```
/*
 БД – структурная совокупность взаимосвязанных данных опр. предметной области.
 Функция БД – предоставить единое хранилище для всей информации, относящейся к опр. теме.
 СУБД – система управления базой данных. Программа для хранения, обработки и поиска информации в БД.

 Реляционная БД (relation – связь) – составленная по реляционной модели БД,
 в которой данные, занесенные в таблицы, имеют изначально заданные отношения.
 SQL (Structured Query Language) – язык структурированных запросов для работы с БД.
 SCHEMA (схема) – описание структуры таблиц, полей, ограничений.
 Примеры: PostgreSQL, MySQL, MSSQL, Oracle, MariaDB.

 ACID – набор требований к СУБД. Atomicity, Consistency, Isolation, Durability
  – атомарность (непрерывность). Либо операция выполняется целиком, либо никак (транзакции)
  – согласованность. Данные должны соотв. всем правилам и ограничениям
  – изолированность. Одновременные транзакции не должны пересекаться и влиять друг на друга
  – надежность. Если пришло подтверждение, что данные сохранены, а потом случился сбой, то данные все равно сохранены

 NoSQL БД:
  – ключ-значение (key-value). Redis, Memcached, DynamoDB
  – колоночные (colomnstore). Cassandra, Apache Hbase, ClickHouse
  – документоориентированные (document DB). MongoDB, amazon DynamoDB, CouchDB
  – графовые (graph DB) Dgraph, Neo4j
  – поисковые (search-engine). ElasticSearch, Solr, Algolia
  – временных рядов (time-series). InfluxDB, Prometheus
  – многомодульные (multi-modul). Redis

 DBeaver – клиент для подключения к СУБД.

   Структура реляционной БД:
 -> Кластер (сервер)
 -> Базы данных / роли (учетные записи и групповые политики)
 -> Схемы (табличные пространства)
 -> Отношения (таблицы) / представления / типы данных / домены / ограничения / индексы
 -> Атрибуты (столбцы)
 -> Кортежи (строки)

К БД можно подключаться облачно или удаленно.

primary key (первичый ключ) – уникальный идентификатор строки
 unique (значения уникальны)
 + not null (значение точно должно быть)
 + index (для более быстрой работы)

 Натуральный: данные, которые уже присутствуют (например, номер СНИЛС)
 Суррогатный: целочисленный идентификатор (как правило используют суррогатный)

 Простой: ограничение накладывается на один столбец (например, id)
 Составной: ограничение накладывается на несколько столбцов (например, код города и номер телефона)

foreign key (внешний ключ) – связь между значением в локальной и внешней таблицах.
 – механизм БД, гарантирующий, что значения в удаленных столбцах
 присутствуют как первичных ключи в локальных. Нужен для контроля ссылочной целостности.

 Типы связей между таблицами:
  Один-ко-многим / многие-к-одному
  Один-к-одному
```

Многие-к-многим

check – условие для проверки данных на их корректность

Нормализация – метод проектирования БД. Используется для разработки таблицы реляционной
БД до более высокой нормальной формы. Как правило достаточно 3-й формы. Зависит от
задачи. Более высокий уровень доступен только при соблюдении предыдущего.
Основная идея – исключить повторяющиеся (избыточные) данные так, чтобы каждая часть данных
была представлена только в одном месте. Позволяет упростить данные до максимально атомарного вида,
с которым проще работать.

1НФ -> Атомарные значения. Значения не явл. списками, множествами.
2НФ -> Все столбцы, не явл. частью ключа, зависят от этого ключа.
3НФ -> Значения, входящие в запись и не явл. частью ключа этой записи, не принадлежат
таблице.
НФБК -> расширенная 3НФ. Реляционная схема считается в нормальной форме Бойса-Кодда,
если для каждой из ее зависимостей A -> B выполняется одно из условий:
-> A -> B – тривиальная функциональная зависимость, то есть B – подмножество A.
-> A – первичный ключ для реляционной схемы.
То есть, если таблица находится в 3НФ и все ее столбцы явл. частью составного
первичного ключа, эта таблица находится в НФБК.
4НФ -> устранение многозначных зависимостей – когда столбец с первичным ключом имеет
связь один-ко-мгогим со столбцом, который не является ключом.
5НФ -> декомпозиция (разделение) таблицы на более малые для устранения избыточности
данных. Разбиение идет до тех пор, пока нельзя будет воссоздать оригинальную
таблицу путем объединения малых таблиц.
Денормализация – процесс ухода от правил нормализации там, где необходимо (например,
сваливание всего в одну таблицу для удобства аналитических запросов).

Схема – способ логического разделения таблиц в рамках одной БД (например, продажи через веб и приложение)
– звезда: имеет централизованное хранилище данных и ряд денормализованных таблиц измерений.
Данные сгруппированы. Проще конструкция. Проще писать сложные запросы. Больше памяти.
– снежинка: использует нормализованные данные. Минимум избыточности. Таблицы разветвляются.
Высокая сложность запросов.

