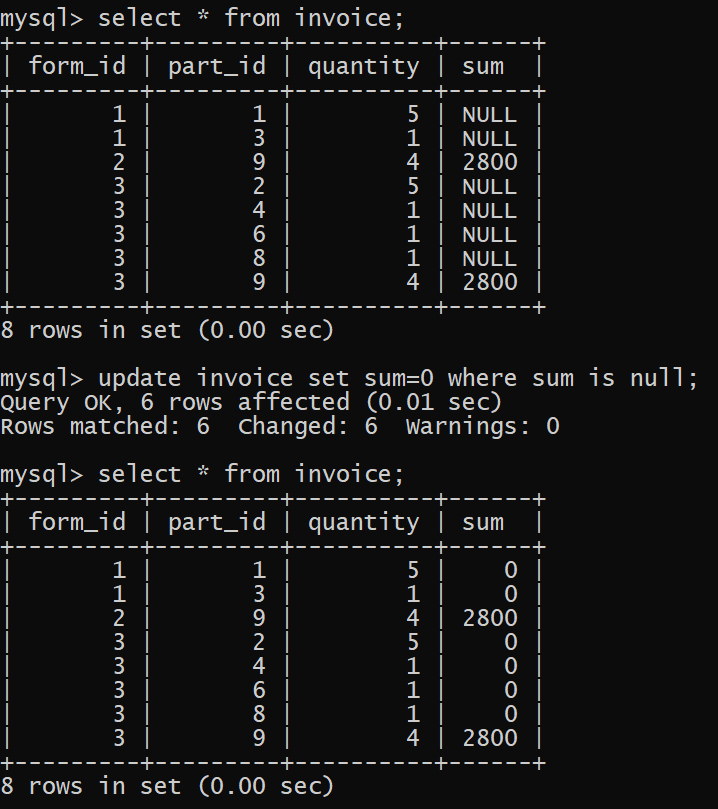


Рисунок 2.15- Результат обновления

Для удаления стобцов, существует оператор DROP:



**Рисунок 2.16 - Изменение столбца в** таблице

Рассмотрим — оператор DELETE, который позволяет удалять строки из таблицы.

Давайте из таблицы car удалим те записи, где id=7:

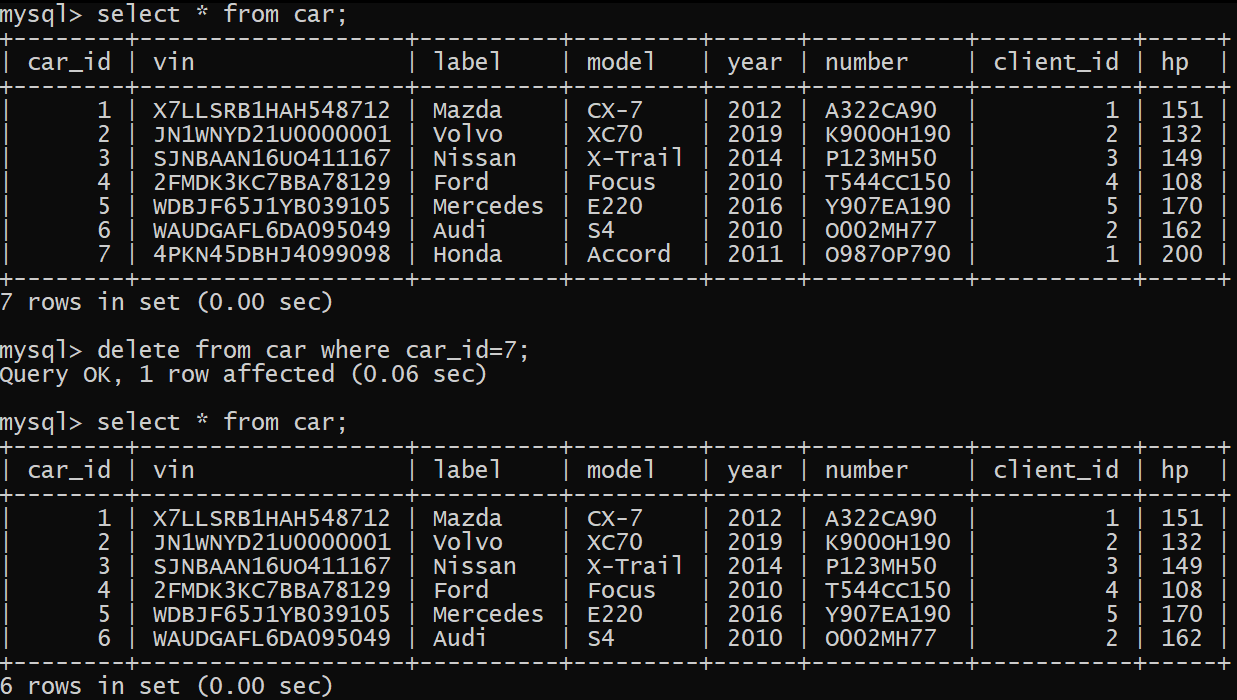


Рисунок 2.18 - Удаление записи из таблицы

ПРАКТИЧЕСКАЯ РАБОТА 3

3.1 Создание БД в PostgreSQL

При создании БД указываем ее имя, адрес подключения, порт, пользователя и пароль:

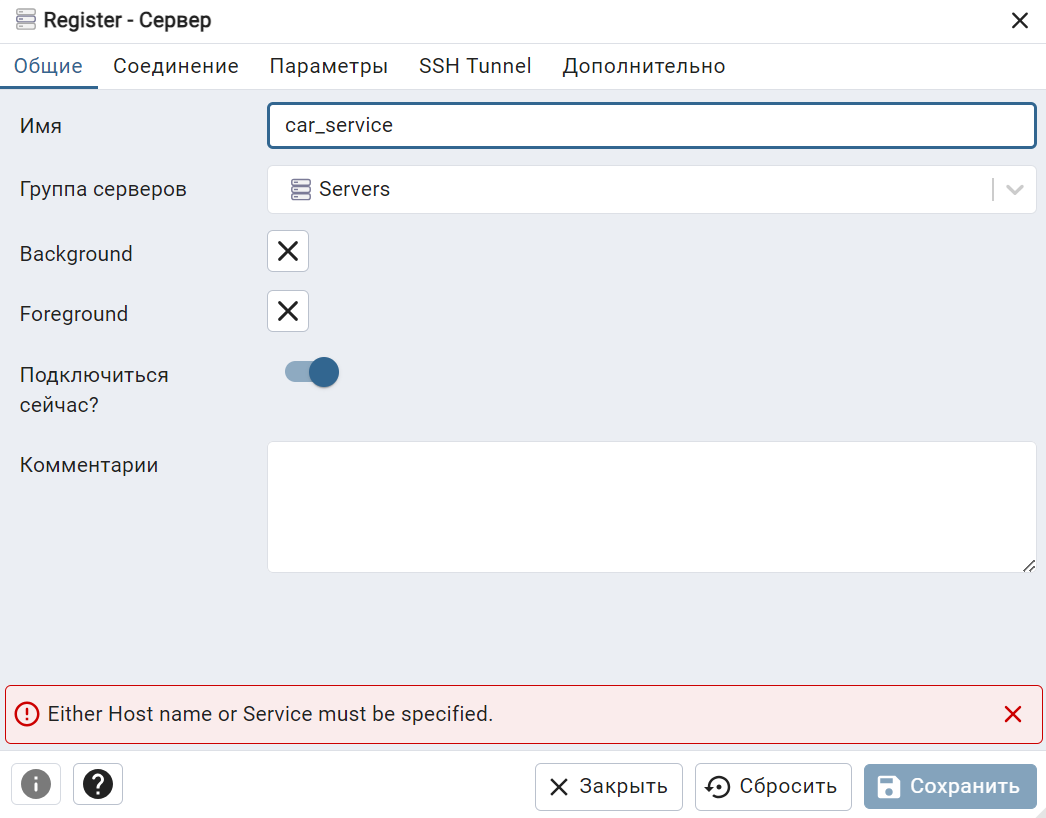


Рисунок 3.1 - Создание БД

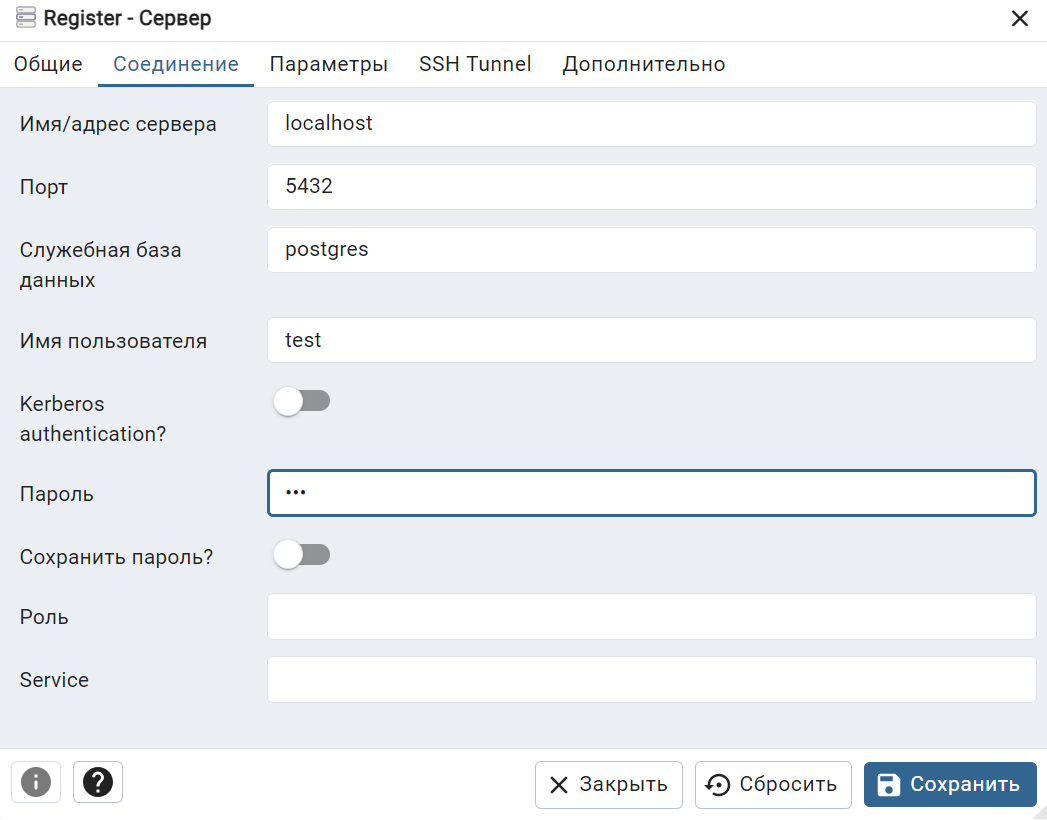


Рисунок 3.2 - Указание параметров подключения

Создадим таблицы БД:

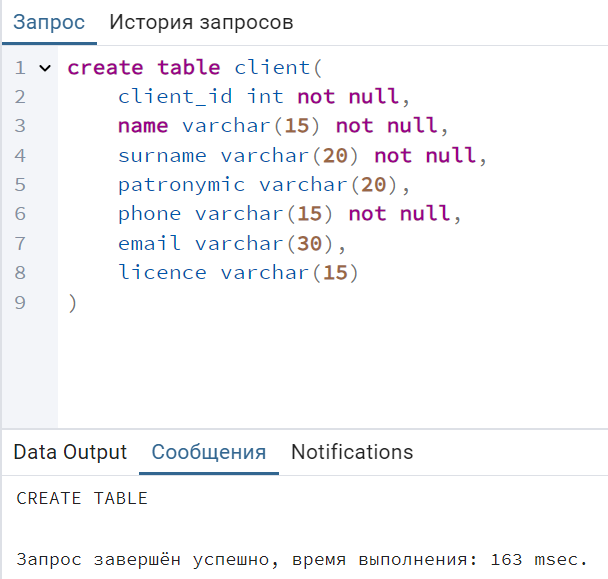


Рисунок 3.3 - Создание таблицы Клиент

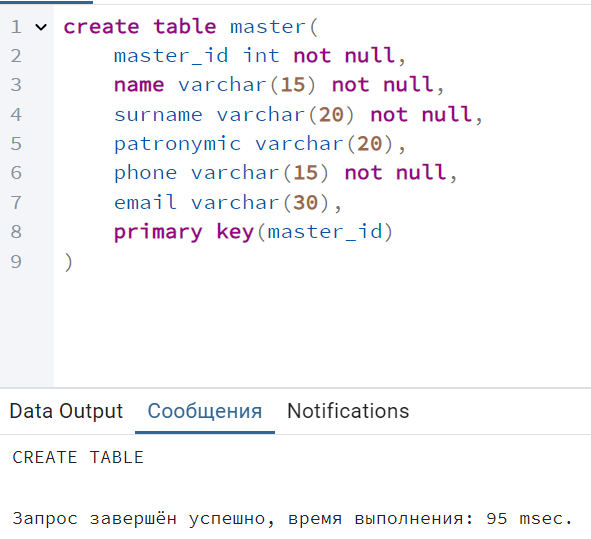


Рисунок 3.4 - Создание таблицы Мастер

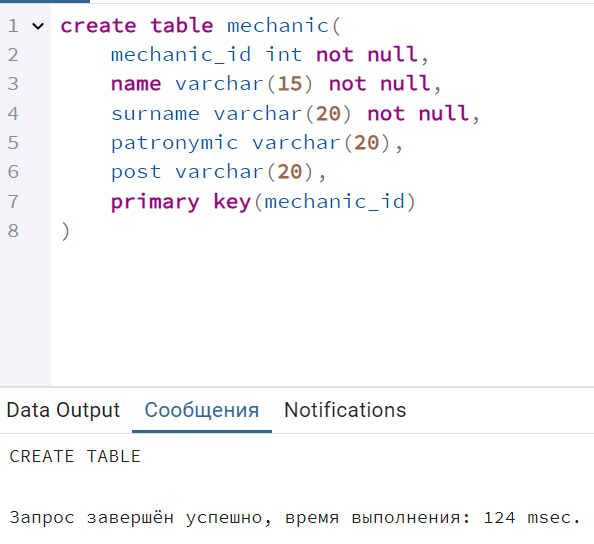


Рисунок 3.5 - Создание таблицы Механик



Рисунок 3.6 - Создание таблицы Машина

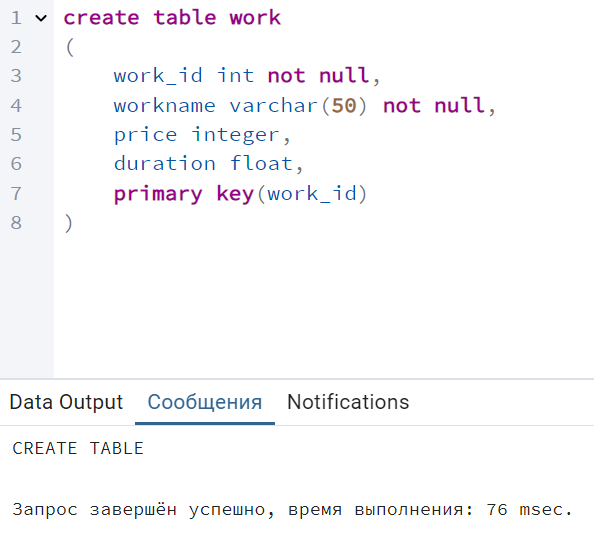


Рисунок 3.7 - Создание таблицы Работа

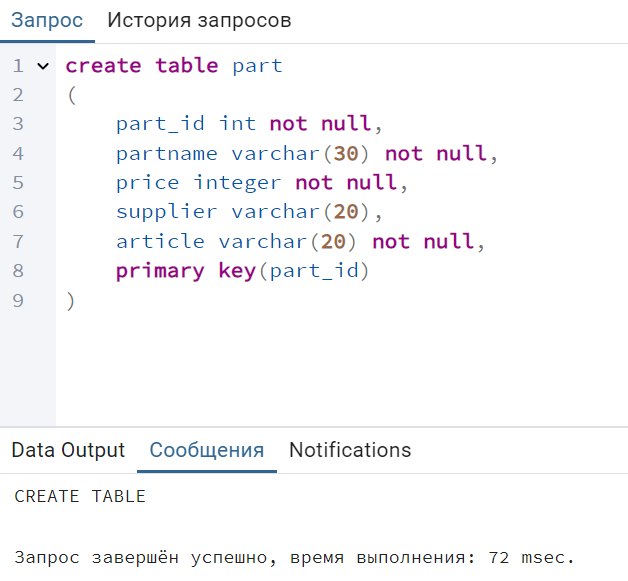


Рисунок 3.8 - Создание таблицы запчасть

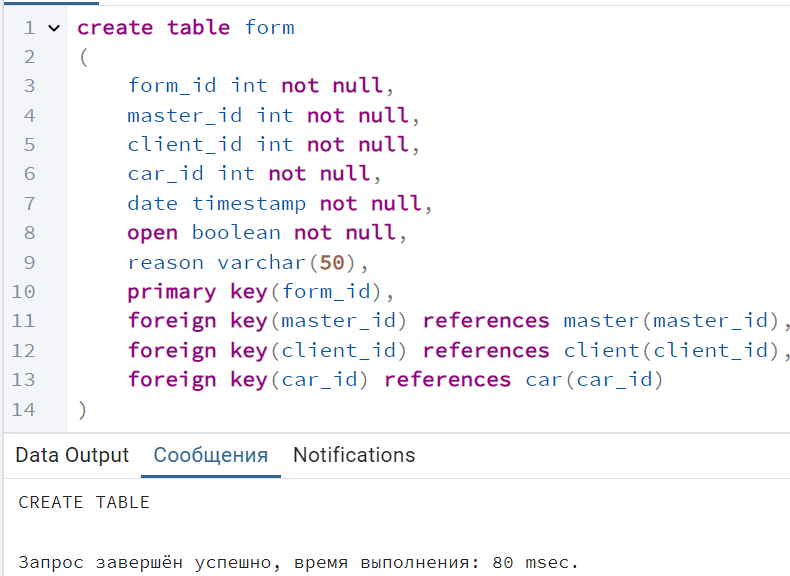


Рисунок 3.9 - Создание таблицы заявка



Рисунок 3.10 - Создание таблицы накладная

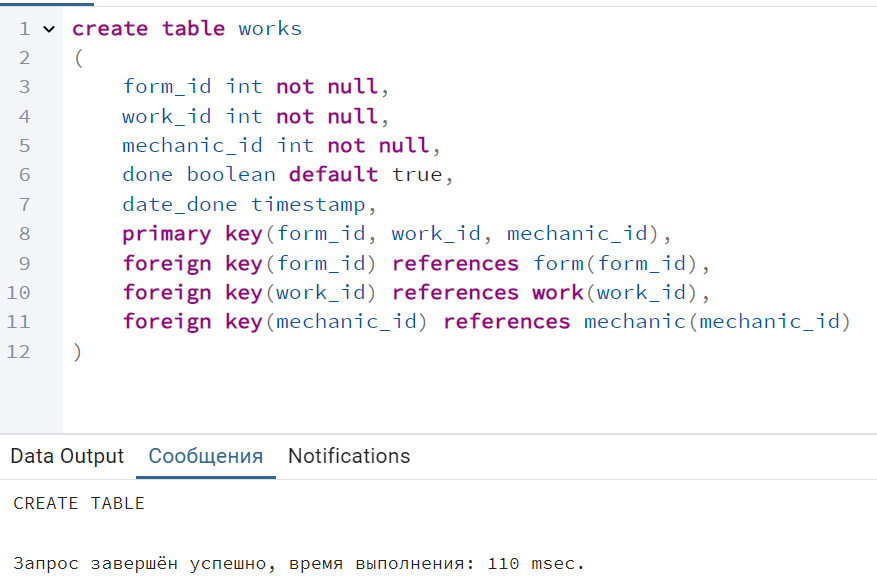


Рисунок 3.11 - Создание таблицы Работы

На основе созданных отношений PostgreSQL может разработать схему созданной БД:

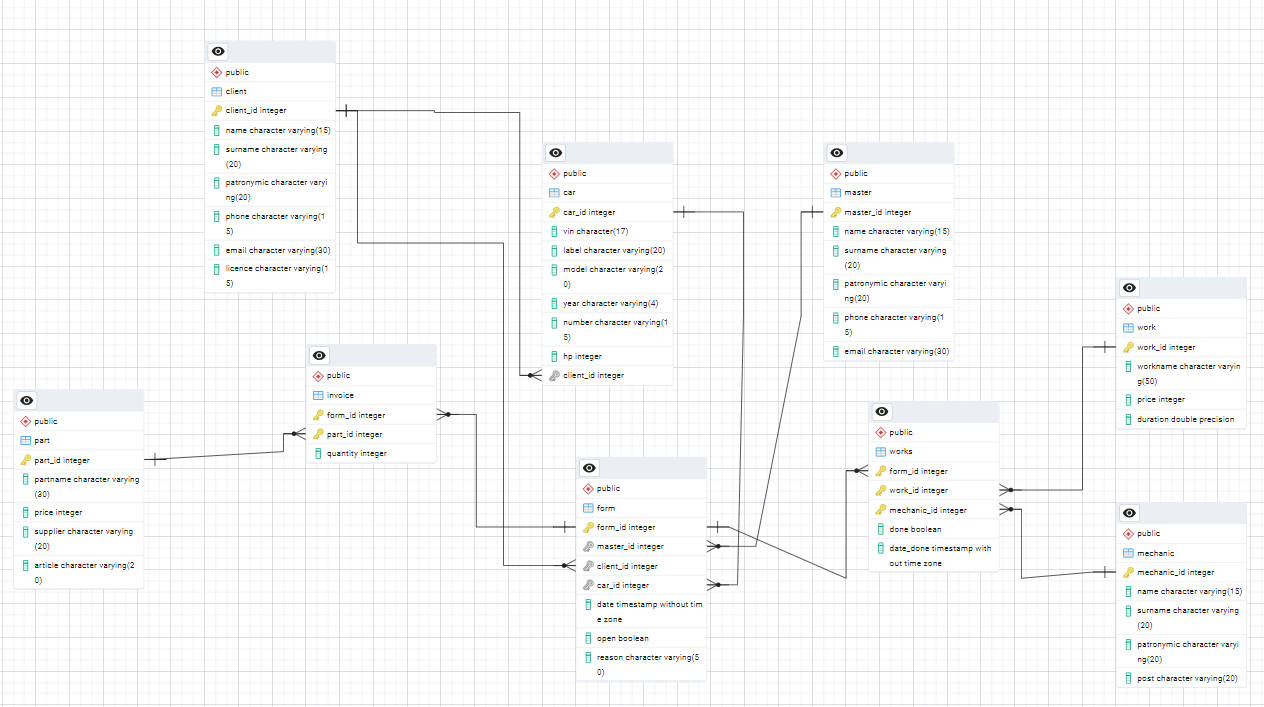


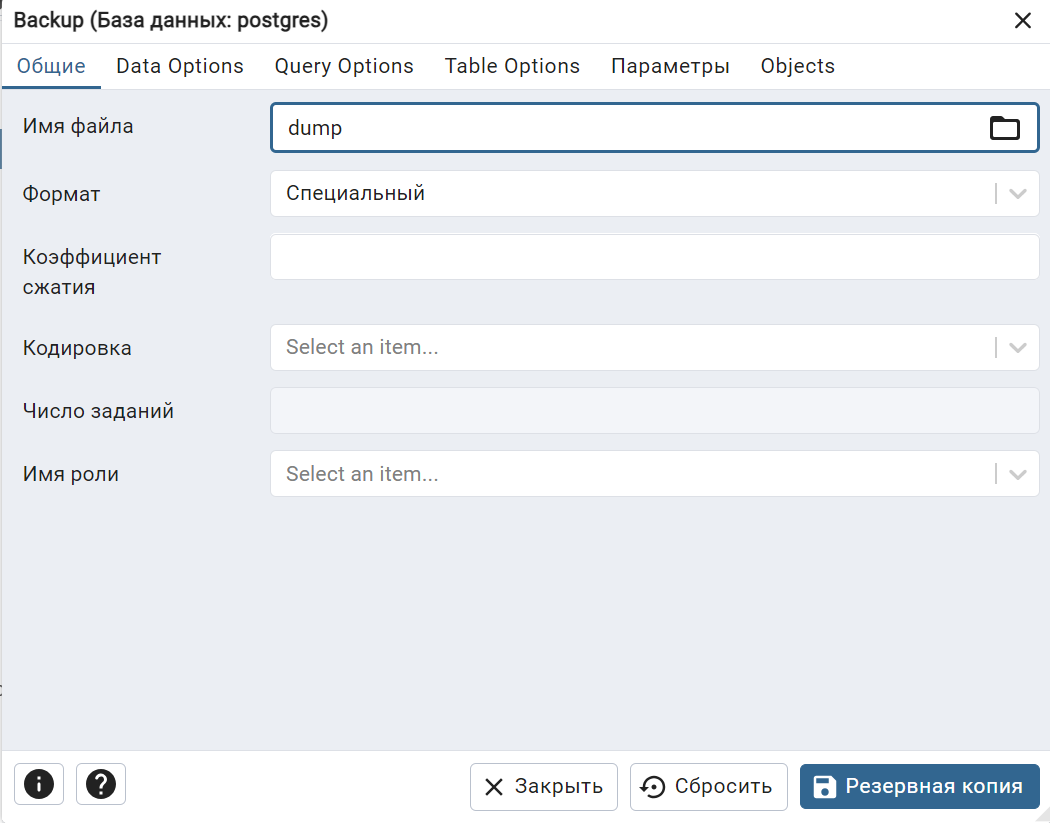
Рисунок 3.12 - Сгенерированная схема отношений

