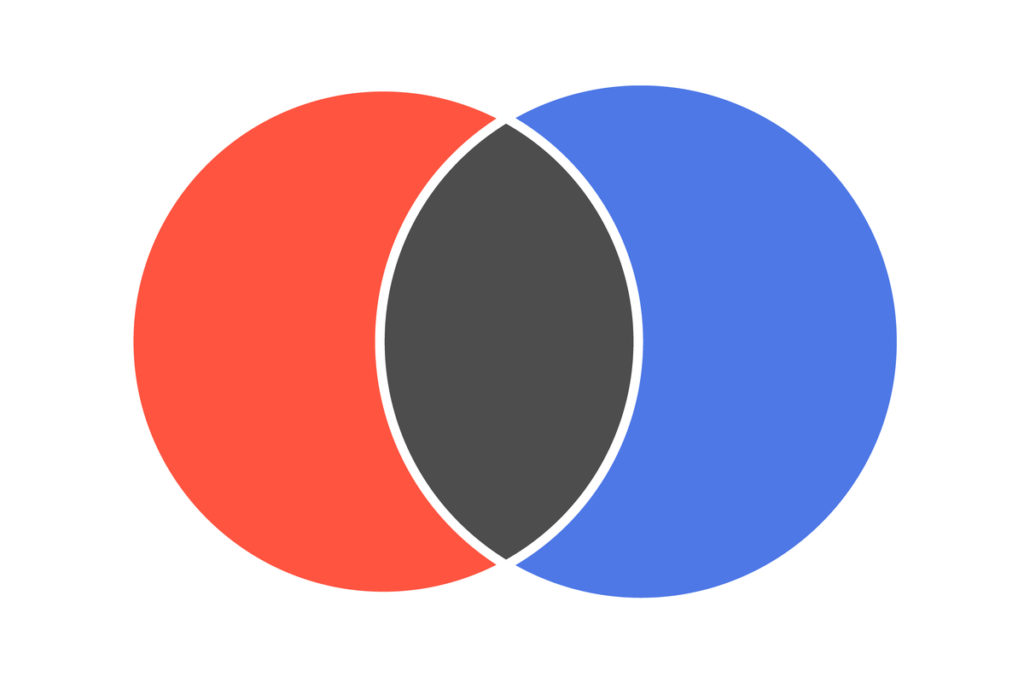
1. Что такое PK и в чем отличие от FK?

Понятия ключей были придуманы для создания отношения между таблицами, PK – primary key, основной ключ. PK обычно указывается для уникальных значений в таблице, поэтому чаще всего PK используется для идентификации. FK – foreign key, внешний ключ, который используется для связывания данных в двух таблицах и контроля данных хранящихся в таблице. FK привязывается к PK таблице.

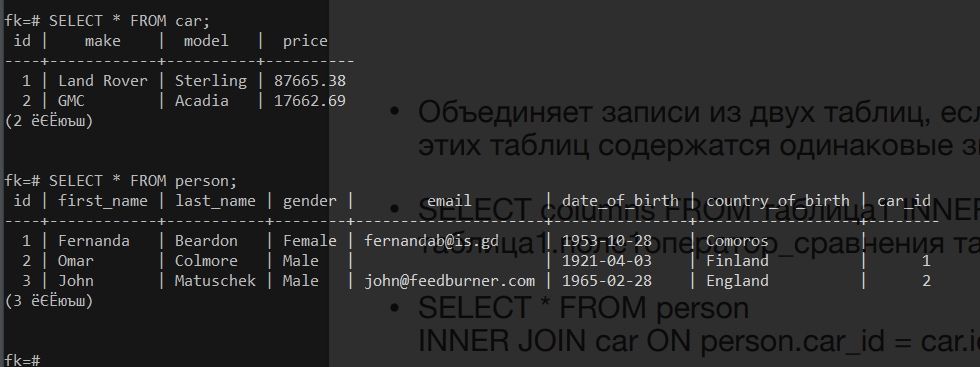
1. Напишите все что знаете про объединение данных.