Числовые типы данных:
integer – целые числе(4 байта)
bigint – целые числа (8 байт)
real – с плавающей точкой (4 байта)
double precision – с плавающей точкой (8 байт)
Символьные типы:
character(n) char(n) – строка из фиксированного количества символов
character varying(n) varchar(n) – строка из нефиксированного количества символов
text – текст произвольной длины
Логические типы:
boolean (true/false) – TRUE, 'true', 't', 'y', 'yes', 'on', '1'
Дата и время:
timestamp – дата и время (количество секунд с опр. даты)
time – время
timestamptz – дата и время и часовой пояс
timetz – время и часовой пояс

```
  date - дата
  interval - временной диапазон
 Специальные типы данных:
  json - данные json в текстовом виде. Стандарт для хранения данных в виде key-
value пар.
  jsonb - данные json в бинарном формате
  xml - данные в формате XML
  массивы
  NULL - значение отсутствует (информация неизвестна). С NULL нельзя сравнивать.
 */

-- Однострочный комментарий
/* ... */ -  многострочный комментарий

--Отличие ' ' от " "
" " - название сущностей (имена таблиц, полей, псевдонимов)
' ' - указание значений (строковые константы)

select pg_typeof() - определение типа
select pg_typeof(100::text) - явное указание типа (text)
select pg_typeof(cast(100 as text)) - явное указание типа (text)

as - использование псевдонима
any - хотя бы одно значение из подзапроса соотв. условию
all - все значения подзапроса соотв. условию

--Синтаксический порядок инструкции SELECT
select - что хотим вывести в результат
from - основная таблица
join - все остальные таблицы
on - условие соединения
where - фильтрация данных
aggregate - агрегация
group by - группировка
having - фильтрация сгрупированных данных
order by - сортировка (asc, desc) (по умолч. asc - по возрастанию)

--Логический порядок инструкции select
from
on
join
where
aggregate
group by
having
select
order by

--Операторы
distinct - уникальность
union  - вертикальное соединение снизу с проверкой на уникальность
union all - вертикальное соединение снизу без проверки на уникальность
except - из верхних значений вычитаются совпадающие с нижними значения
limit - ограничение на количество строк

--Типы join
inner - совпадающие значения
left - левая целиком, из правой совпадения, где нет совпадений будет null
right - правая целиком, из левой совпадения, где нет совпадений будет null
full - обе таблицы целиком, где нет совпадений будет null
cross - декартово произведение (Полная запись). Значение первого подмножества
перемножаем со значениями второго.

--Работа со строками
like - регистрозависимый поиск (Вернет true, если строка соотв. заданному
шаблону)
```

```
ilike – регистронезависимый поиск
% – от 0 до бесконечности
_ – ровно 1 символ

--Агрегация
aggregate – данные группируются по ключу, в качестве которого выступает один или несколько атрибутов
и внутри каждой группы вычисляются некоторые статистики
sum – суммирование всех значений (только для чисел)
count – счетчик записей (count distinct – счетчик уникальных записей)
avg – среднее значение (average)
min – минимальное значение
max – максимальное значение
string_agg – агрегация значений строк в одну строку
array_agg

--Группировка
group by – агрегирующий оператор. Можно группировать по нескольким полям.

--Фильтрация
where – фильтрует строки до агрегации
having – фильтрует строки после агрегации

--Сортировка
order by – сортирует конечный результат запроса
asc – по возрастанию (по умолчанию)
desc – по убыванию

--Подзапросы
Подзапрос – способ разделения логики формирования выборки (select, результаты которого исп. в др. select)
 Запрос внутри запроса
 Обычно используется в операторе where
  В соединениях – таблица
  В условиях in/exists/any/all – одномерный массив
  В условиях с операторами =/>/< – отдельные значения
select model, price
from printer
where price = (select max(price) from printer)

--Оператор case
case – условное выражение. Аналог if/else
--пример:
select amount,
      case
            when amount < 3 then 'низкий платеж'
            when amount between 3 and 9 then  'средний платеж'
            else 'высокий платеж'
      end case_pay
from payment

--работа с таблицами
create table – создание таблицы
alter table table_name action – изменение таблицы
drop table [if exists] table_name [cascade|restrict] – удаление таблицы
 restrict – не даст удалить таблицу пока есть ссылки на нее
 cascade – удаляет таблицу и все каскадные зависимости
insert into table(col_1, col_2...) values (val_1, val_2...) – вставить данные
update table – обновить данные
delete – удаление данных

Ограничения колонок:
 – not null
 – unique – каждое значение, кроме null, должно быть уникально
 – primary key – not null + unique (на одну таблцу один primary key)
```

Ограничения таблиц:
 - **unique** (colomn_list) - ограничение на группу колонок
 - **primary key** (column_list) - первичный ключ по группе колонок

serial - целочисленная колонка к которой привязан счетчик (суррогатный первичный ключ)

--Оконные (аналитические функции)
Оконная функция никак не меняет количество строк в выдаче, но как правило обогощает информацией
group by - возвращает по группе строк одну результирующую строку
оконная - для каждой строки вычисляет что-то в рамках заданного окна
select **row_number**() **over**(**partition by** customer_id **order by** rental_date **desc**) **as** rental_rank
- группируем относительно customer_id, внутри каждой группы (окна) сортируем по убыванию, добавляем колонку
 rental_rank с порядковым номером аренды (за это отвечает **row_number**())

row_number() - ф-я, кот. применяется к окну
over - описание окна
partition by - поле или список полей, которые описывают группу строк для применения оконной ф-и
order by - поле, кот. задает порядок записей внутри окна

CTE - временный результат запроса, кот. можно использовать с другими запросами
 Временный - существует только в рамках запроса
--пример:
with название (список колонок) **as** (
CTE - запрос
)
основной запрос

Рекурсивный запрос - вычисление чего-то итерациями до того, как будет выполнено некоторое условие

Представления (**view**) - именованные запросы, которые помогают сделать представление (именно вид)
данных, лежащий в таблицах. Удобно для часто исп. запросов.
create view название **as** query
Отличие от **view** от CTE:
- Представления могут быть проиндексированы
- Представления - физические сущности в БД
- ОТВ (обобщенные табличные выражения) работают с рекурсией
- ОТВ - временные

Материализованное представление хранит запрос и его результат. Нужно переодически обновлять.
При обращении запрос не выполняется.

explain - демонстрирует этапы выполнения запроса и может быть использован для оптимизации.

