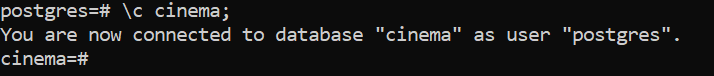
**Задание №1**

Создание предметной БД в среде PostgreSQL Цель работы: предварительно: выбрать предметную область, построить логическую модель БД. БД должна содержать от 5 таблиц, в каждой 3-7 полей разных типов, от 10 строк-записей. Создать базу и таблицы в Postgresql с помощью команд CREATE и INSERT. Рабочие экраны (скрины) представить в отчете. По команде SELECT \* получить данные из базы.

Для создания базы данных выполним команду CREATE DATABASE;

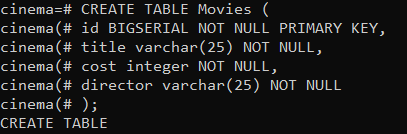


База данных создана об этом говорит надпись CREATE DATABASE. Для того, чтобы начать работу с БД, нужно подключиться к ней, для этого мы вводим \c cinema и видим надпись, которая говорит, что мы успешно подключись к БД.



Для создания таблиц в языке SQL служит команда CREATE TABLE. Мы будем создавать базу данных кинотеатр. Начнем с создания структур таблиц нашей базы. Первой создадим таблицу «Фильмы». Для этого используем упрощенный синтаксис: имя-поля тип-данных [ограничения целостности],

Таблица «Фильмы»



Далее по аналогии создаем остальные таблицы

Таблица «Филиалы»

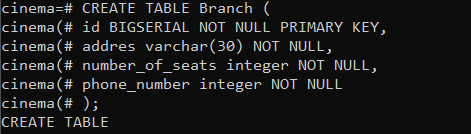


Таблица «Зритель»

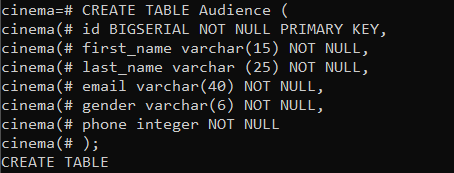


Таблица «Работники»

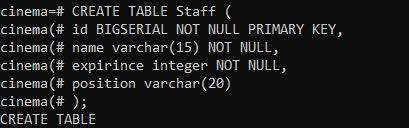
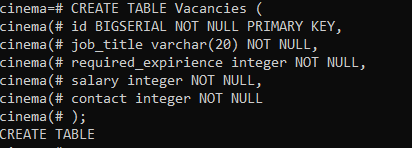


Таблица «Вакансии»



В результате работы создана структура БД. Таблицы созданы и готовы к

заполнению.

Заполним таблицы данными, для выполнения этой операции служит

команда INSERT. Ее упрощенный формат таков:

INSERT INTO имя-таблицы [( имя-атрибута, имя-атрибута, ... )]

VALUES ( значение-атрибута, значение-атрибута, ... );

Давайте добавим строки в таблицу Product. Для атрибутов символьных

типов данных одинарные кавычки обязательны, а для числовых типов

кавычки использовать не нужно.

Таблица «Фильмы»

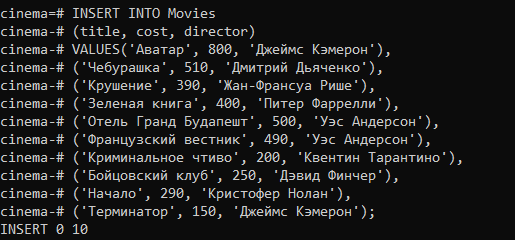


Таблица «Филиалы»

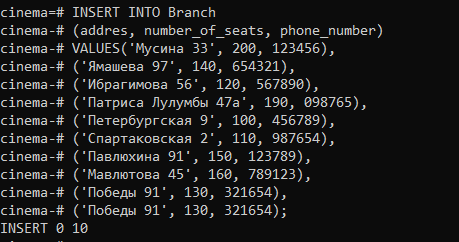


Таблица «Зритель»



Таблица «Работники»