3.2 Перенос БД на другой сервер

Любое СУБД имеет средства резервного копирования базы данных. Та-

кому копированию подвергаются как метаданные (структура данных базы), так и сами данные. Конечно, каждое СУБД имеет свои собственные форматы, но традиционным форматом является сохранение в виде последовательности SQL- команд (создания, вставки, изменения) (SQL-скрипт), выполнение которых приведет к текущему состоянию базы данных.

Для создания бэкапа в pgAdmin есть специальная функция «Резервное копирование»:



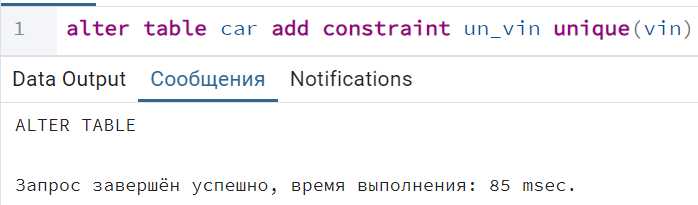
**Рисунок 3.13 - Создание бэкапа**

Листинг 1 – Код для создания БД и таблиц

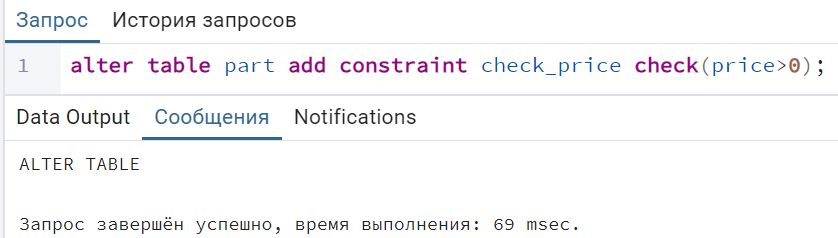
|  |
| --- |
| -- Table: public.car  -- DROP TABLE IF EXISTS public.car;  CREATE TABLE IF NOT EXISTS public.car  (  car\_id integer NOT NULL,  vin character(17) COLLATE pg\_catalog."default" NOT NULL,  label character varying(20) COLLATE pg\_catalog."default" NOT NULL,  model character varying(20) COLLATE pg\_catalog."default" NOT NULL,  year character varying(4) COLLATE pg\_catalog."default" NOT NULL,  "number" character varying(15) COLLATE pg\_catalog."default" NOT NULL,  hp integer NOT NULL,  client\_id integer NOT NULL,  CONSTRAINT car\_pkey PRIMARY KEY (car\_id),  CONSTRAINT car\_client\_id\_fkey FOREIGN KEY (client\_id)  REFERENCES public.client (client\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.car  OWNER to test;  -- Table: public.client  -- DROP TABLE IF EXISTS public.client;  CREATE TABLE IF NOT EXISTS public.client  (  client\_id integer NOT NULL,  name character varying(15) COLLATE pg\_catalog."default" NOT NULL,  surname character varying(20) COLLATE pg\_catalog."default" NOT NULL,  patronymic character varying(20) COLLATE pg\_catalog."default",  phone character varying(15) COLLATE pg\_catalog."default" NOT NULL,  email character varying(30) COLLATE pg\_catalog."default",  licence character varying(15) COLLATE pg\_catalog."default",  CONSTRAINT client\_pkey PRIMARY KEY (client\_id)  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.client  OWNER to test;  -- Table: public.form  -- DROP TABLE IF EXISTS public.form;  CREATE TABLE IF NOT EXISTS public.form  (  form\_id integer NOT NULL,  master\_id integer NOT NULL,  client\_id integer NOT NULL,  car\_id integer NOT NULL,  date timestamp without time zone NOT NULL,  open boolean NOT NULL DEFAULT true,  reason character varying(50) COLLATE pg\_catalog."default",  CONSTRAINT form\_pkey PRIMARY KEY (form\_id),  CONSTRAINT form\_car\_id\_fkey FOREIGN KEY (car\_id)  REFERENCES public.car (car\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT form\_client\_id\_fkey FOREIGN KEY (client\_id)  REFERENCES public.client (client\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT form\_master\_id\_fkey FOREIGN KEY (master\_id)  REFERENCES public.master (master\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.form  OWNER to test;  -- Table: public.invoice  -- DROP TABLE IF EXISTS public.invoice;  CREATE TABLE IF NOT EXISTS public.invoice  (  form\_id integer NOT NULL,  part\_id integer NOT NULL,  quantity integer NOT NULL DEFAULT 1,  CONSTRAINT invoice\_pkey PRIMARY KEY (form\_id, part\_id),  CONSTRAINT invoice\_form\_id\_fkey FOREIGN KEY (form\_id)  REFERENCES public.form (form\_id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT invoice\_part\_id\_fkey FOREIGN KEY (part\_id)  REFERENCES public.part (part\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.invoice  OWNER to test;  -- Table: public.master  -- DROP TABLE IF EXISTS public.master;  CREATE TABLE IF NOT EXISTS public.master  (  master\_id integer NOT NULL,  name character varying(15) COLLATE pg\_catalog."default" NOT NULL,  surname character varying(20) COLLATE pg\_catalog."default" NOT NULL,  patronymic character varying(20) COLLATE pg\_catalog."default",  phone character varying(15) COLLATE pg\_catalog."default" NOT NULL,  email character varying(30) COLLATE pg\_catalog."default",  CONSTRAINT master\_pkey PRIMARY KEY (master\_id)  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.master  OWNER to test;  -- Table: public.mechanic  -- DROP TABLE IF EXISTS public.mechanic;  CREATE TABLE IF NOT EXISTS public.mechanic  (  mechanic\_id integer NOT NULL,  name character varying(15) COLLATE pg\_catalog."default" NOT NULL,  surname character varying(20) COLLATE pg\_catalog."default" NOT NULL,  patronymic character varying(20) COLLATE pg\_catalog."default",  post character varying(20) COLLATE pg\_catalog."default",  CONSTRAINT mechanic\_pkey PRIMARY KEY (mechanic\_id)  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.mechanic  OWNER to test;  -- Table: public.part  -- DROP TABLE IF EXISTS public.part;  CREATE TABLE IF NOT EXISTS public.part  (  part\_id integer NOT NULL,  partname character varying(30) COLLATE pg\_catalog."default" NOT NULL,  price integer NOT NULL,  supplier character varying(20) COLLATE pg\_catalog."default",  article character varying(20) COLLATE pg\_catalog."default" NOT NULL,  CONSTRAINT part\_pkey PRIMARY KEY (part\_id)  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.part  OWNER to test;  -- Table: public.work  -- DROP TABLE IF EXISTS public.work;  CREATE TABLE IF NOT EXISTS public.work  (  work\_id integer NOT NULL,  workname character varying(50) COLLATE pg\_catalog."default" NOT NULL,  price integer,  duration double precision,  CONSTRAINT work\_pkey PRIMARY KEY (work\_id)  )  TABLESPACE pg\_default;  ALTER TABLE IF EXISTS public.work  OWNER to test;  -- Table: public.works  -- DROP TABLE IF EXISTS public.works;  CREATE TABLE IF NOT EXISTS public.works  (  form\_id integer NOT NULL,  work\_id integer NOT NULL,  mechanic\_id integer NOT NULL,  done boolean DEFAULT true,  date\_done timestamp without time zone,  CONSTRAINT works\_pkey PRIMARY KEY (form\_id, work\_id, mechanic\_id),  CONSTRAINT works\_form\_id\_fkey FOREIGN KEY (form\_id)  REFERENCES public.form (form\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT works\_mechanic\_id\_fkey FOREIGN KEY (mechanic\_id)  REFERENCES public.mechanic (mechanic\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT works\_work\_id\_fkey FOREIGN KEY (work\_id)  REFERENCES public.work (work\_id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE) |

3.3 Data Modifying Language

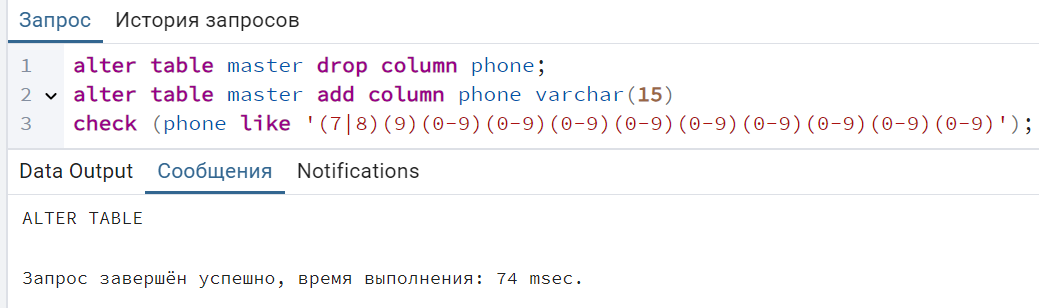
Создадим проверки при вставке записей в таблицу при помощи оператора ADD CONSTRAINT:



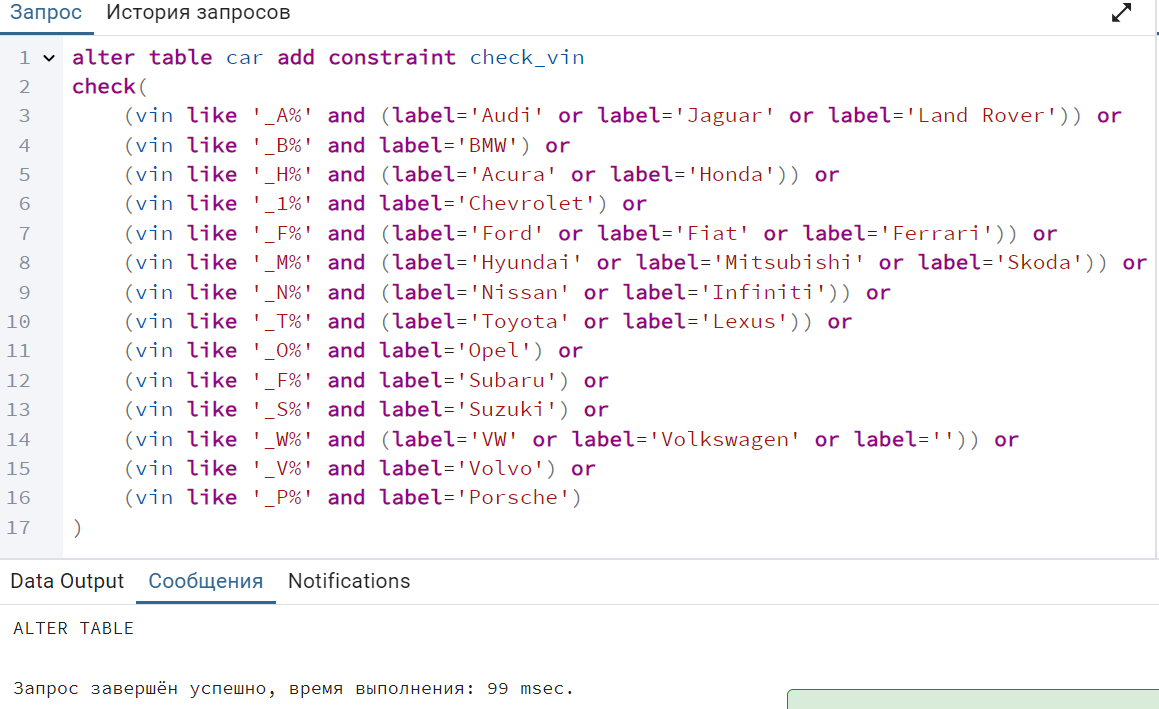
**Рисунок 3.14 - Создание проверки на уникальность VIN номера**