Индексы - специальные сущности, кот. добавляются в БД и позволяют выполнять поиск быстрее.
минус - при добавлении новых строк нужно обновлять индексы.
create index on film(film_id) - создание индекса

Массив - коллекция элементов одного типа.

Партиционирование - это метод разделения больших (исходя из количества записей, а не столбцов)
таблиц на много маленьких.
 Партиции - наследованные таблицы.
минус - не может быть внешних ключей, указывающих на партиционированную таблицу
 Партиционирование в PostgreSQL используется для разделения таблицы на несколько

логических частей (партиций), что позволяет улучшить производительность запросов и упростить
администрирование больших таблиц, улучшает масштабируемость системы.

```
select pg_typeof(100) --определение типа (integer)

select pg_typeof(100.0) --определение типа (numeric)

select pg_typeof('100.0') --определение типа (unknown)

select pg_typeof(100::text) --явное указание типа (text)

select pg_typeof(cast(100 as text)) --явное указание типа (text)

--Синтаксический порядок инструкции SELECT
select - что хотим вывести в результат
from - основная таблица
join - все остальные таблицы
on - условие соединения
where - фильтрация данных
aggregate - агрегация
group by - группировка
having - фильтрация сгрупированных данных
order by - сортировка (asc, desc) (по умолч. asc - по возрастанию)

--Логический порядок инструкции select
from
on
join
where
aggregate
group by
having
select
order by

1. Получить информация по фильмам

--все столбцы
select *
from film

--только указанные столбцы
select title, description, release_year
from film

--использование псевдонима (alias)
select title as название, description as описание, release_year as год_релиза
from film

--использование псевдонима (alias) в "..." если нужны пробелы или регистр
select title as "Название Фильма", description as описание, release_year as
год_релиза
from film

2. Получить информацию по стоимости аренды фильма в день

select title, rental_rate, rental_duration, rental_rate / rental_duration as
cost_per_day
from film

--Основные типы данных
integer
```

```
numeric(6, 2) --9999.99 (6 символов всего, из которых 2 после .). Работает посимвольно. Для финансов
float --округляется при .5 до ближайшего четного числа

--Арифметические операции
select 2 + 2

select 2 - 2

select 2 * 2

select 2 / 2

select power(2, 3) --возведение в степень

select mod(4, 3) --остаток от деления

select sin(0.5) --синус в радианах

select sind(0.5) --синус в градусах
```

2.1. Округлить стоимость аренды до сотых

```
select title, rental_rate, rental_duration, round(rental_rate / rental_duration, 2) as cost_per_day
from film

round(numeric, integer) --округление до указанного знака после запятой
round(float) --округление до целого числа
```

3. Отличие **int** от **float** от **numeric**, почему при работе с финансами нужно строго использовать тип **numeric**

```
select x,
      round(x::numeric) as num_round,
      round(x::float) as fl_round
from generate_series(0.5, 5.5, 1) x
```

4. Отсортировать стоимость аренды в день от большего к меньшему

```
--по умолчанию asc - по возрастанию
select title, rental_rate, rental_duration, round(rental_rate / rental_duration, 2) as cost_per_day
from film
order by round(rental_rate / rental_duration, 2)

--по убыванию - desc
select title, rental_rate, rental_duration, round(rental_rate / rental_duration, 2) as cost_per_day
from film
order by round(rental_rate / rental_duration, 2) desc --по убыванию

--через псевдоним
select title, rental_rate, rental_duration, round(rental_rate / rental_duration, 2) as cost_per_day
from film
order by cost_per_day desc

--через номерное указание столбца
select title, rental_rate, rental_duration, round(rental_rate / rental_duration, 2) as cost_per_day
from film
order by 4 desc

--двойная сортировка. Сначала по стоимости, а внутри группы по названию
```

```sql
select title, rental_rate, rental_duration, round(rental_rate / rental_duration,
2) as cost_per_day
from film
order by cost_per_day desc, title
```

5. Топ-10 самых дорогих по стоимости аренды за день, начиная с 58 позиции

```sql
--первые 10 строк
select title, rental_rate, rental_duration, round(rental_rate / rental_duration,
2) as cost_per_day
from film
order by cost_per_day desc
limit 10

--начиная с 13 версии postrgesql. первые 10 строк и остаток
select title, rental_rate, rental_duration, round(rental_rate / rental_duration,
2) as cost_per_day
from film
order by cost_per_day desc
fetch first 1 rows with ties --почему то не работает

--первые 10 начиная с 58
select title, rental_rate, rental_duration, round(rental_rate / rental_duration,
2) as cost_per_day
from film
order by cost_per_day desc
offset 57
limit 10
```

6. Вывести уникальный список индентификаторов пользователей из таблицы по платежам

```sql
select customer_id --16049 строк
from payment

select distinct customer_id --599 строк
from payment
```

7. Вывести данные о последнем платеже каждого пользователя

```sql
select distinct on (customer_id) *
from payment
order by customer_id, payment_date desc
```

8. Вывести ФИО пользователей в одном столбце

```sql
varchar(N) - от нуля до N
char(N) - ровно N (недостающие заполнятся пробелами)
text

--Конкатинация - объединение подстрок в одну строку
select last_name || ' ' || first_name || ' ' || middle_name
from person

select concat(last_name, ' ', first_name, ' ', middle_name)
from person

select concat_ws(' ', last_name, first_name, middle_name)
from person

select 2 + null

select 'hello' || null

--функция проигнорирует null
select concat('hello', null) --функция проигнорирует null
```

9. Вывести города, начинающиеся на Z

```
like – регистрозависимый поиск
ilike – регистронезависимый поиск
% – от 0 до бесконечности
_ – ровно 1 символ

select *
from city
where city like 'Z%'

select *
from city
where left(city, 1) = 'Z'
```