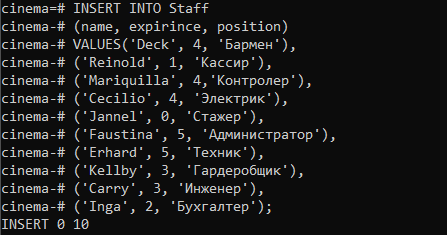
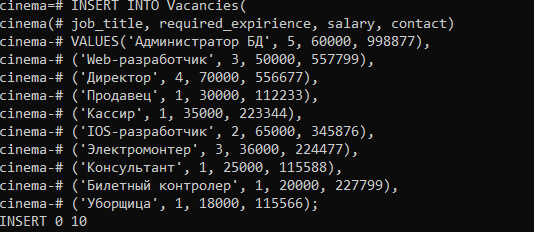
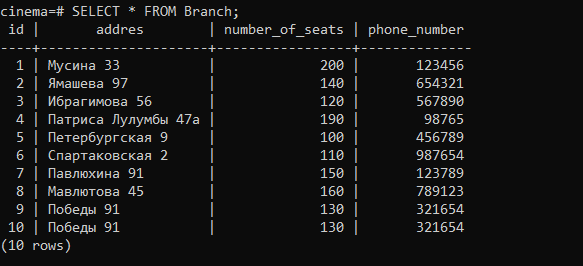


Таблица «Вакансии»

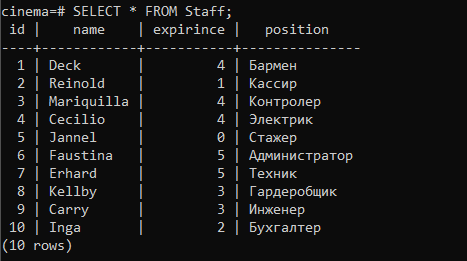


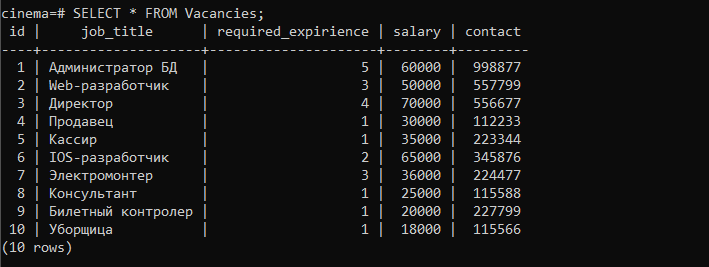
Вывод таблиц











**Задание №2**

Цель работы: загрузить демобазу «Авиаперевозки» (мини-версия) с официального сайта postgrespro.ru.

Выполнить запросы по типам данных: числовой, символьный, логический, дата/время, JSON по примерам\* .

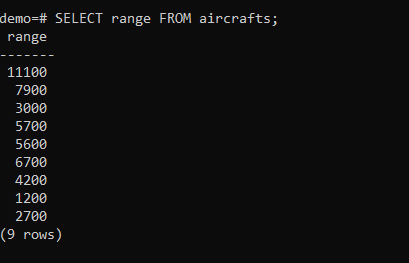
Разработать и выполнить подходящие по логике/тематике запросы (по несколько запросов для каждого типа данных )для предметной базы данных. Зафиксировать скрины результатов в отчете.

Источники: основной-учебник Моргунов гл. 4, стр. 51

Выполним запрос по числовому типу данных из примера в учебнике при

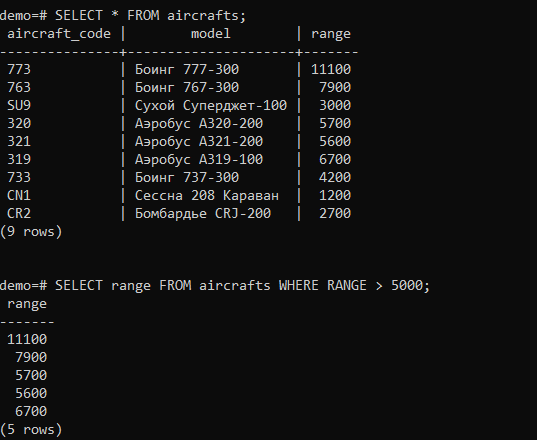
помощи команды SELECT.