**Рисунок 3.15- Создание проверки положительную цену**



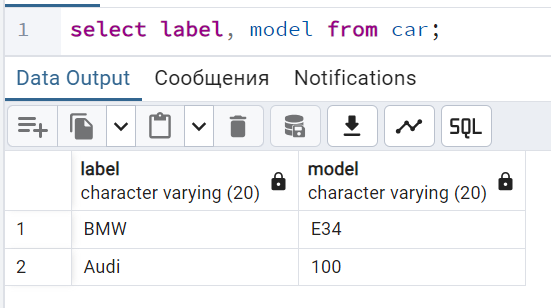
**Рисунок 3.16- Создание проверки на соответствие номера шаблону**



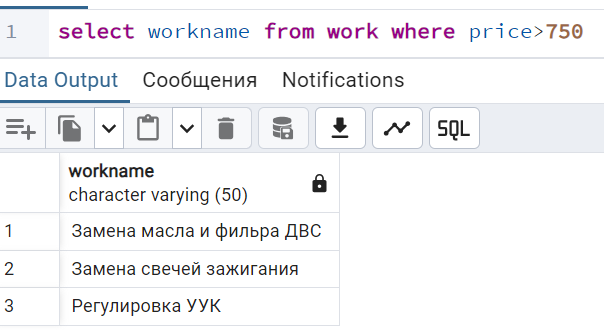
**Рисунок 3.17- Создание проверки на соответствие VIN и модели**

Одной из основных задач, которую решают базы данных – это эффективный поиск необходимых данных. Универсальным подходом для решения этой задачи является применение специального оператора языка SQL - SELECT. Этот оператор достаточно сложный, имеет множество возможностей. Теоретической основой этого оператора является реляционная алгебра, которая доказывает возможность получения с помощью конечного набора операций любых возможных наборов данных.

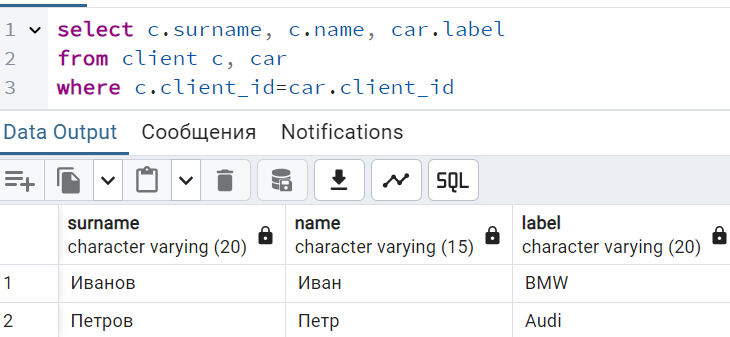
Приведем несколько примеров запросов к таблицам базы данных, демонстрирующих различные возможности языка SQL и различные операции реляционной алгебры.



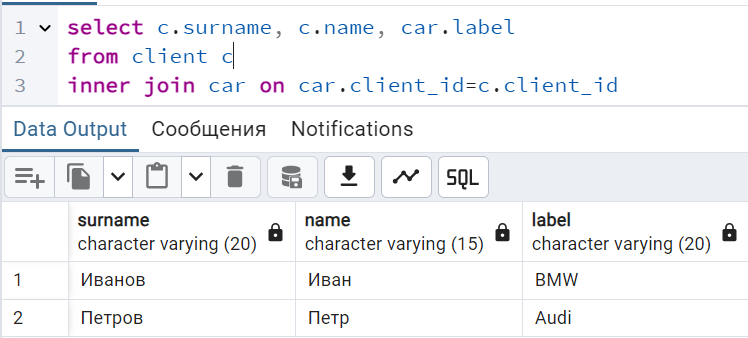
**Рисунок 3.18 - операция проекции**



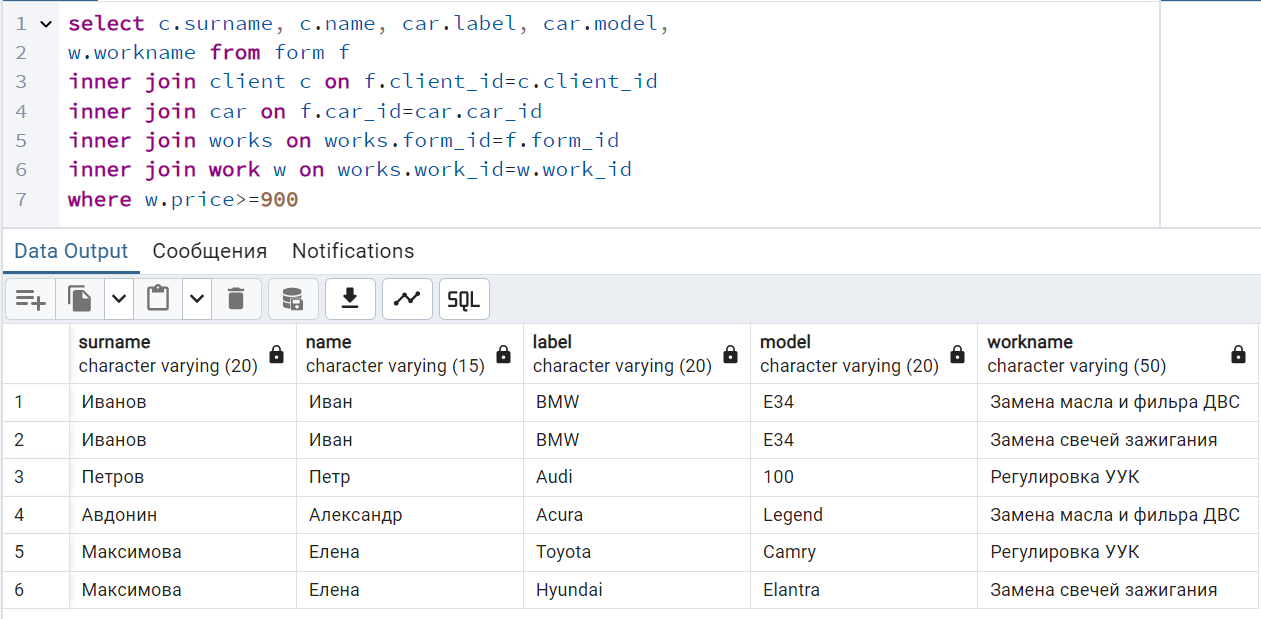
**Рисунок 3.18 - операция селекции**



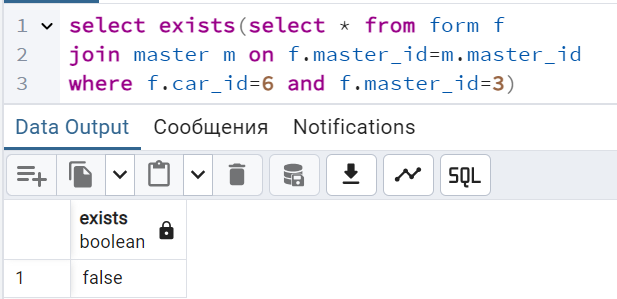
**Рисунок 3.18 - операция тета-соединения**



**Рисунок 3.18 - операция соединения по эквивалентности**



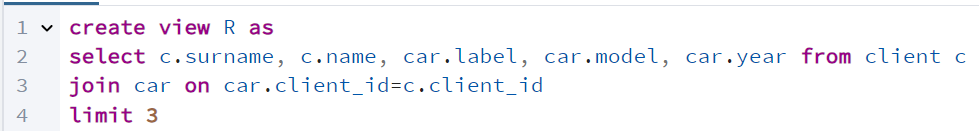
**Рисунок 3.18 - операция пересечения**



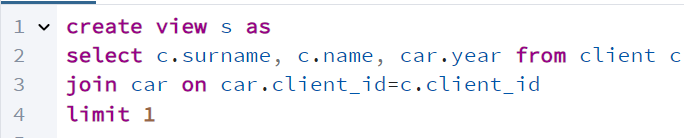
**Рисунок 3.19 - проверка существования записи**



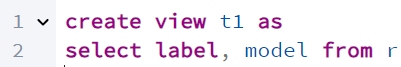
**Рисунок 3.20 - операция разности**



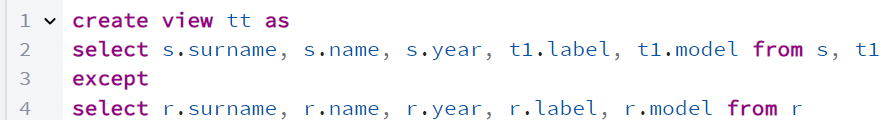
**Рисунок 3.21 - операция деления: создание отношения R**



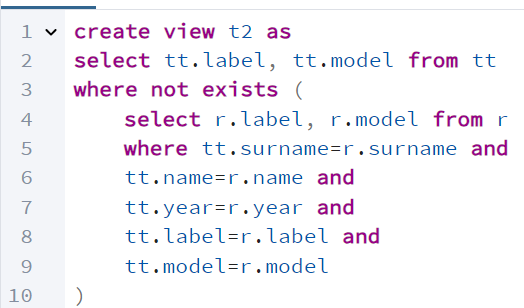
**Рисунок 3.22 - операция деления: создание отношения S**



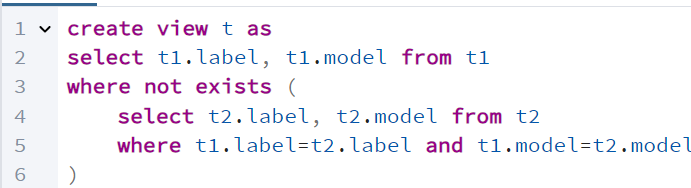
**Рисунок 3.23 - операция деления: создание отношения T1**



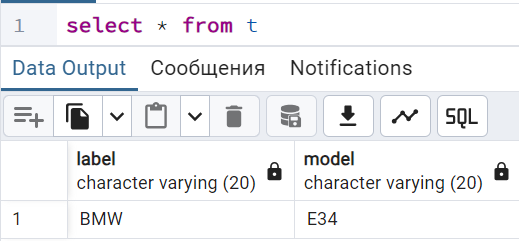
**Рисунок 3.24 - операция деления: создание отношения TT**



**Рисунок 3.25 - операция деления: создание отношения T2**



**Рисунок 3.26 - операция деления: создание отношения T**



**Рисунок 3.27 - операция деления: вывод результата**

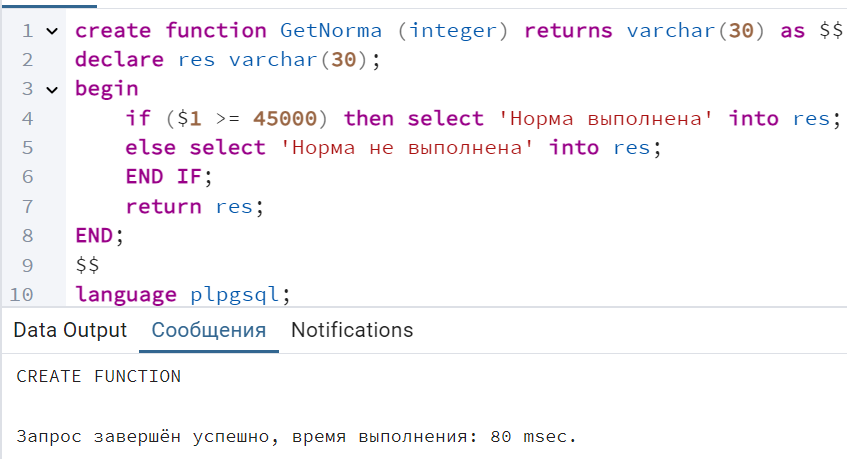
3.4 Создание триггеров и функций

Хранимые процедуры, функции и триггеры вводятся в базу данных для

обеспечения бизнес-логики приложения на уровне серверной его компоненты. Обычно хранимые процедуры и функции представляют собой утилиты, которые определенным образом обрабатывают данные или реализуют достаточно сложный алгоритм вычисления некоторых показателей.