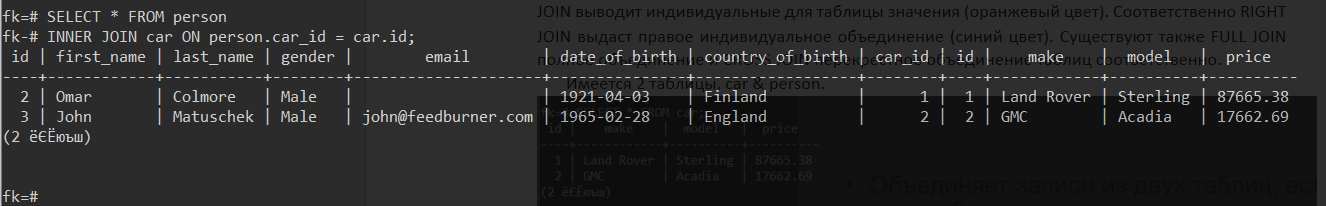
В sql имеется возможность объединения двух и более таблиц в одну, с использованием команды JOIN. Объединение происходит по формату кругов Эйлера:



Существует внутренние объединение таблиц (серый цвет), который объединяет лишь схожие данные, для этого используется команда INNER JOIN. Левое внешнее объединение LEFT JOIN выводит индивидуальные для таблицы значения (оранжевый цвет). Соответственно RIGHT JOIN выдаст правое индивидуальное объединение (синий цвет). Существуют также FULL JOIN полное объединение и CROSS JOIN перекрестное объединение таблиц соответственно.

Имеется 2 таблицы, car & person.



Произведем вывод объединения.   


Итог вывод объединения, благодаря привязкам car id & person car id.

1. Какие существуют связи таблиц? дайте примеры (создайте таблицы со связями) и скрины прикрепите к ответам.

Связи создаются с помощью внешних ключей. Виды связей:

* Многие ко многим
* Один ко многим
* Одик к одному

**Many to many:**

Представим компанию, в которой работник может иметь одну и более должностей, должность может быть у одного и более работников.

Таблица Employee содержит id, age, name сотрудника.

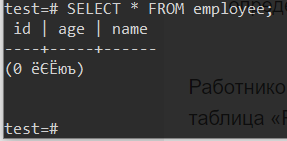
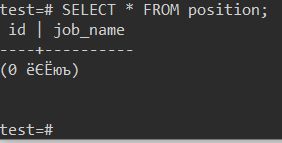
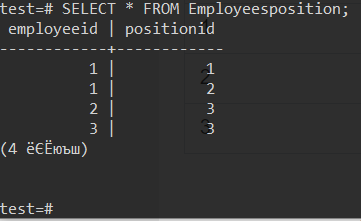


Таблица Position содержит id, job\_name.



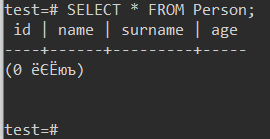
Имея две таблицы, нам нужно создать еще одну EmployeePosition для описания связей многие ко многим.



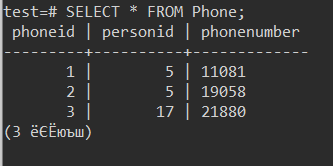
Обратите внимание, работнику под id = 1 соответствуют две должности, также можно сказать, что должность positionId = 3 принадлежит двум сотрудникам.

**One to many:**

Стандартное и наиболее распространённое отношение в таблицах. Создадим БД, которая ведет учет данных о пользователях. У пользователя есть: имя, фамилия, возраст, номера телефонов.



При этом у каждого пользователя может быть от одного и больше номеров телефонов (многие номера телефонов). Таблица Phone выглядит так:

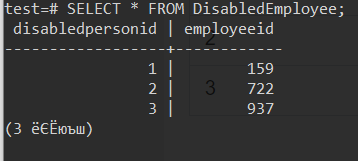


Данная таблица представляет три номера телефона. При этом номера телефона с id 1 и 2 принадлежат пользователю с id 5. А вот номер с id 3 принадлежит пользователю с id 17.

**One to one:**

Представим, что на работе вам дали задание написать БД для учета всех работников для HR. Начальник уверял, что компании нужно знать только об имени, возрасте и телефоне работника. Вы разработали такую БД и поместили в нее всю 1000 работников компании. И тут начальник говорит, что им зачем-то нужно знать о том, является ли работник инвалидом или нет. Наиболее простое, что приходит в голову — это добавить новый столбец типа bool в вашу таблицу. Но это слишком долго вписывать 1000 значений и ведь true вы будете вписывать намного реже, чем false (2% будут true, например).