9.1. Название с пробелом

```
select *
from city
where city like '% %'

select *
from city
where strpos(city, ' ') != 0
```

9.2. Города где нет пробела и 4 символа

```
select *
from city
where city not like '% %' and city like '____'

select *
from city
where city not like '% %' and char_length(city) = 4
```

10. Вывести ФИО в нижнем регистре

```
select lower(concat_ws(' ', last_name, first_name, middle_name))
from person
```

11. Использование оператора overlay для замены слова в строке

```
select concat_ws(' ', last_name, first_name, middle_name),
      overlay(concat_ws(' ', last_name, first_name, middle_name)
            placing 'Nikolay'
            from strpos(concat_ws(' ', last_name, first_name, middle_name),
'Николай')
            for char_length('Николай'))
from person
where first_name = 'Николай'
```

12. Вывести ФИО в красивом регистре. Каждое слово с большой буквы

```
select initcap(lower(concat_ws(' ', last_name, first_name, middle_name)))
from person
```

13. Получить id пользователей, арендовавших фильмы в срок с 27-05-2005 по 28-05-2005

```
timestamp – дата и время (количество секунд с опр. даты)
time – время
timestamptz – дата и время и часовой пояс
timetz – время и часовой пояс
date – дата
interval – временной диапазон
```

```
show lc_time --en_US.UTF-8

yyyy.mm.dd
mm.dd.yyyy

select  '2023.12.15'::date

select now()

2023-12-14 18:15:45.453 +0300 - универсальный формат

select *
from payment
where payment_date::date between '2005-05-27' and '2005-05-28'

14. Демонстрация отличия date_part от date_trunc
Получить день и месяц платежа

select date_part('year', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('month', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('day', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('hour', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('minutes', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('seconds', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('week', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('isodow', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_part('epoch', '2023-12-14 18:15:45.453 +0300'::timestamptz)

select date_trunc('year', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_trunc('month', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_trunc('day', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_trunc('hour', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_trunc('minutes', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_trunc('seconds', '2023-12-14 18:15:45.453 +0300'::timestamptz),
       date_trunc('week', '2023-12-14 18:15:45.453 +0300'::timestamptz)

select *
from payment
where date_trunc('month', payment_date) = '2005-08-01'

select *
from payment
where date_part('month', payment_date) = 8 and date_part('year', payment_date) =
2005

14. Получить количество дней, месяцев, лет с '2020-02-29' по сегодняшний день

select now() --текущая дата и время (универсально для всех СУБД)

select current_timestamp

select current_time

select current_date

select current_date - '2020-02-29'::date --дни

select age(current_date, '2020-02-29'::date)

select date_part('year', age(current_date, '2020-02-29'::date)) --годы

select date_part('year', age(current_date, '2020-02-29'::date)) * 12 +
       date_part('month', age(current_date, '2020-02-29'::date))
```

15. Вывести данные по платежам пользователей со значениями идентификаторов менее 5 и размерами платажей 2.99 и 4.99

```
and -> *
or -> +
not -> не
```

```
select customer_id, amount
from payment
where (customer_id = 1 or customer_id = 2) and (amount = 2.99 or amount = 4.99)
```

```
boolean
true 1 't' 'yes' 'on'
false 0 'f' 'no' 'off'
```

```
select *
from customer
where activebool is false
```

```
select *
from customer
where activebool is true
```

```
--всегда пустая таблица
select *
from customer
where activebool = null
```

```
--для сравнение с null используется is
select *
from customer
where activebool is null
```

```
--Список названий всех фильмов и их языков
select title, l."name"
from film f
join "language" l on f.language_id  = l.language_id
```

```
--Список актеров, снимавшихся в фильме с id = 508
select a.actor_id, a.first_name || ' ' || a.last_name as actor
from film f
join film_actor fa on f.film_id = fa.film_id
join actor a on fa.actor_id = a.actor_id
where f.film_id = 508
```

```
--Количество актеров в фильме с id = 384
select count(fa.actor_id)
from film f
join film_actor fa on f.film_id = fa.film_id
where f.film_id = 384
```

```
--Список фильмов, в которых снималось больше 10 актеров
select f.title, count(fa.actor_id)
from film f
join film_actor fa on fa.film_id = f.film_id
group by f.film_id
having count(fa.actor_id) > 10
```