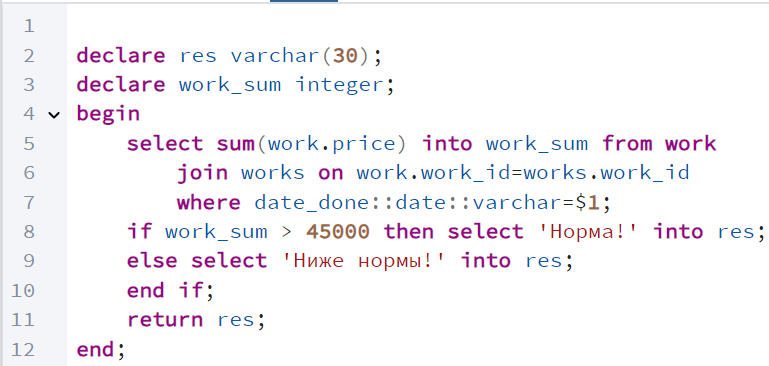
Триггеры – это частный случай хранимой процедуры, который выполняется автоматически при выполнении команд обновления данных (INSERT, DELETE, UPDATE). Триггеры привязываются к конкретным таблицам базы данных. Для каждой команды должны быть свои триггеры.

Создадим функцию, которая проверяет выполнена ли норма за день:



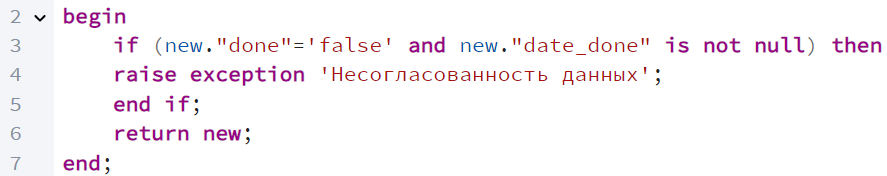
**Рисунок 3.28 - создание функции GetNorma**

Доработаем функцию так, чтобы она определала выполнена ли норма в день по указанной дате:

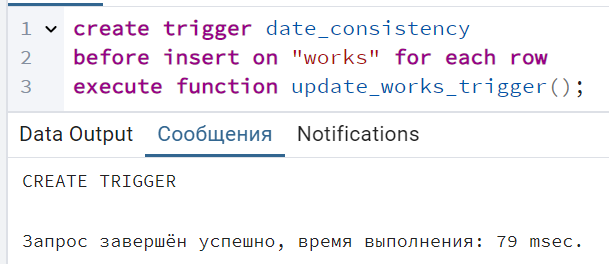


**Рисунок 3.29 - доработка функции GetNorma**

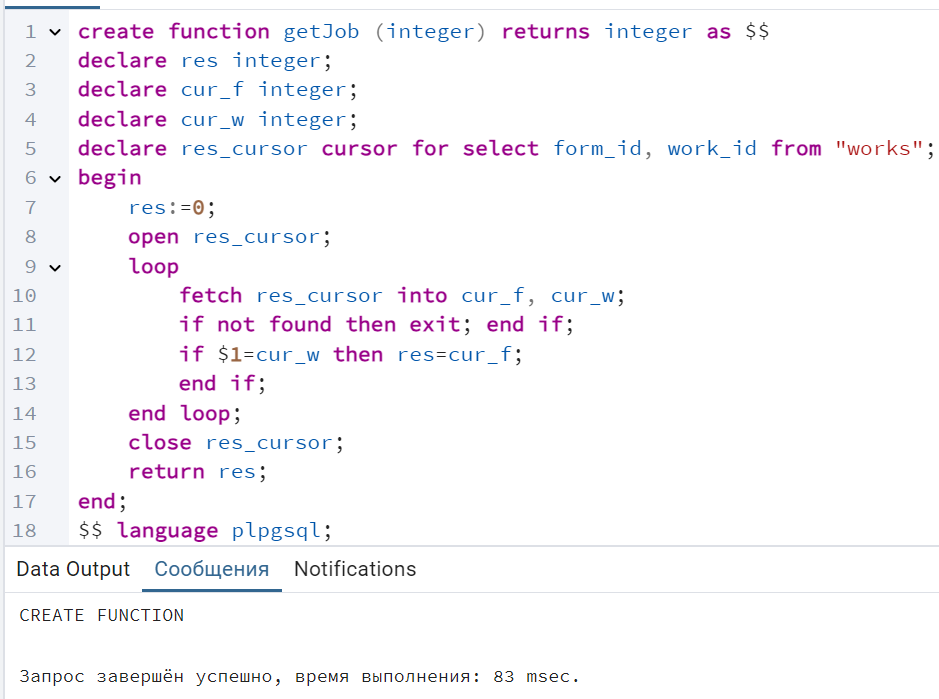
Создадим триггер, который проверяет согласованность данных при вставке записей, а именно: поле «Работа выполнена» равно «False», но при этом в кортеже стоит дата выполнения этой работы:



**Рисунок 3.30 - создание триггерной функции**



**Рисунок 3.31 - создание триггера**



**Рисунок 3.32 - создание курсора**

ПРАКТИЧЕСКАЯ РАБОТА 4

4.1 Оконные функции

Окно для функции определяется посредством обязательной инструкции OVER(). Откроем окно при помощи OVER() и найдём средную цену по заявкам.

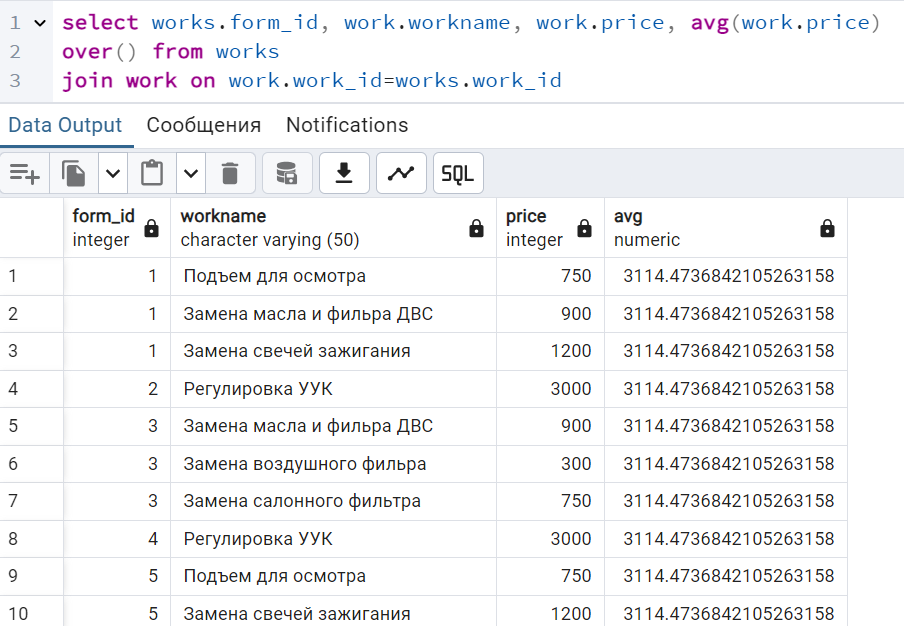


Рисунок 4.1 – Результат открытия окна

Так как мы использовали инструкцию OVER() без предложений, то окном является весь набор данных, без сортировок и группировок. В результате мы видим новый столбец avg, который для каждой строки выводит одно и то же значение — 3114.473, что является средним количеством потраченной валюты.

Теперь применим инструкцию PARTITION BY, которая используется для разделения таблицы на логические подгруппы данных, называемые разделами (partitions). Разделение происходит на основе одного или нескольких столбцов.

Результат работы команды показан на Рисунке 4.2.

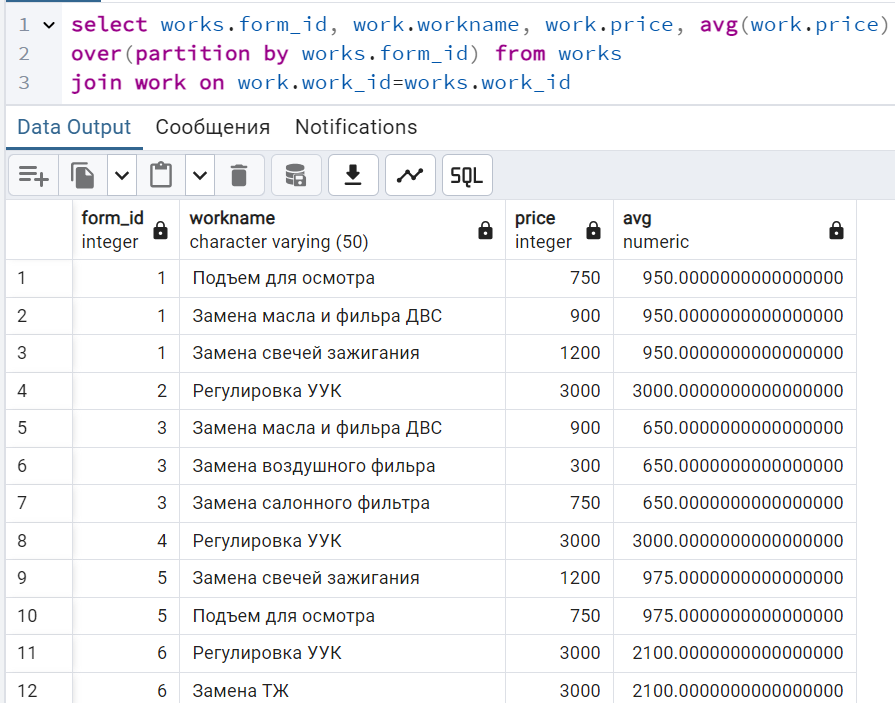


Рисунок 4.2 - Применение инструкции группировки PARTITION BY

В результате получаем таблицу, сгруппированную по заявкам.

Теперь разберемся с сортировками при использовании оконных функций. Хоть нам и знаком ORDER BY, в рамках оконной функции оно пусть и выполняет сортировку, но имеет несколько иное назначение.

Результат работы команды показан на Рисунке 4.3.