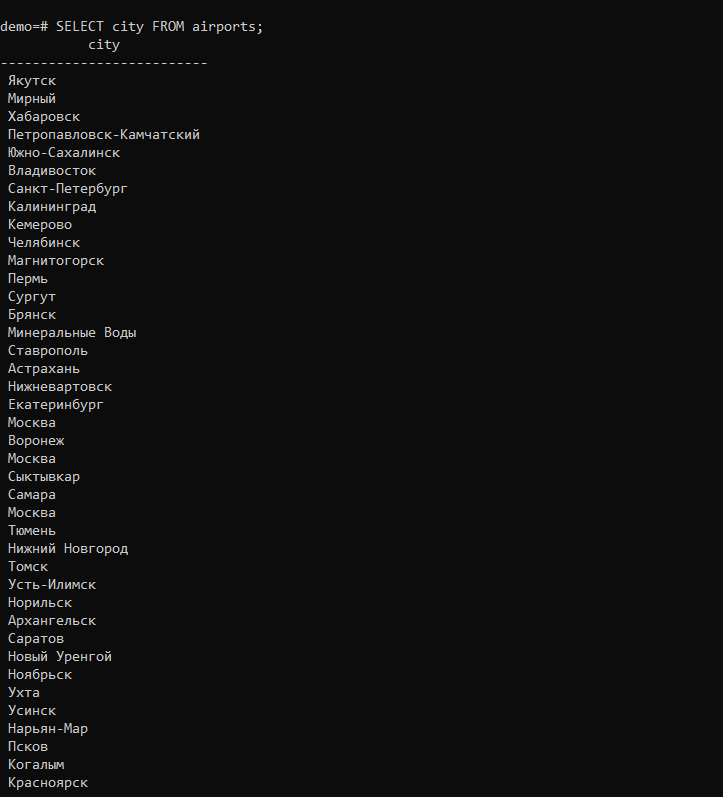
Посмотрим столбец числовых данных range в таблице aircrafts.



Выполним запрос по числовому типу данных, определим все самолёты

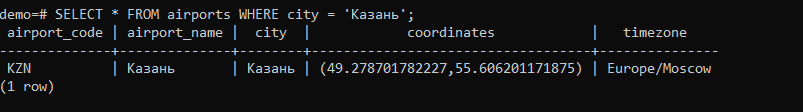
которые имеют range составляет больше 5000.



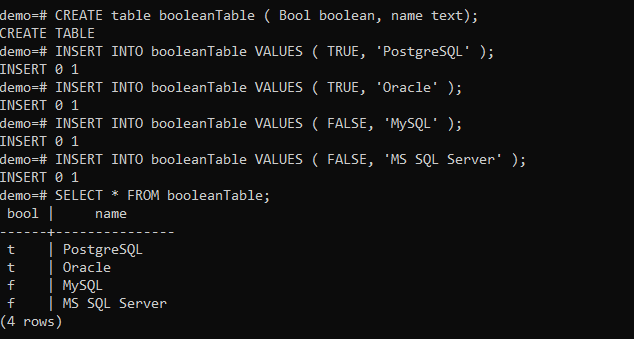


Выполним запрос по символьному типу данных.

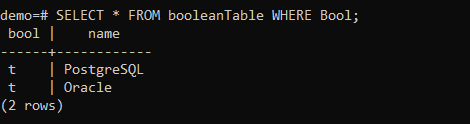
Выведем информацию по городу Казань:



По логическому типу: создадим таблицу booleanTable и добавим в нее несколько строк со следующими значениями -

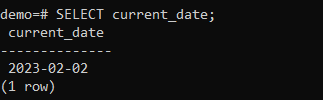


В условии WHERE для проверки логических значений можно не писать выражение WHERE bool = 'yes', а достаточно просто указать имя столбца, содержащего логическое значение: WHERE bool. Поэтому выберем СУБД с открытым исходным кодом с помощью команды.

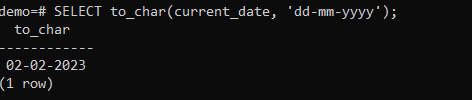


Запрос по типу данных «дата/время»

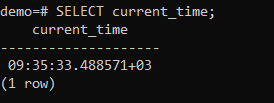
В PostgreSQL предусмотрен целый ряд функций для работы с датами и временем. Например, для получения значения текущей даты служит функция current\_date. Ее особенностью является то, что при ее вызове круглые скобки не используются.