=== Часть 1. Сложные запросы. Соединение таблиц. ===

При соединении данных как минимум с одной стороны значения должны быть уникальны.

```
create table table_one (
      name_one varchar(255) not null
);
```

```sql
create table table_two (
      name_two varchar(255) not null
);

insert into table_one (name_one)
values ('one'), ('two'), ('three'), ('four'), ('five')

insert into table_two (name_two)
values ('four'), ('five'), ('six'), ('seven'), ('eight')

select * from table_one

select * from table_two

--Типы join
inner / left / right / full / cross

select table_one.name_one, table_two.name_two
from table_one
inner join table_two on table_one.name_one = table_two.name_two

--Тоже самое (inner можно опустить)
select table_one.name_one, table_two.name_two
from table_one
join table_two on table_one.name_one = table_two.name_two

--Можно использовать псевдонимы (alias) для компактности кода
select t1.name_one, t2.name_two
from table_one t1
join table_two  t2 on t1.name_one = t2.name_two

--left: Левая целиком, из правой совпадения, где нет совпадений будет null
select t1.name_one, t2.name_two
from table_one t1
left join table_two  t2 on t1.name_one = t2.name_two

--right: Правая целиком, из левой совпадения, где нет совпадений будет null
select t1.name_one, t2.name_two
from table_one t1
right join table_two  t2 on t1.name_one = t2.name_two

--full: Обе таблицы целиком, где нет совпадений будет null
select t1.name_one, t2.name_two
from table_one t1
right join table_two  t2 on t1.name_one = t2.name_two

--Уникальные значения для двух таблиц
select t1.name_one, t2.name_two
from table_one t1
full join table_two  t2 on t1.name_one = t2.name_two
where t1.name_one is null or t2.name_two is null

--Cross: Декартово произведение (Полная запись). Значение первого подмножества
перемножаем со значениями второго.
select t1.name_one, t2.name_two
from table_one t1
cross join table_two t2

--Краткая запись
select t1.name_one, t2.name_two
from table_one t1, table_two t2

--Чтобы обращаться к одной таблице два раза, нужно исп. псевдонимы. (Все
варианты)
select t1.name_one, t2.name_one
```

```sql
from table_one t1, table_one t2
where t1.name_one > t2.name_one --Исключим зеркальные пары и пары типа A - A

--Удаление данных
delete from table_one

delete from table_two

--Count: количество записей
select count(*)
from table_one t1

--Пример верной записи. Второй join присоединяет таблицу к результату первого
join, потоум важно следить за уникальностью значений.
select count(*)
from customer c
join payment p on c.customer_id = p.customer_id
join rental r on r.rental_id = p.rental_id

1. Выведите уникальный список фильмов, которые брали в аренду '24-05-2005'

select distinct f.title
from film f
join inventory i on f.film_id = i.film_id
join rental r on r.inventory_id = i.inventory_id and r.rental_date::date =
'2005-05-24'

select distinct f.title
from film f
join inventory i on f.film_id = i.film_id
join rental r on r.inventory_id = i.inventory_id
where r.rental_date::date = '2005-05-24'

2. Вывести фильмы, которые отсутствуют на дисках

select f.title, i.inventory_id
from film f
left join inventory i on f.film_id = i.film_id
where i.inventory_id is null

select *
from film
where film_id not in (select film_id from inventory i)

=== Часть 2. Сложные запросы. Условия в операторе case ===

3. Используя оператор case проранжировать размеры платежей пользователей:
до 3 - низкий платеж
от 3 до 9 - средний платеж
более 9 - высокий платеж

select case_pay, count(*)
from (
      select amount,
            case
                  when amount < 3 then 'низкий платеж'
                  when amount between 3 and 9 then  'средний платеж'
                  else 'высокий платеж'
            end case_pay
      from payment) t
group by case_pay

=== Часть 3. Сложные запросы. Агрегатные фунции и группировка данных ===

sum
count
```

```
avg
min
max
string_agg
array_agg
```

4. Посчитать количество актеров в каждом фильме

```
select f.rental_duration, f.rental_rate, count(actor_id), string_agg(distinct
f.title, ', ')
from film_actor fa
join film f on fa.film_id = f.film_id
group by f.rental_duration, f.rental_rate

select f.title, count(actor_id)
from film_actor fa
join film f on fa.film_id = f.film_id
group by f.film_id
```

5. Найти значения среднего, минимального и максимального платежей

```
select count(*), sum(amount), avg(amount), min(amount), max(amount)
from payment p
```

6. Найти сумму платежей на каждый месяц

```
select date_trunc('month', payment_date), sum(amount)
from payment p
where date_trunc('month', payment_date) < '2005.08.01'
group by date_trunc('month', payment_date)
having sum(amount) > 10000
```

7. Вывести результаты агрегаций по каждому пользователю, каждому сотрудику на каждый месяц

```
select customer_id, staff_id, date_trunc('month', payment_date), sum(amount)
from payment p
where date_trunc('month', payment_date) < '2005.08.01'
group by date_trunc('month', payment_date)

select customer_id, staff_id, date_trunc('month', payment_date), sum(amount)
from payment p
where customer_id < 3
group by 1, 2, 3

select customer_id, staff_id, date_trunc('month', payment_date), sum(amount)
from payment p
where customer_id < 3
group by grouping sets (1, 2, 3)

select customer_id, staff_id, date_trunc('month', payment_date), sum(amount)
from payment p
where customer_id < 3
group by grouping sets (1, 2, 3), grouping sets (1)

--все возможные результаты агрегатов
--create temporary table temp_pay as - временная таблица
select customer_id, staff_id, date_trunc('month', payment_date), sum(amount)
from payment p
group by cube (1, 2, 3)

select customer_id, staff_id, date_trunc('month', payment_date), sum(amount)
from payment p
where customer_id < 3
group by rollup (1, 2, 3)
```

8. Найти сумму платажей пользователей меньше 5 и больше 5

```
select customer_id,
      sum(case when amount < 5 then amount end),
      sum(case when amount >= 5 then amount end)
from payment p
group by customer_id

select customer_id,
      sum(amount) filter (where amount < 5),
      sum(amount) filter (where amount >= 5)
from payment p
group by customer_id
```

=== Часть 4. Сложные запросы. Подзапросы. ===

9. Вывести количество фильмов, со стоимостью аренды за день больше, чем среднее по всем фильмам
 скаляр - **select**, условия, не должно быть алиаса
 таблицу - **from**, **join**, обязательно адрес
 одномерный массив - **in**, не должно быть алиаса

```
select count(*)
from film
where rental_rate / rental_duration > (select avg(rental_rate / rental_duration)
from film)
```

Какое количество всех возможных уникальных пар имен можно получить используя данные по пользователям, если убрать варианты где имя А равно имени А.

```
select first_name --599
from customer c

select distinct first_name --591
from customer c

select 591 * 591 - 591 --348690

select count(distinct c1.first_name || ' - ' || c2.first_name) --348690
from (
      select distinct first_name
      from customer) c1
cross join (
      select distinct first_name
      from customer) c2
where c1.first_name != c2.first_name

select
from customer c, customer c2
where c1.first_name != c2.first_name
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 5.

===== Часть 1. Оконные функции. =====

Вспоминаем логический порядок инструкции **select**