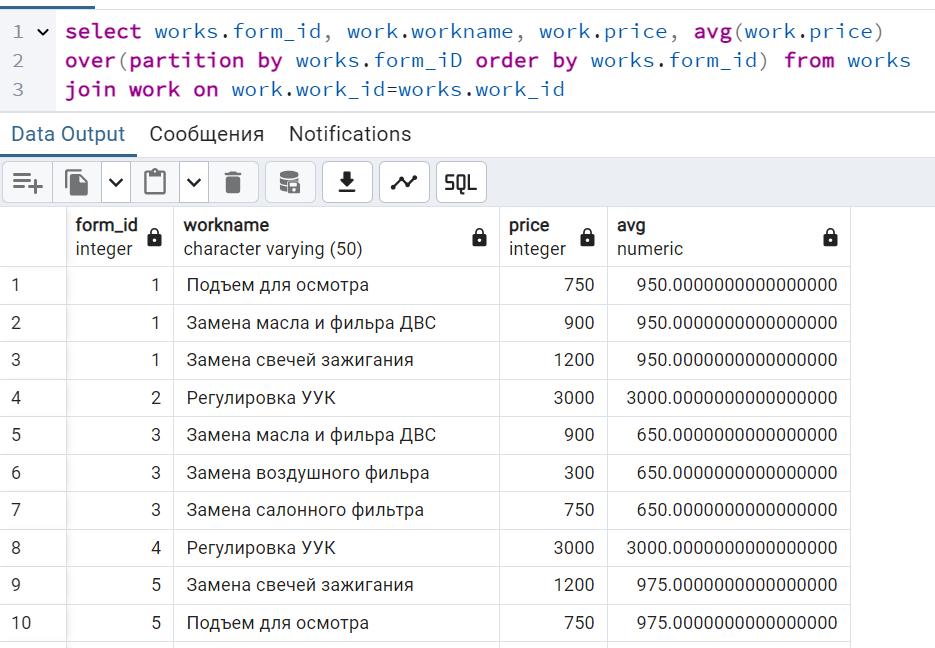


Рисунок 4.3 – Результат применения сортировки

Теперь разберемся с инструкциями ROWS или RANGE.

Инструкция ROWS позволяет ограничить строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей.

Инструкция RANGE же, в свою очередь, работает не со строками, а с диапазоном строк в инструкции ORDER BY. То есть в RANGE одной строкой могут считаться несколько реальных строк, одинаковых по рангу.

Обе инструкции всегда используются только вместе с ORDER BY.

Так же для ограничения строк посредством ROWS или RANGE можно использовать следующие ключевые слова:

* UNBOUNDED PRECEDING — указывает, что окно начинается с первой строки группы;
* UNBOUNDED FOLLOWING – с помощью данной инструкции можно указать, что окно заканчивается на последней строке группы;
* CURRENT ROW – инструкция указывает, что окно начинается или заканчивается на текущей строке;
* BETWEEN «граница окна» AND «граница окна» — указывает нижнюю и верхнюю границу окна;
* «Значение» PRECEDING – определяет число строк перед текущей строкой (не допускается в предложении RANGE);
* «Значение» FOLLOWING — определяет число строк после текущей строки (не допускается в предложении RANGE).

Результат работы команды показан на Рисунке 4.4.

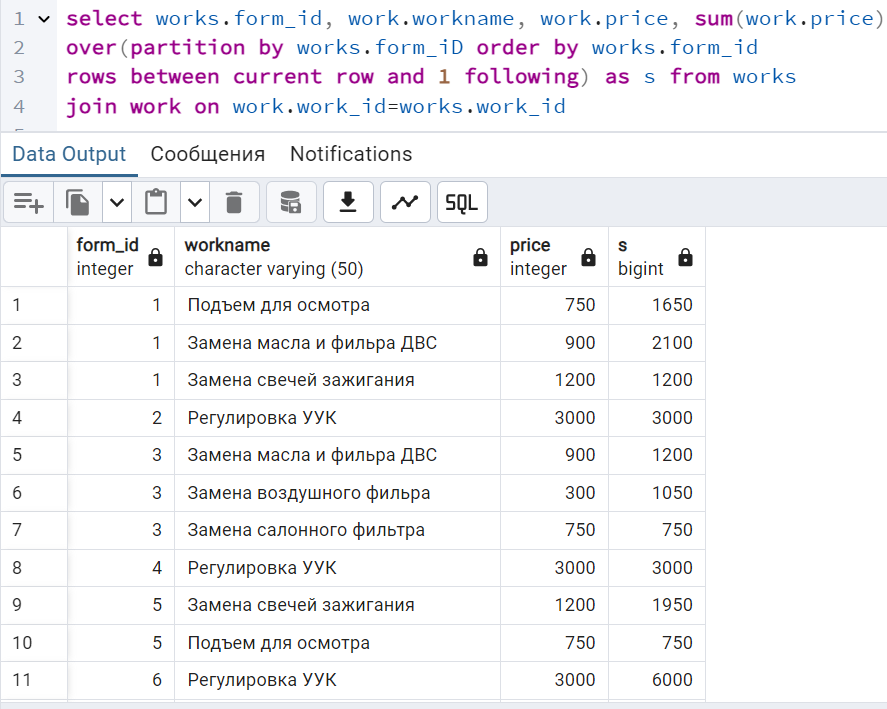


Рисунок 4.4 - Результат применения инструкция ROWS и RANGE

Команда RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING определяет, что все строки от самой первой и до самой последней строки будут включены в вычисление среднего количества потраченной валюты внутри группы.

Команда ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING определяет, что суммироваться будет потраченная валюта из предыдущей строки, текущей и следующей.

Команда RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW определяет, какие строки будут учтены при поиске минимального количества потраченной валюты. В данном случае, все строки от самой первой и до текущей строки для каждой группы будут учтены, чтобы найти минимальное количество потраченной валюты.

4.2 Агрегатные функции

Агрегатными функциями называются функции, которые выполняют арифметические вычисления на наборе данных и возвращают итоговое значение.

К агрегатным функциям относятся:

• COUNT() — вычисляет количество значений в столбце (не учитывает значения NULL);

• SUM() — возвращает сумму значений в столбце;

• AVG() — определяет среднее значение в столбце;

• MAX() — определяет максимальное значение в столбце;

• MIN() — определяет минимальное значение в столбце.

Воспользуемся функцией COUNT() для подсчета того, сколько раз была выполнена та, или иная работа.

Результат работы команды показан на Рисунке 4.5.

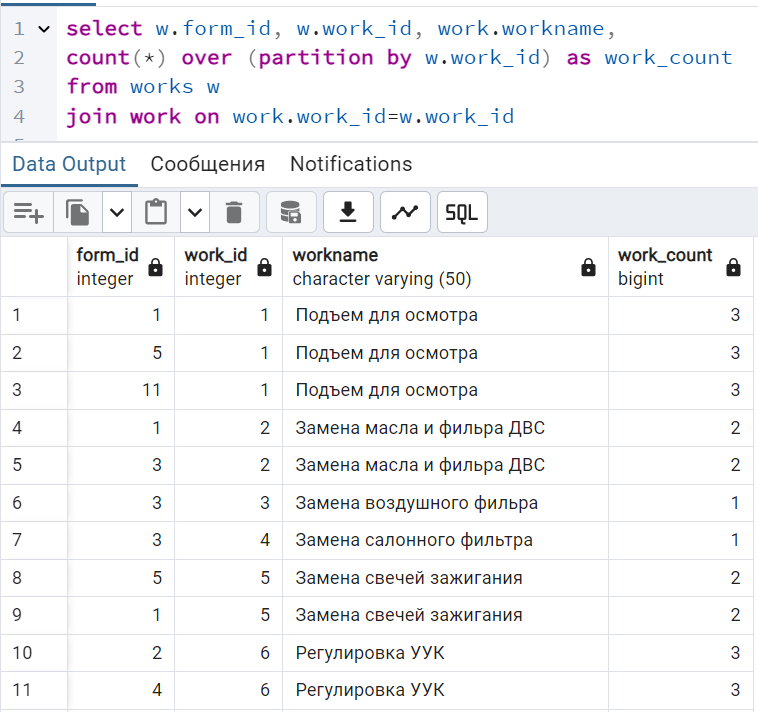
4

Рисунок 4.5 - Пример использования оконной агрегатной функции COUNT

Воспользуемся функцией SUM() для нахождения суммы для каждой работы.

Результат работы команды показан на Рисунке 4.6.

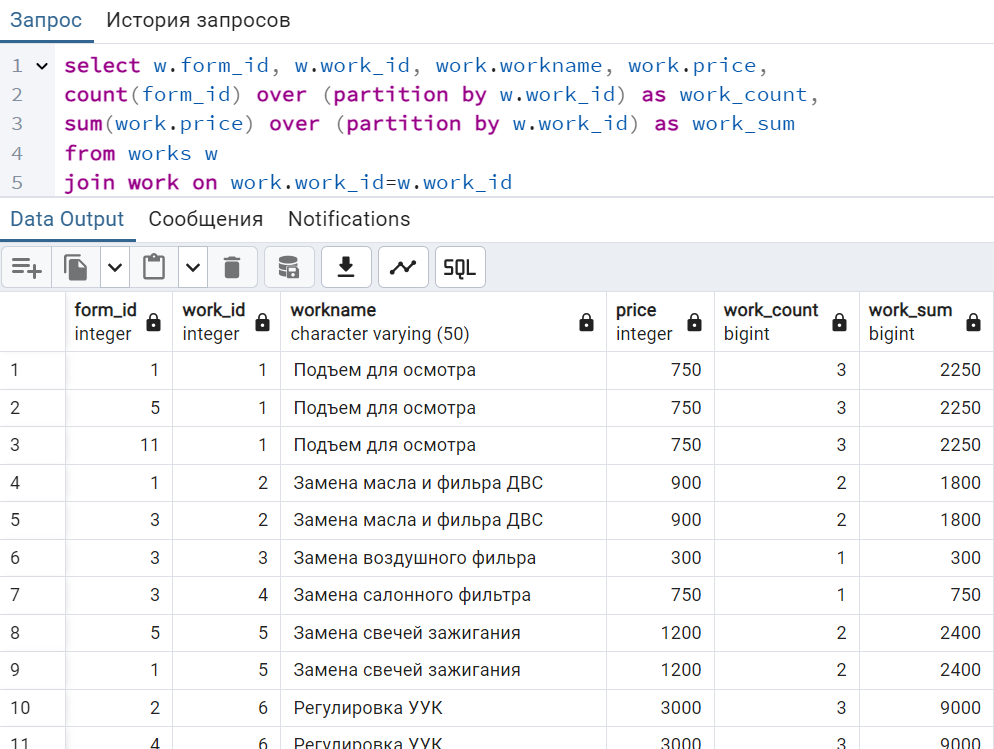


Рисунок 4.6 - Пример использования оконной агрегатной функции SUM

Воспользуемся функцией AVG() и узнаем средний год автомобилей по марке.

Результат работы команды показан на Рисунке 4.7.

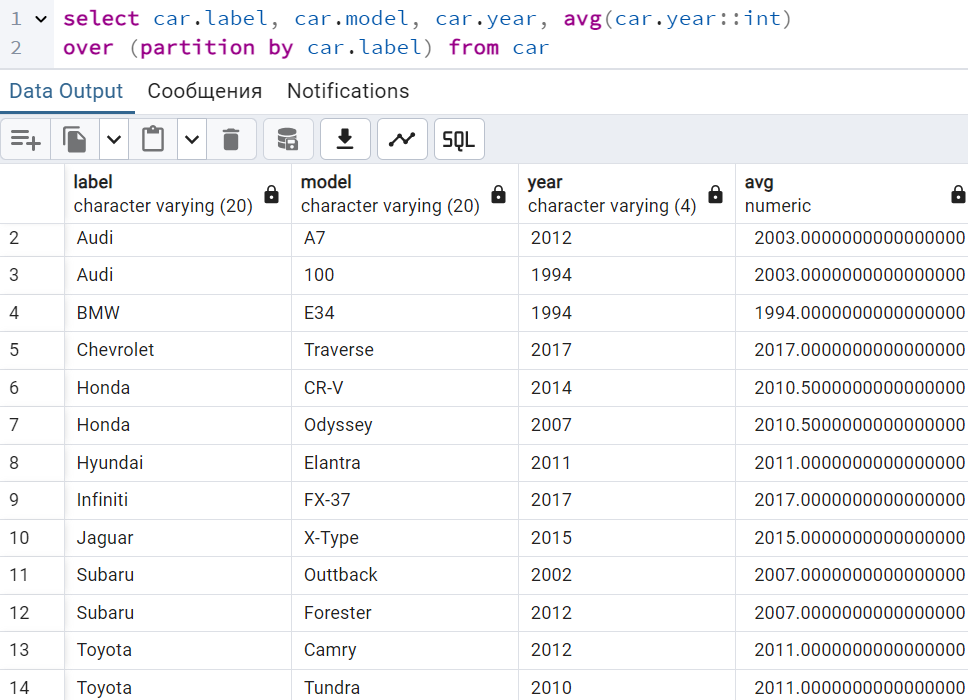


Рисунок 4.7 - Пример использования оконной агрегатной функции AVG

Воспользуемся функцией MIN() для получения максимального года выпуска автомобиля для каждой марки.