Если нам требуется вывести дату в другом формате, то для разового преобразования формата можно использовать функцию to\_char, например:

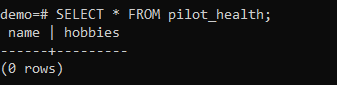


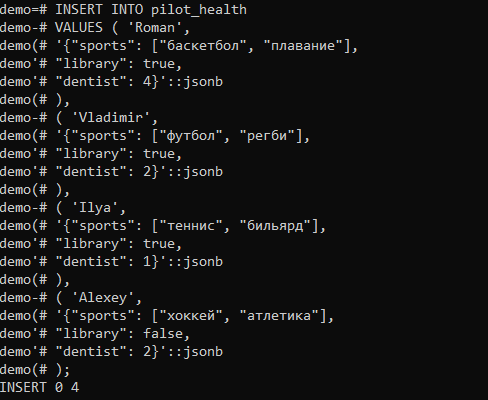
Попробуем получить текущие время с помощью функции current\_time.При её вызове круглые скобки использовать не нужно.



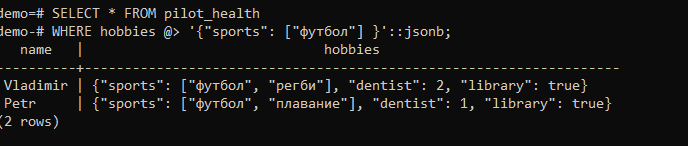
Тип данных «Json»

Для иллюстрации использования типов JSON обратимся к тематике авиаперевозок. Предположим, что руководство авиакомпании всемерно поддерживает стремление пилотов улучшать свое здоровье, повышать уровень культуры и расширять кругозор. Поэтому разработчики базы данных авиакомпании получили задание создать специальную таблицу, в которую будут заноситься сведения о тех видах спорта, которыми занимается пилот, будет отмечаться наличие у него домашней библиотеки, а также количество посещений стоматологической клиники.





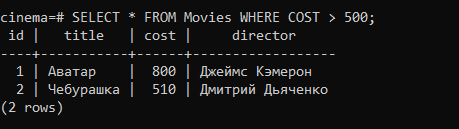
Как видно, при выводе строк из таблицы порядок ключей в JSON-объектах не был сохранен. Предположим, что нужно сформировать футбольный турнир нашей авиакомпании для участия в мероприятиях. Мы можем выбрать всех людей, кто занимается теннисом таким способом:

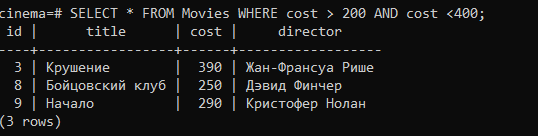


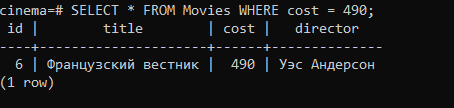
**Задание №3**

Задание заключается в создании типовых запросов для предметной базы данных. В отчете представить результаты работы не менее 10 различных запросов.

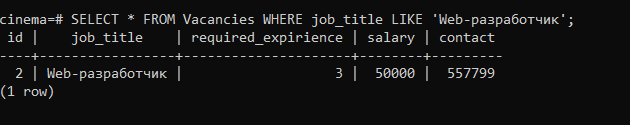
Начнем с различных условий отбора строк в предложении WHERE. Эти условия могут конструироваться с использованием следующих операторов сравнения: =, < >, >, > =, < =.

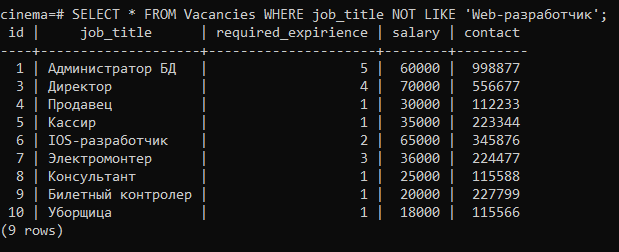




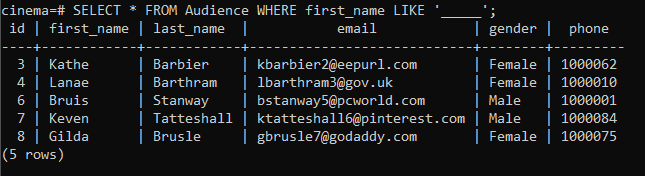


Оператор LIKE:

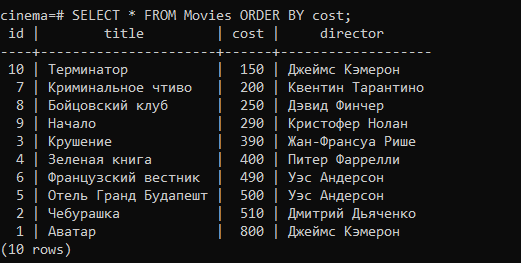


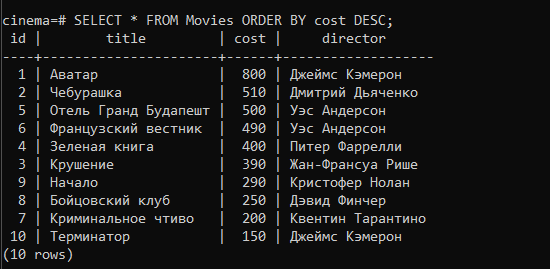


Найдем имена зрителей, у которых имени слова – 5 символов

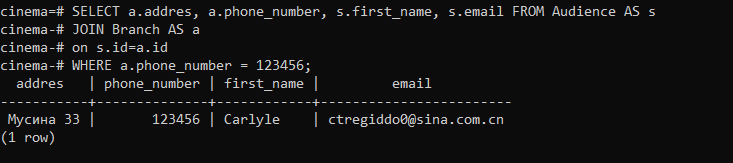


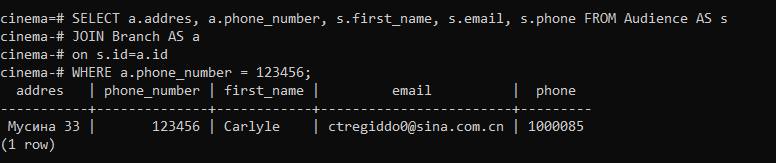
Для упорядочения строк служит предложение ORDER BY, если мы захотим разместить цены товаров в порядке возрастания



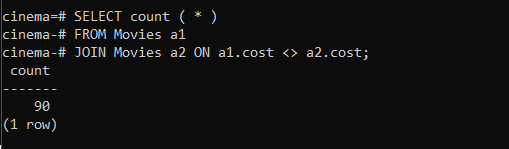
В порядке убывания: 

В тех случаях, когда информации, содержащейся в одной таблице, недостаточно для получения требуемого результата, используют соединение (join) таблиц. Выберем филиал с номером телефона 123456, и узнаем его адрес и имя зрителя, который посетил филиал



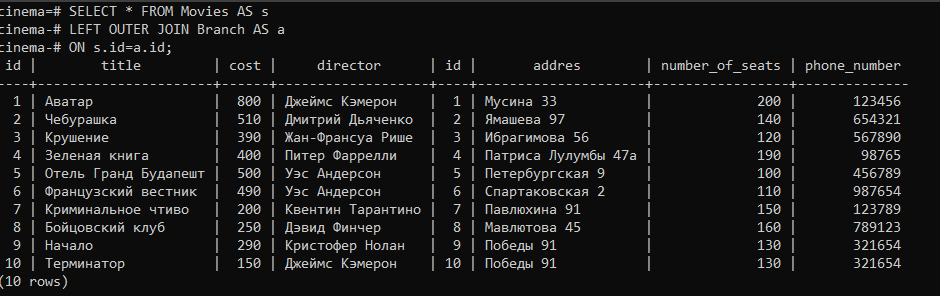
Можем добавить номер телефона клиента 

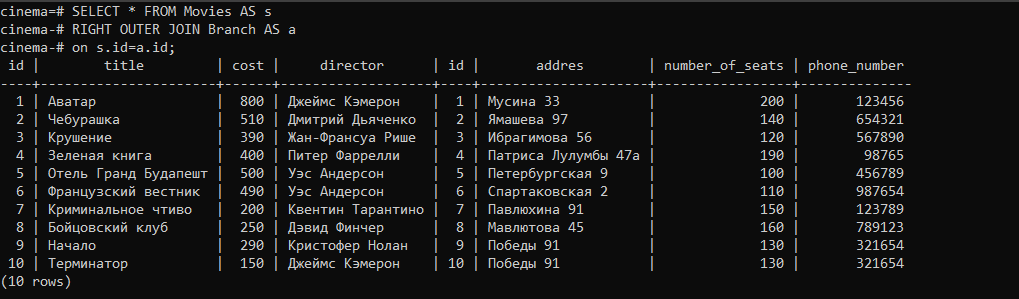
Во втором варианте запроса мы используем соединение таблиц на основе неравенства значений атрибутов. Тем самым мы перенесли условие отбора результирующих строк из предложения WHERE в предложение FROM.