Более простым решением будет создать новую таблицу, назовем ее «DisabledEmployee». Она будет выглядеть так:



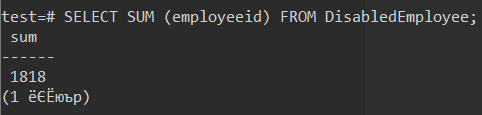
Но это еще не связь один к одному. Дело в том, что в такую таблицу работник может быть вписан более одного раза, соответственно, мы получили отношение один ко многим: работник может быть несколько раз инвалидом. Нужно сделать так, чтобы работник мог быть вписан в таблицу только один раз, соответственно, мог быть инвалидом только один раз. Для этого нам нужно указать, что столбец EmployeeId может хранить только уникальные значения. Нам нужно просто наложить на столбец EmloyeeId ограничение unique.



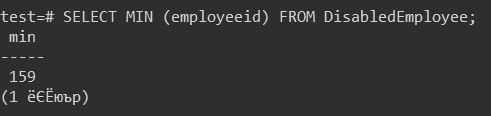
1. Какие бывают агрегатные функции?

Агрегатные функции – это математические функции, применяемые к набору данных и выдающие одно результирующее значение. Наиболее часто применяются агрегатные функции SQL SUM, MIN, MAX, AVG и COUNT.

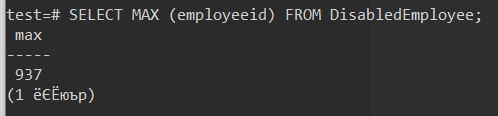
Функция SQL SUM возвращает сумму значений столбца таблицы базы данных. Она может применяться только к столбцам, значениями которых являются числа.



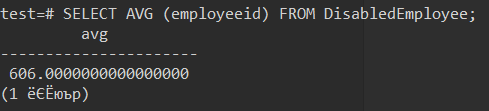
Функция SQL MIN также действует в отношении столбцов, значениями которых являются числа и возвращает минимальное среди всех значений столбца.



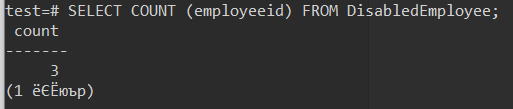
Аналогично работает и имеет аналогичный синтаксис функция SQL MAX, которая применяется, когда требуется определить максимальное значение среди всех значений столбца.



Указанное в отношении синтаксиса для предыдущих описанных функций верно и в отношении функции SQL AVG. Эта функция возвращает среднее значение среди всех значений столбца.



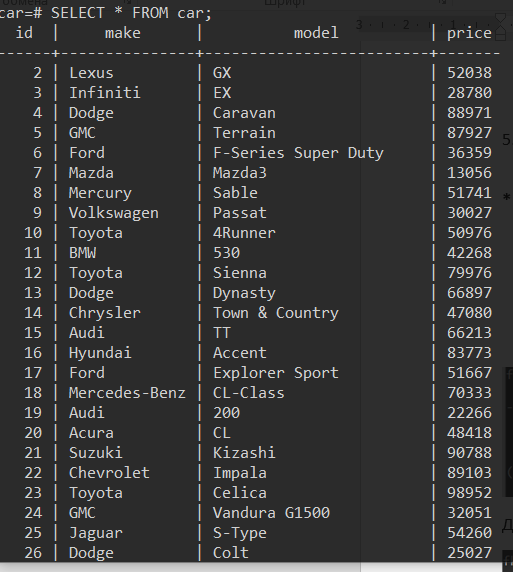
Функция SQL COUNT возвращает количество записей таблицы базы данных.



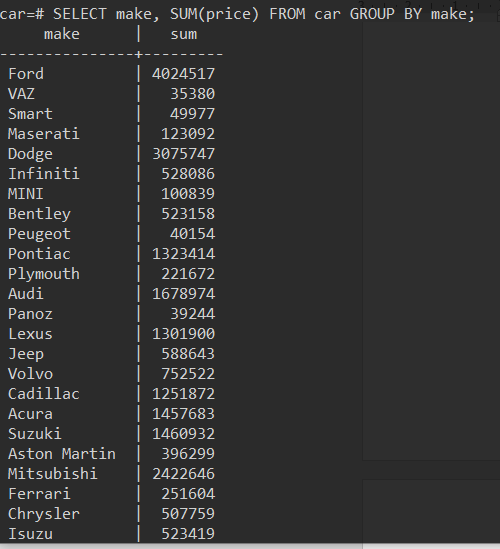
1. Напишите все что знаете про группировку с примерами запросов.