Результат работы команды показан на Рисунке 4.8.

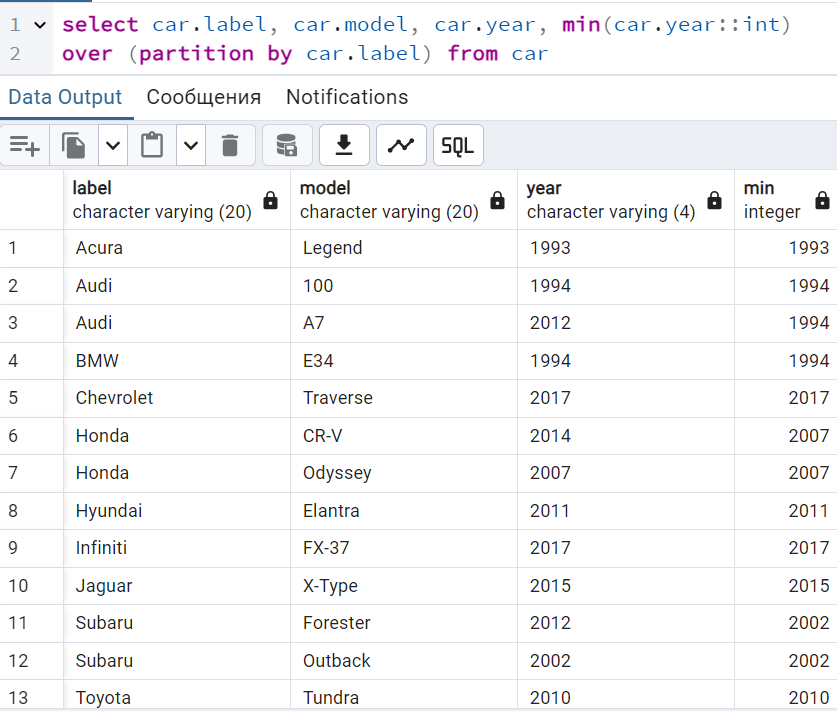


Рисунок 4.8 - Пример использования оконной агрегатной функции MIN

Воспользуемся функцией MAX() для получения максимального года выпуска автомобиля для каждой марки.

Результат работы команды показан на Рисунке 4.9.

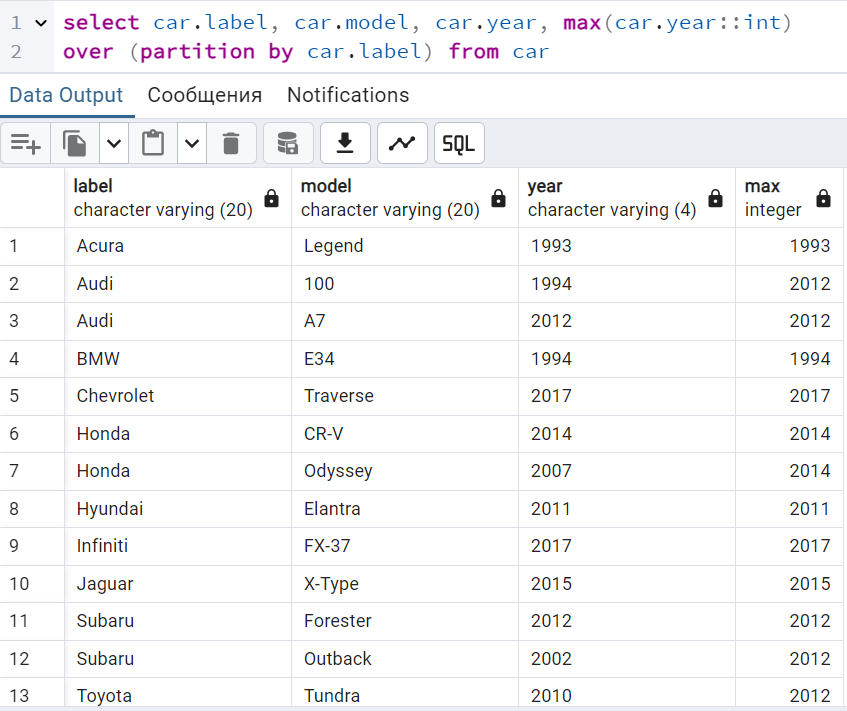


Рисунок 4.9 - Пример использования оконной агрегатной функции MAX

4.3 Ранжирующие функции

Ранжирующие функции — это функции, которые определяют ранг для каждой строки в окне. Например, их можно использовать для присвоения порядковых номеров или для составления рейтинга. К ранжирующим функциям относятся:

• ROW\_NUMBER() — функция возвращает номер строки и используется для нумерации;

• RANK() — функция возвращает ранг каждой строки. Данная функция в том числе анализирует данные и, в случае нахождения одинаковых – возвращает одинаковый ранг с пропуском следующего значения (например, два различных товара были проданы на одинаковую сумму по итогам месяца. При использовании данной функции для оценки ранга продаж за месяц обоим товарам будет выставлен ранг 1, а следующий за ними товар получит ранг 3);

• DENSE\_RANK() — так же, как и прошлая функция, возвращает ранг каждой строчки, но в отличие от функции RANK, следующий ранг пропускаться не будет;

• NTILE() — это функция, которая позволяет определить, к какой группе относится текущая строка. Количество групп задаётся в скобках.

Внутри оконной функции сгруппируем оплаты по названию валюты и отсортируем по методу оплаты, воспользовавшись ранжирующей функцией ROW\_NUMBER(), которая вернет номер строки внутри группы.

Результат работы команды показан на Рисунке 4.10.

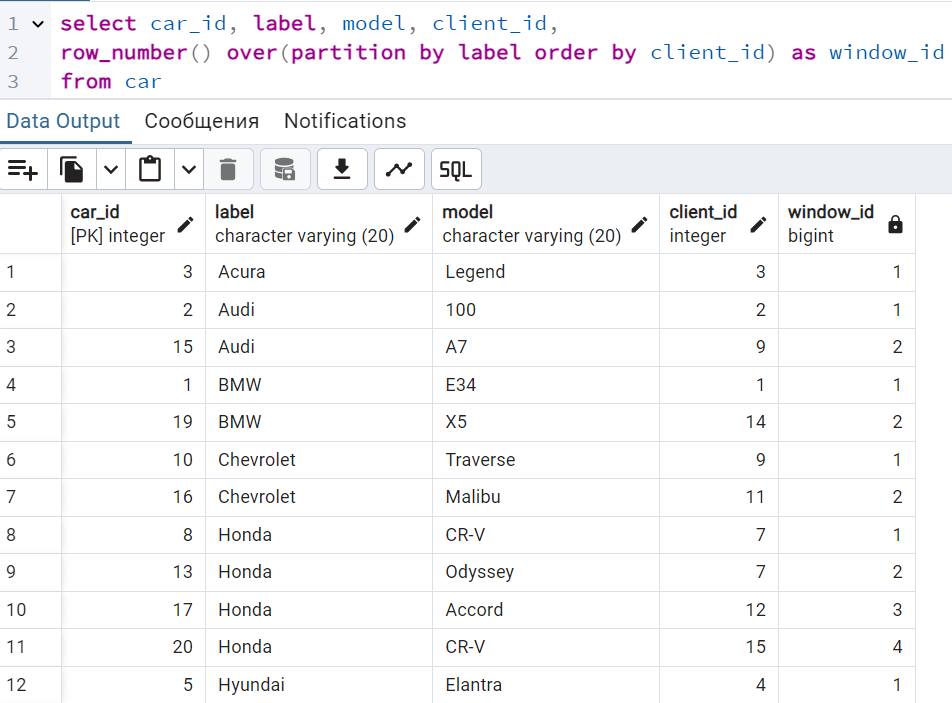


Рисунок 4.10 - Пример использования оконной ранжирующей функции ROW\_NUMBER

Функция RANK() возвращает ранг строки в окне.

Результат работы команды показан на Рисунке 4.11.

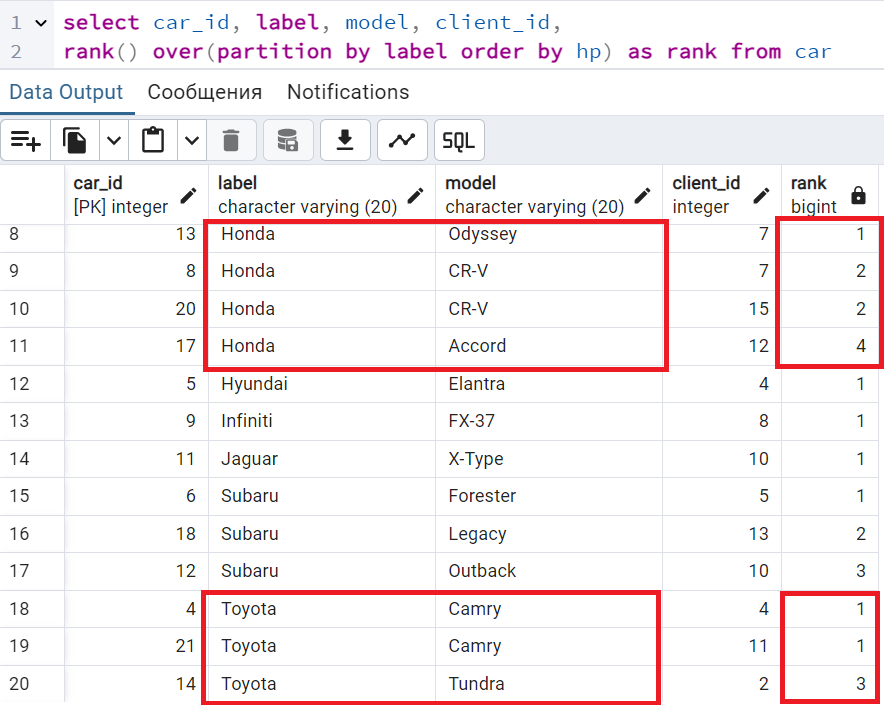


Рисунок 4.11 - Пример использования оконных ранжирующих функций RANK

Функция DENSE\_RANK возвращает ранг строки в окне без пропуска ранга для дубликатов.

Результат работы команды показан на Рисунке 4.12.

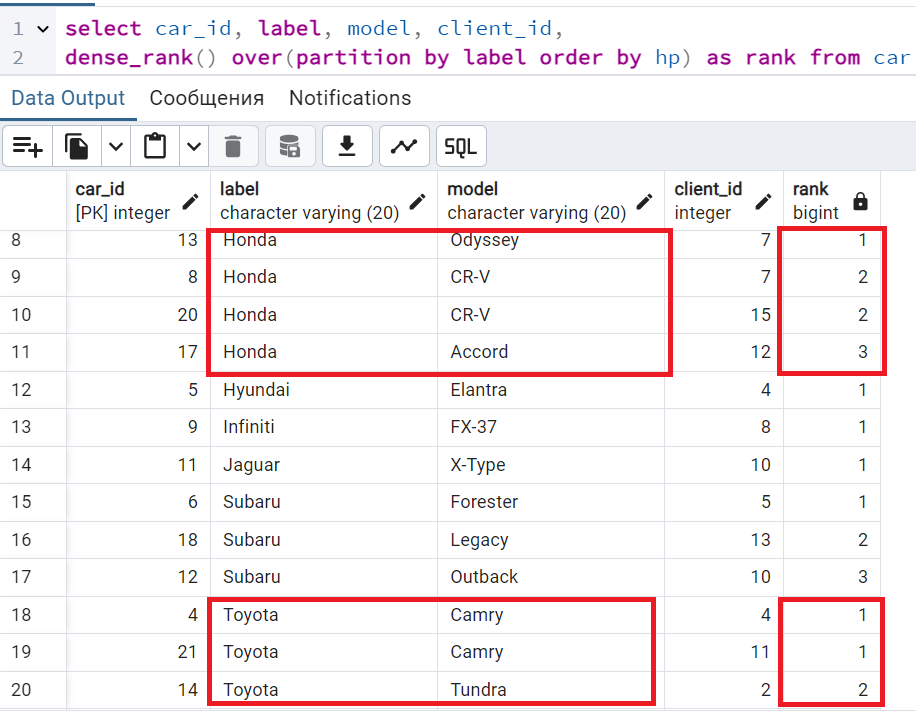


Рисунок 4.12 - Пример использования оконной ранжирующей функций DENSE\_RANK

Функция NTILE «делит» значения на несколько равных частей, кол-во которых указывается в параметре функции.