```
from
on
join
where
aggregate
group by
having
over
select
```

```
функция (аргументы) over (partition by arg1, arg2 order by arg1, arg2)

over () - по всем данным глоабльно в хаотичном порядке
over (partition by ... ) - в рамках групп  в хаотичном порядке
over (order by ... ) - по всем данным глобально  в упорядоченном виде
over (partition by ... order by ... ) - в рамках групп  в упорядоченном виде

1. Вывести ФИО пользователя и название третьего фильма, который он брал в
аренду.

select f.title, c.first_name || ' ' || c.last_name
from (
      select *, row_number() over (partition by customer_id order by
rental_date)
      from rental) r
join inventory i on i.inventory_id = r.inventory_id
join film f on f.film_id = i.film_id
join customer c on c.customer_id = r.customer_id
where row_number = 3

2. Выведите таблицу, содержащую имена покупателей, арендованные ими фильмы и
средний платеж
каждого покупателя

select f.title, c.first_name || ' ' || c.last_name,
      avg(p.amount) over (partition by c.customer_id),
      sum(p.amount) over (partition by c.customer_id),
      count(p.amount) over (partition by c.customer_id),
      avg(p.amount) over (partition by p.staff_id),
      sum(p.amount) over (partition by p.staff_id),
      count(p.amount) over (partition by p.staff_id),
      avg(p.amount) over (partition by c.customer_id, p.staff_id),
      sum(p.amount) over (partition by c.customer_id, p.staff_id),
      count(p.amount) over (partition by c.customer_id, p.staff_id),
      avg(p.amount) over (),
      sum(p.amount) over (),
      count(p.amount) over ()
from rental r
join inventory i on i.inventory_id = r.inventory_id
join film f on f.film_id = i.film_id
join customer c on c.customer_id = r.customer_id
join payment p on p.rental_id = r.rental_id

select customer_id, sum(amount) * 100. / sum(sum(amount)) over ()
from payment
group by customer_id

3. Вывести накопительным итогом дневные суммы платежей по сотрудникам на каждый
месяц

select staff_id, payment_date::date, sum(amount),
      sum(sum(amount)) over (partition by staff_id order by payment_date::date),
      avg(sum(amount)) over (partition by staff_id order by payment_date::date)
from payment
group by staff_id, payment_date::date

sum(96.80) = 96.80
sum(96.80 + 1246.91) =
sum(96.80 + 1246.91 + 1212.02) =

1     2005-08-22  1286.80
1     2005-07-26
1     2005-08-20  1263.00
1     2005-07-28
1     2005-08-19
```

```
2     2005-06-19  740.32
2     2005-06-19  740.32
2     2005-08-17  1254.07
2     2005-07-11  983.68

select customer_id, payment_date::date, amount,
      sum(amount) over (partition by customer_id order by payment_date::date),
      avg(amount) over (partition by customer_id order by payment_date::date)
from payment

4. Используя функции lag/lead показать изменения по прибыли за месяц

тада_убытия1 | дата_прибытия1
тада_убытия2 | дата_прибытия2

тада_убытия2 - lag(дата_прибытия1)

select date_trunc('month', payment_date), sum(p.amount),
      lag(sum(p.amount)) over (order by date_trunc('month', payment_date)),
      sum(p.amount) - lag(sum(p.amount)) over (order by date_trunc('month',
payment_date))
from payment p
group by date_trunc('month', payment_date)

select customer_id, payment_date,
      lag(amount) over (partition by customer_id order by payment_date),
      amount,
      lead(amount) over (partition by customer_id order by payment_date)
from payment p

select customer_id, payment_date,
      lag(amount, 5) over (partition by customer_id order by payment_date),
      amount,
      lead(amount, 5) over (partition by customer_id order by payment_date)
from payment p

select customer_id, payment_date,
      lag(amount, 5, 0.) over (partition by customer_id order by payment_date),
      amount,
      lead(amount, 5, 0.) over (partition by customer_id order by payment_date)
from payment p

5. ранжирование

row_number - сквозная нумерация
dense_rank - одинаковые ранги по общим знаментелям предыдущий ранг + 1
rank - одинаковые ранги по общим знаментелям количество значений в предыдущем
ранге + 1

select customer_id, payment_date::date,
      row_number() over (order by payment_date::date),
      dense_rank() over (order by payment_date::date),
      rank() over (order by payment_date::date)
from payment p

===== Часть 2. Общие табличные выражения. Рекурсия. =====

5. В CTE получить список сотрудников, нанятых в 2020 году, к этим данным
добавить информацию по ФИО сотрудника

with cte1 as (
      select *
      from employee
      where date_part('year', hire_date) = 2020),
cte2 as (
```

```
      select *
      from person
      where lower(left(last_name, 1)) = 'a'),
cte3 as (
      select concat_ws(' ', cte2.last_name, cte2.first_name, cte2.middle_name)
      from cte1
      join cte2 on cte1.person_id = cte2.person_id)
select *
from cte3
```

6. Демонстрация рекурсии на факториале

```
with recursive r as (
      --стартовая часть
      select 1 as x, 1 as factorial
      union
      --рекурсивная часть
      select x + 1 as x, factorial * (x + 1) as factorial
      from r
      where x < 10)
select *
from r
```

7. Используя рекрсию показать работу с иерархическим списком департаментов.