Группы создаются с помощью предложения GROUP BY оператора SELECT. Синтаксис SELECT **\*** FROM имя\_таблицы WHERE условие GROUP BY поле\_для\_группировки.

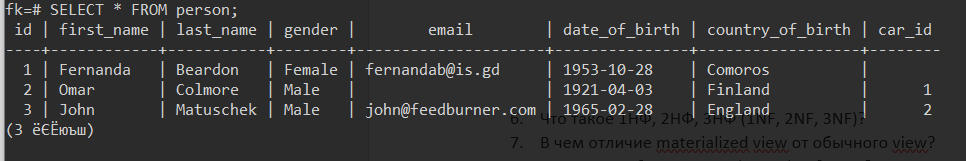
Предположим у нас есть таблица car которая содержит следующие данные:



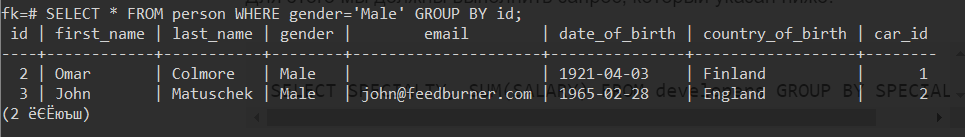
Допустим нам нужно узнать сумму цены за модели каждого производителя. Для этого сделаем:



Предположим у нас есть таблица person которая содержит следующие данные:



Допустим необходимо вывести всех мужчин, и сгруппируем их по id.



1. Что такое 1НФ, 2НФ, 3НФ (1NF, 2NF, 3NF)?

Нормальные формы — это требование, предъявляемое к структуре таблиц в теории реляционных баз данных. Она (они) нужны чтобы устранить из базы избыточные функциональные зависимости между таблицами.

**Первая нормальная форма (1NF):**

Критерии: Все строки должны быть различными. Все элементы внутри ячеек должны быть не списками, а атомарными (неделимыми). Нужно: Устранить одинаковые строки, создать отдельную таблицу для каждого набора связанных данных. Идентифицировать каждый набор связанных данных с помощью первичного ключа, то есть, добавить уникальный id для каждой строки.

**Вторая нормальная форма (2NF):**

Таблица должна находиться в первой нормальной форме. Любое её поле, не входящее в состав первичного ключа, функционально полно зависит от первичного ключа. Нужно: Создать отдельные таблицы для наборов значений, относящихся к нескольким записям и cвязать эти таблицы с помощью внешнего ключа.

**Третья нормальная форма (3NF):**

Таблица находится во второй нормальной форме. Любой её не ключевой атрибут функционально зависит только от первичного ключа. Нужно: Вынести в отдельную таблицу потенциальные первичные ключи.

1. В чем отличие materialized view от обычного view?

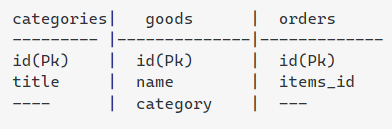
Materialized view основаны на диске и периодически обновляются на основе определения запроса.

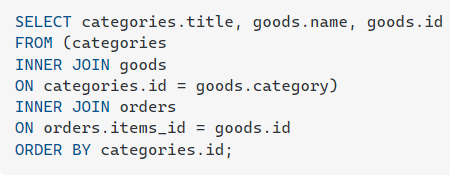
View являются только виртуальными и запускают определение запроса при каждом доступе к ним. Представление — это просто именованный запрос. Он ничего не хранит. Когда есть запрос в представлении, он выполняет запрос определения представления. Фактические данные взяты из таблицы.

1. Можно ли объединить 3 (4, 5, 2) таблицы?

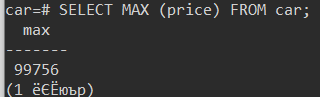
Фактически объединять таблицы лучше не стоит, так как Best Practice это разделять таблицы для удобства читабельности и структурированности.