Результат работы команды показан на Рисунке 4.13.

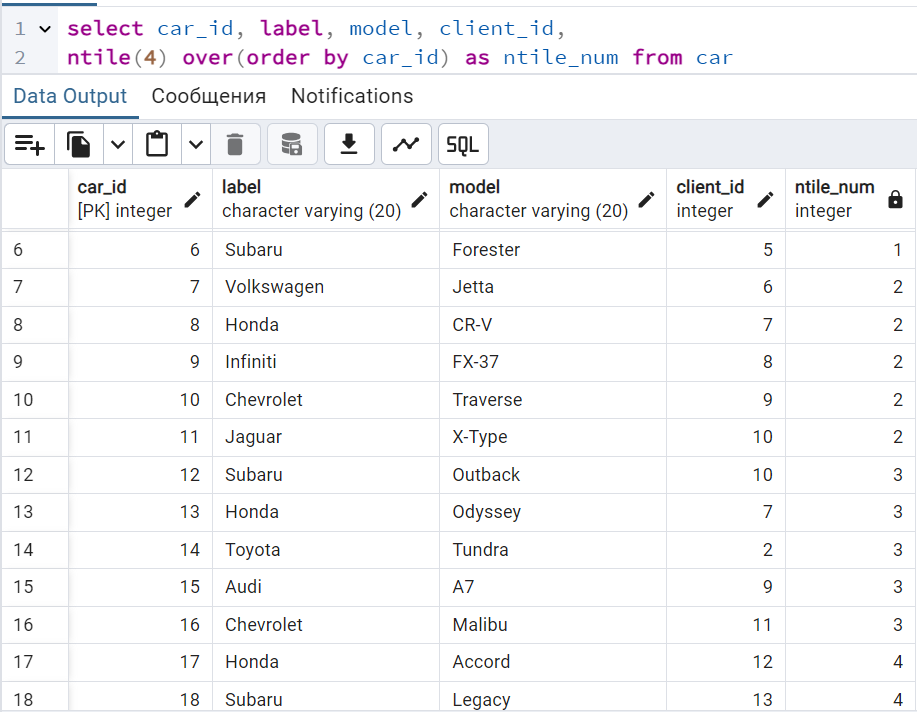


Рисунок 4.12 - Пример использования оконной ранжирующей функций NTILE

4.4 Функции смещения

Функции смещения — это функции, которые позволяют перемещаться и обращаться к разным строкам в окне относительно текущей строки, а также обращаться к значениям в начале или в конце окна. К функциям смещения относятся:

• LAG() и LEAD() — функция LAG обращается к данным из предыдущей строки окна, а LEAD к данным из следующей строки. Функцию можно использовать для сравнения текущего значения строки с предыдущим или следующим. Имеет три параметра: столбец, значение которого необходимо вернуть, количество строк для смещения (по умолчанию это 1) и значение, которое необходимо вернуть, если после смещения возвращается значение NULL;

• FIRST\_VALUE() и LAST\_VALUE() — функция FIRST\_VALUE позволяет получить первое значение в окне, а LAST\_VALUE, соответственно, последнее значение.

Напишем запрос для получения предыдущей и следующей модели внутри группы по марке при помощи функций смещения LAG() и LEAD().

Результат работы команды показан на Рисунках 4.13. и 4.14



Рисунок 4.13 - Пример использования оконных функций смещения LAG

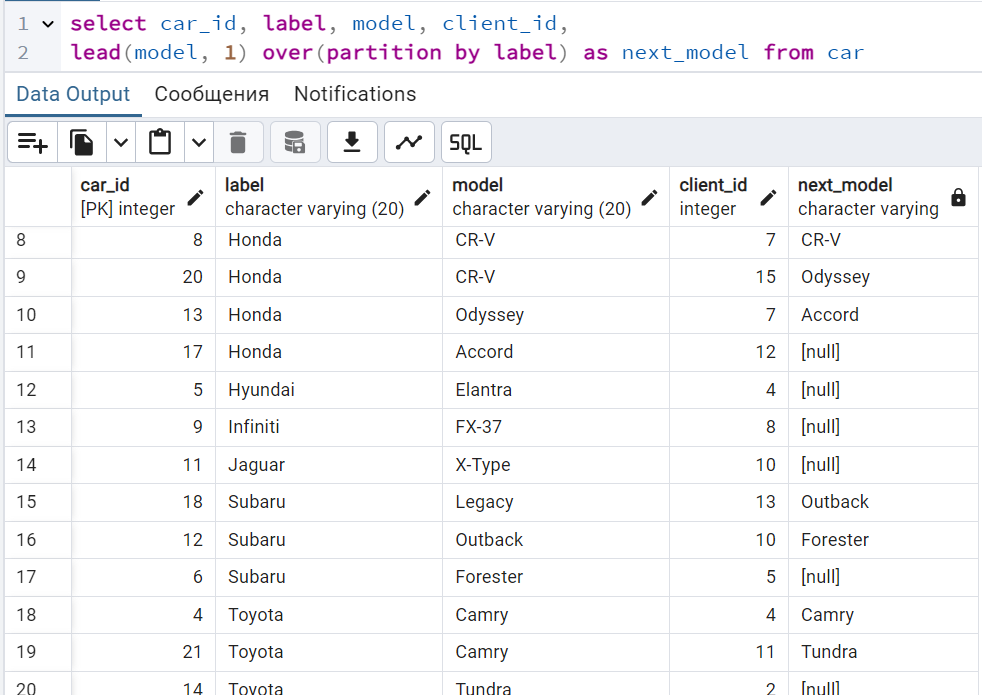


Рисунок 4.14 - Пример использования оконных функций смещения LEAD

Напишем запрос для первой и последней модели в марке FIRST\_VALUE() и LAST\_VALUE().

Результат работы команды показан на Рисунках 4.15 и 4.16.

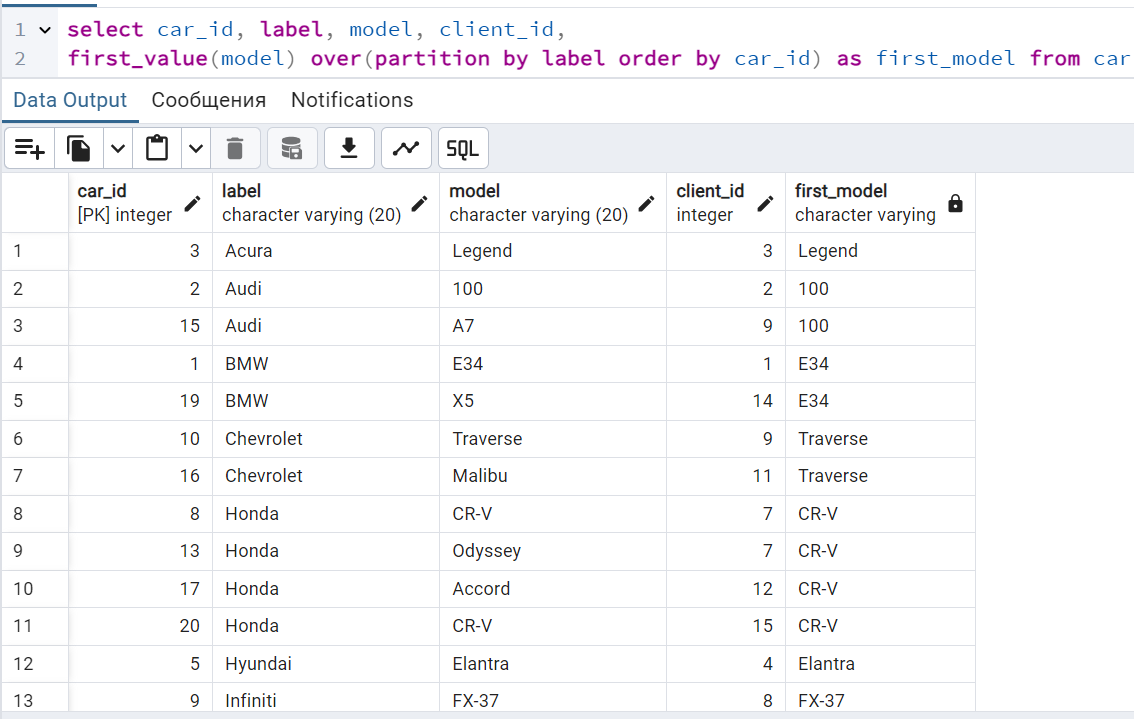


Рисунок 4.15 - Пример использования оконных функций смещения FIRST\_VALUE

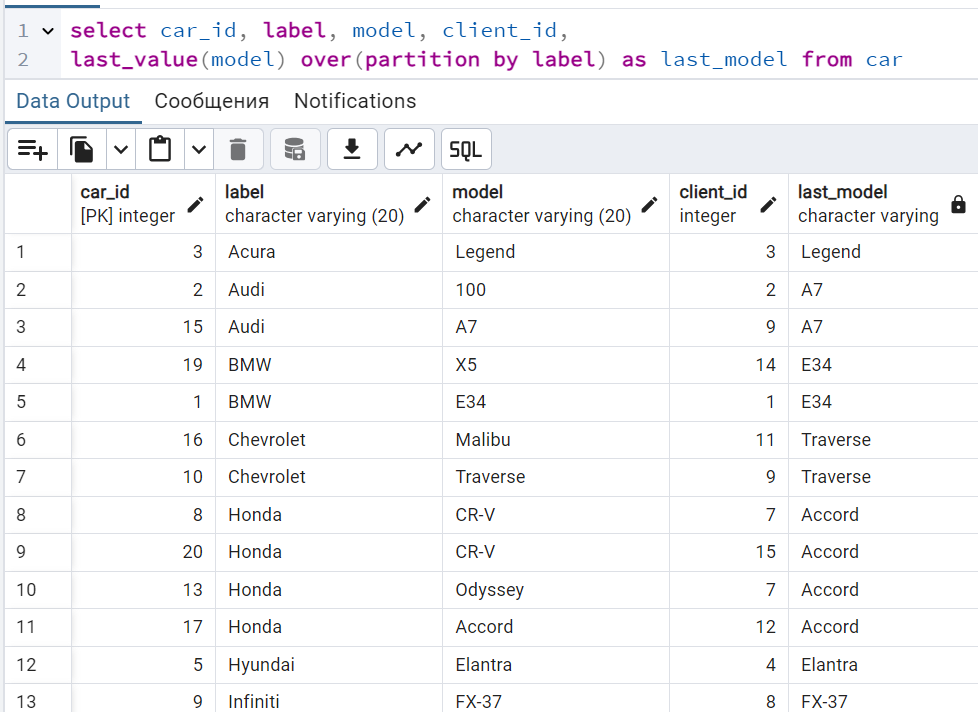


Рисунок 4.16 - Пример использования оконных функций смещения LAST\_VALUE

ВЫВОДЫ

Была успешно нормализована схема базы данных до 3 нормальной формы. После этого база данных была реализована в СУБД Mysql и PostgreSQL. Создана резервная копия БД. Затем были рассмотрены основные операции манипулирования данными: создание, просмотр, обновление, удаление. В конце были реализованы основные операции реляционной алгебры, были написаны несколько функций, процедур и триггеров. Реализованы оконные функции.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Методические указания по курсу «Разработка баз данных» [Электронный ресурс]. – URL: <https://online-edu.mirea.ru/mod/folder/view.php?id=263588> (дата обращения 18.11.2024)
2. Лекции по курсу «Разработка баз данных» [Электронный ресурс]. – URL: <https://online-edu.mirea.ru/course/view.php?id=7699> (дата обращения 18.11.2024)