```
уровень 1
уровень 2
уровень 3
...
уровень N
```

```
with recursive r as (
      --стартовая часть
      select *, 1 as level
      from structure
      where unit_id = 114
      union
      --рекурсивная часть
      select s.*, level + 1 as level
      from r
      join structure s on r.unit_id = s.parent_id)
select count(*)
from r
join position p on p.unit_id = r.unit_id
join employee e on e.pos_id = p.pos_id
```

```
114   12    Отдел Центр разработки Medio  1
```

```
with recursive r as (
      --стартовая часть
      select *, 1 as level
      from structure
      where unit_id = 114
      union
      --рекурсивная часть
      select s.*, level + 1 as level
      from r
      join structure s on r.parent_id = s.unit_id)
select *
from r
```

8. Генерация календарей с использованием рекурсии

```
with recursive r as (
      --стартовая часть
      select 1 as x
```

```
	union
	--рекурсивная часть
	select x + 3 as x
	from r
	where x < 10)
select *
from r

select x
from generate_series(1, 10, 3) x

with recursive r as (
	--стартовая часть
	select '01.01.2024'::date as x
	union
	--рекурсивная часть
	select x + 1 as x
	from r
	where x < '31.12.2024'::date)
select *
from r

select x::date
from generate_series('01.01.2024'::date, '31.12.2024'::date, interval '1 day') x
```

9. Используя функции lag/lead показать корректные изменения по прибыли за месяц

```
select date_trunc('month', payment_date), sum(p.amount),
	lag(sum(p.amount)) over (order by date_trunc('month', payment_date)),
	sum(p.amount) - lag(sum(p.amount)) over (order by date_trunc('month',
payment_date))
from payment p
group by date_trunc('month', payment_date)

explain analyze --5177.40 / 12
with recursive r as (
	--стартовая часть
	select min(date_trunc('month', payment_date)) x from payment
	union
	--рекурсивная часть
	select x + interval '1 month' as x
	from r
	where x < (select max(date_trunc('month', payment_date)) from payment))
select x::date, coalesce(p.sum, 0),
	lag(coalesce(p.sum, 0), 1, 0.) over (order by x),
	coalesce(p.sum, 0) - lag(coalesce(p.sum, 0), 1, 0.) over (order by x)
from r
left join (
	select date_trunc('month', payment_date), sum(amount)
	from payment
	group by date_trunc('month', payment_date)) p on x = date_trunc
order by x

select coalesce(null, null, null, null, 45, null, null, null, null, 98)

explain analyze --16366.55 / 12
select x::date, coalesce(p.sum, 0),
	lag(coalesce(p.sum, 0), 1, 0.) over (order by x),
	coalesce(p.sum, 0) - lag(coalesce(p.sum, 0), 1, 0.) over (order by x)
from generate_series(
	(select min(date_trunc('month', payment_date)) x from payment),
	(select max(date_trunc('month', payment_date)) x from payment),
	interval '1 month') x
left join (
	select date_trunc('month', payment_date), sum(amount)
	from payment
```

```
      group by date_trunc('month', payment_date)) p on x = date_trunc
order by x
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 6.

===== Часть 1. Представления. Материализованные представления. =====

1. Нужно создать представление, которое будет выводить информацию по текущему окладу сотрудников.

```
create view salary_actual as
      explain analyze --547.91 / 3.2
      select *
      from (
            select *, row_number() over (partition by emp_id order by
effective_from desc)
            from employee_salary)
      where row_number = 1

explain analyze --547.91 / 3.2
select *
from salary_actual
```

2. Сформировать материализованное представление, которое будет хранить информацию по ФИО сотрудника, должности и департаменту, текущему окладу.

```
create materialized view some_task as

explain analyze --640.88 / 14
      select concat_ws(' ', p.last_name, p.first_name, p.middle_name),
p2.pos_title, s.unit_title, sa.salary
      from employee e
      join person p on p.person_id = e.person_id
      join "position"p2 on p2.pos_id = e.pos_id
      join "structure" s on s.unit_id = p2.unit_id
      join salary_actual sa on sa.emp_id = e.emp_id
      where lower(left(concat_ws(' ', p.last_name, p.first_name, p.middle_name),
1)) in ('а', 'б', 'в') --627.25 / 12

with no data

select *
from some_task

refresh materialized view some_task

explain analyze --84.32 / 0.15
select *
from some_task
where lower(left(concat_ws, 1)) in ('а', 'б', 'в') --107.35 / 1.5

select 14 / 0.15

create index first_letter on some_task (lower(left(concat_ws, 1)))

explain analyze --84.32 / 0.15
select *
from some_task
where lower(left(concat_ws, 1)) in ('а', 'б', 'в') --69.15 / 0.12
```