Теперь обратимся к так называемым внешним соединениям.

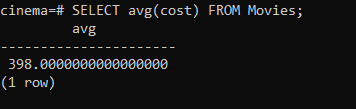
Левое внешнее соединение:



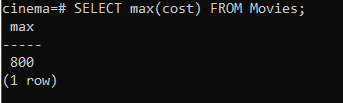
Кроме левого внешнего соединения существует также и правое внешнее соединение RIGHT OUTER JOIN. Комбинацией этих двух видов внешних соединений является полное внешнее соединение — FULL OUTER JOIN. Используем правое внешнее соединение: 

Среди множества функций, имеющихся в PostgreSQL, важное место занимают агрегатные функции

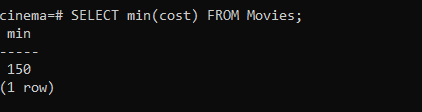
Для расчета среднего значения по столбцу используется функция avg (от слова average). SELECT avg( Столбец) FROM Таблица;



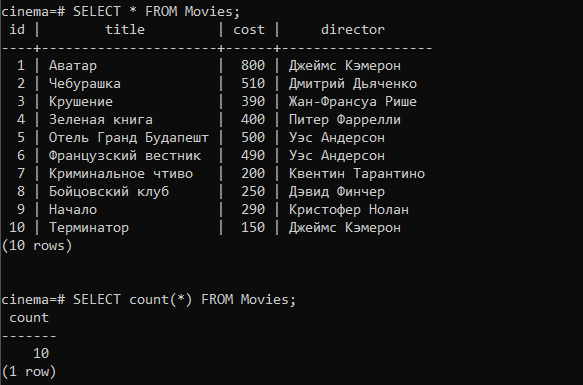
Для получения максимального значения по столбцу используется функция max. SELECT max( Столбец ) FROM Таблица;



Для получения минимального значения по столбцу используется функция min. SELECT min( Столбец ) FROM Таблица;

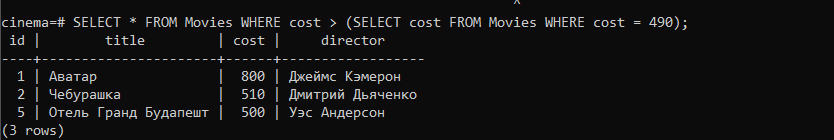


SELECT COUNT(\*) FROM Таблица; Функция COUNT (\*) возвращает количество строк, возвращаемых оператором SELECT.

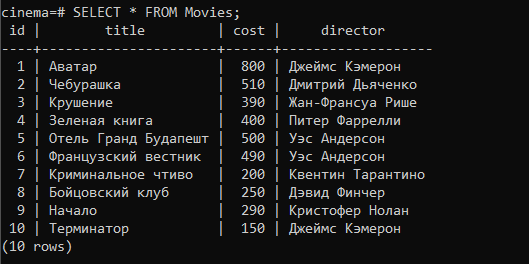


**Задание №4**

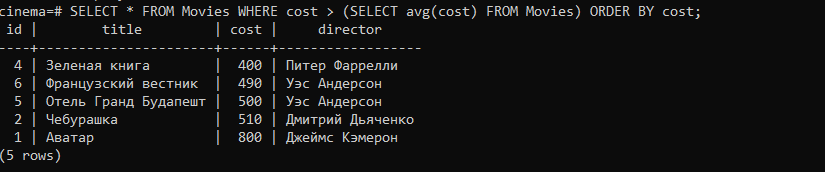
Определим фильм, цена которого выше 490. СУБД сначала выполнит подзапрос, в результате чего получит цену товара, затем поместит значение во внешний подзапрос и выполнит его:



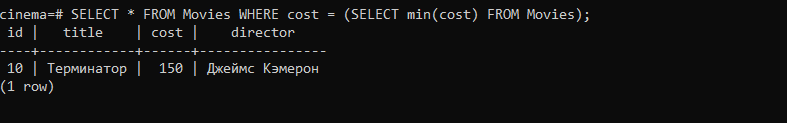
Проверяем



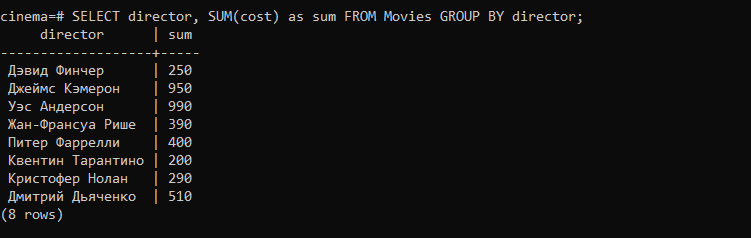
Определим фильмы, цена билетов которых меньше средней цены. Здесь СУБД также выполняет подзапрос и узнаёт цену которые меньше средней цены, а затем помещает значение во внешний запрос и выполняет сравнение со знаком “>”



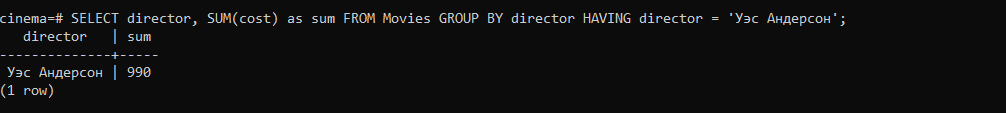
Также выполняем подзапрос и узнаём наименьшую цену билета и выполняем внешний запрос, где приравниваем все значения к значению наименьшей цены.



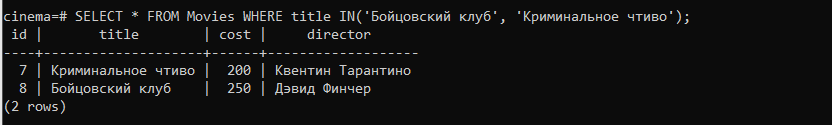
Сгруппируем фильмы по режиссерам и посмотрим на общую стоимость их продукции при помощи GROUP BY

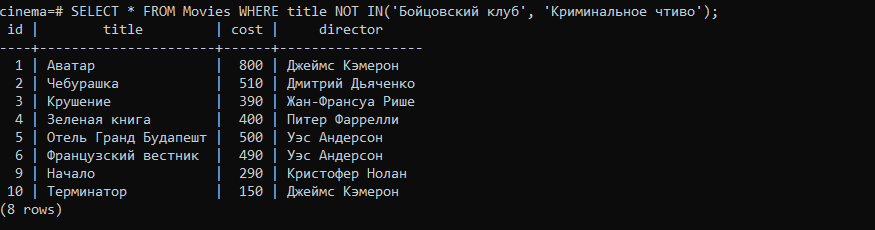


Теперь выведем только фильмы Уэса Андресона при помощи условия HAVING

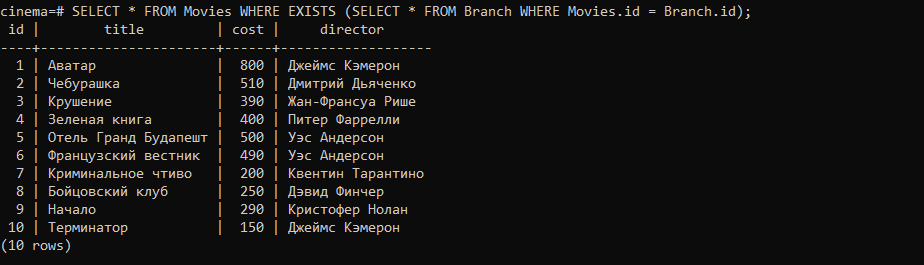


Чтобы проверить соответствие фильмов в таблице нужно использовать IN



Чтобы, наоборот, найти несоответствие нужно использовать NOT IN 

При помощи команды EXISTS выведем из таблицы фильмов только те фильмы, которые совпадают по id



Отобразим все фильмы, id которых меньше 5 при помощи NOT EXISTS