===== Часть 2. Индексы =====

```
select film_id
from film
```

```
where film_id = 127

select *
from film

alter table film drop constraint film_pkey cascade

0 индексов - 472 кб

btree = < > between null
hash =
gin
gist

explain analyze --Seq Scan on film
select film_id
from film
where film_id = 127

create index title_idx on film (title)

1 индекс - 528 кб

explain analyze
select title, film_id, * --Index Scan
from film
where title = 'AGENT TRUMAN'

explain analyze
select title, film_id, * --Index Scan
from film
order by title

alter table film add constraint film_pkey primary key (film_id)

explain analyze --Seq Scan on film  (cost=0.00..67.50 rows=1 width=4) (actual
time=0.027..0.132 rows=1 loops=1)
select film_id
from film
where film_id = 127

explain analyze --Index Only Scan using film_pkey on film  (cost=0.28..8.29
rows=1 width=4) (actual time=0.023..0.024 rows=1 loops=1)
select film_id
from film
where film_id = 127

1-1000
1-500 501-1000
1-250 251-500 501-750 751-1000
1-125 126-250

create index film_id_hash_idx on film using hash(film_id)

explain analyze --Index Scan using film_id_hash_idx on film  (cost=0.00..8.02
rows=1 width=4) (actual time=0.011..0.012 rows=1 loops=1)
select film_id
from film
where film_id = 127

explain analyze --Index Only Scan using film_pkey on film  (cost=0.28..16.72
rows=126 width=4) (actual time=0.073..0.108 rows=126 loops=1)
select film_id
from film
where film_id < 650
```

```
create index strange_1_idx on film (title, rental_duration, rental_rate, length)

explain analyze
select film_id
from film
where rental_duration = 8 and rental_rate = 2.99

create index strange_2_idx on film (title, rental_duration, rental_rate, length,
description)

5 индексов - 848 кб

insert x 6

explain analyze --Seq Scan on payment p  (cost=0.00..359.74 rows=80 width=26)
(actual time=3.238..3.238 rows=0 loops=1)
select *
from payment p
where payment_date::date = '10/08/2005'

create index payment_date_idx on payment (payment_date)

explain analyze --Seq Scan on payment p  (cost=0.00..359.74 rows=80 width=26)
(actual time=3.238..3.238 rows=0 loops=1)
select *
from payment p
where payment_date::date = '10/08/2005'

create index payment_date_date_idx on payment (cast(payment_date as date))

explain analyze --Seq Scan on payment p  (cost=0.00..359.74 rows=80 width=26)
(actual time=3.238..3.238 rows=0 loops=1)
select *
from payment p
where payment_date::date = '10/08/2005'

drop index payment_date_date_idx

===== Часть 3. План запроса =====

Ссылка на сервис по анализу плана запроса
https://tatiyants.com/pev/
https://habr.com/ru/post/203320/

explain analyze
select x::date, coalesce(p.sum, 0),
       lag(coalesce(p.sum, 0), 1, 0.) over (order by x),
       coalesce(p.sum, 0) - lag(coalesce(p.sum, 0), 1, 0.) over (order by x)
from generate_series(
       (select min(date_trunc('month', payment_date)) x from payment),
       (select max(date_trunc('month', payment_date)) x from payment),
       interval '1 month') x
left join (
       select date_trunc('month', payment_date), sum(amount)
       from payment
       group by date_trunc('month', payment_date)) p on x = date_trunc
order by x

explain (format json, analyze)
select x::date, coalesce(p.sum, 0),
       lag(coalesce(p.sum, 0), 1, 0.) over (order by x),
       coalesce(p.sum, 0) - lag(coalesce(p.sum, 0), 1, 0.) over (order by x)
from generate_series(
       (select min(date_trunc('month', payment_date)) x from payment),
       (select max(date_trunc('month', payment_date)) x from payment),
       interval '1 month') x
```

```
left join (
      select date_trunc('month', payment_date), sum(amount)
      from payment
      group by date_trunc('month', payment_date)) p on x = date_trunc
order by x


===== Часть 4. Сложные типы данных. json =====

create table orders (
     id serial primary key,
     info json not null
);

insert into orders (info)
values
 (
'{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}'
 ),
 (
'{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}'
 ),
 (
'{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}'
 ),
 (
'{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}'
 );

select split_part(info::text, 'items', 2) --грубая ошибка
from orders

select info, pg_typeof(info)
from orders

11. Выведите общее количество заказов

select info->'items', pg_typeof(info->'items')
from orders

select info->'items'->'qty', pg_typeof(info->'items'->'qty')
from orders

select info->'items'->>'qty', pg_typeof(info->'items'->>'qty')
from orders

select sum((info->'items'->>'qty')::numeric)--, pg_typeof((info->'items'-
>>'qty')::numeric)
from orders

12.  Выведите среднее количество заказов, продуктов начинающихся на "Toy"

select avg((info->'items'->>'qty')::numeric)
from orders
where info->'items'->>'product' like 'Toy%'

select json_object_keys(info->'items')
from orders

json - не очень хорошо, так как это строка
jsonb - бинарный json

===== Часть 5. Сложные типы данных. array =====

time[] ['10:00', '16:00']
int[] [5674,1234,697]
```

```
text[] ['3457647', '01.01.2023', 'fghdfgh']
```

13. Используя функцию array_length найти сколько сотрудников задействовано на проектах

```
select "name", employees_id, assigned_id
from projects

select array_length('{{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},
{1,2,3}}'::text[], 1) --8

select array_length('{{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},
{1,2,3}}'::text[], 2) --3

select cardinality('{{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},{1,2,3},
{1,2,3}}'::text[]) --24

select "name", employees_id, assigned_id, cardinality(employees_id) + 1
from projects

select "name", employees_id, assigned_id,
      case
            when employees_id @> array[assigned_id] then
cardinality(employees_id)
            else cardinality(employees_id) + 1
      end
from projects

select project_id, name, count(distinct unnest)
from (
      select *, unnest(array_append(employees_id, assigned_id))
      from projects)
group by project_id, name
```

14. Вывести все проекты, в которых задействован сотрудник с идентификатором 2253 демонстрация операторов @>, <@, any, all, array_position

```
select *
from projects
where employees_id && array[2253, 567567,456785]

select *
from projects
where employees_id @> array[2253]

select *
from projects
where employees_id <@ array[2253]

select *
from projects
where 2253 = any(employees_id) --some

select *
from projects
where 2253 = all(employees_id)

select *, array_position(employees_id, 2253)
from projects
where array_position(employees_id, 2253) is not null

select *, array_positions(employees_id || array[2253], 2253)
from projects
where array_position(employees_id, 2253) is not null

create table aa (
```

```
      id int,
      val int[])

insert into aa
values (1, array[5, 10])

select val[1]
from aa

update aa
set val[-10] = 100

select val[-10]
from aa

{[-10:2]={100,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,5,10}}
```