С другой стороны таблицы можно объединять в запросах:

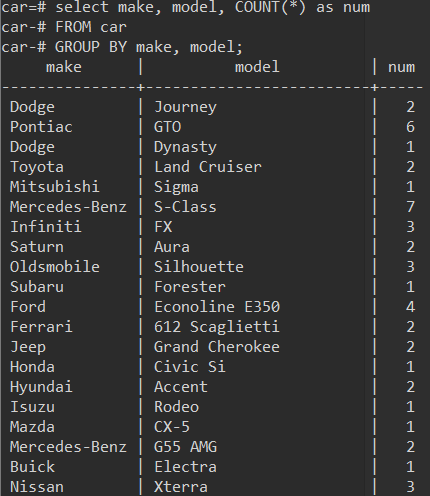




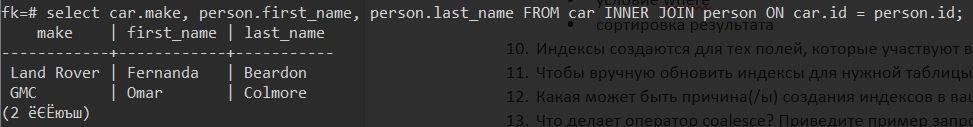
1. Сделать запрос где есть:
   * выборка: функция агрегации + обычные поля



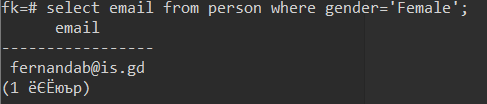
* + группировка по нескольким полям



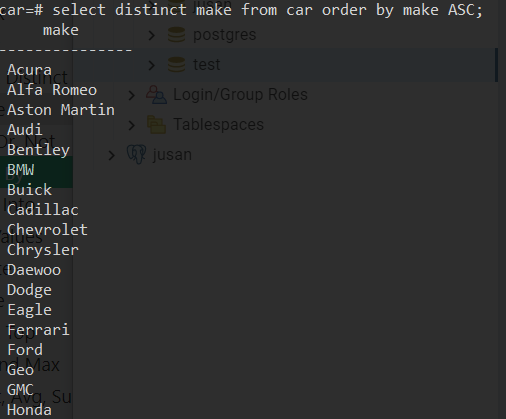
* + join нескольких таблиц



* + условие where



* + сортировка результата



1. Индексы создаются для тех полей, которые участвуют в условиях каких? (запросы)

Индексы – это специальные таблицы, которые могут быть использованы поисковым двигателем базы данных, для ускорения получения данных. Необходимо просто добавить указатель индекса в таблицу. Индекс помогает ускорить запросы на получение данных (SELECT [WHERE]), но замедляет процесс добавления и изменения записей (INSERT, UPDATE). Индексы могут быть добавлены или удалены без влияния на сами данные.

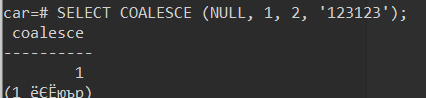
1. Чтобы вручную обновить индексы для нужной таблицы, нужно выполнить команду \_\_\_\_\_\_\_\_

ALTER INDEX index\_name REBUILD;

1. Какая может быть причина(/ы) создания индексов в вашей БД?

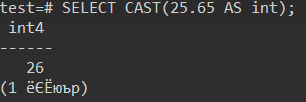
Индексы являются способом ускорения запросов на чтение путем сортировки строк таблицы в соответствии с столбцом. Эффект индекса не заметен для небольших баз данных, но, если имеется большое количество строк, это может значительно повысить производительность. Вместо проверки каждой строки таблицы сервер может выполнять двоичный поиск по индексу.

1. Что делает оператор сoalesce? Приведите пример запроса.

COALESCE возвращает первое выражение из списка параметров, неравное NULL. В качестве параметров могут выступать выражения любого типа данных, в том числе и подзапросы. В случае если все параметры равны NULL, то функция вернет значение NULL. 

1. Существует ли преобразование с одного типа в другой в Postgresql? Если да, то как это делать?

В реализациях языка SQL может быть выполнено неявное преобразование типов. Функция CAST преобразует выражение одного типа к другому.



Если же необходимо какое-то дополнительное форматирование, то можно использовать функцию CONVERT.

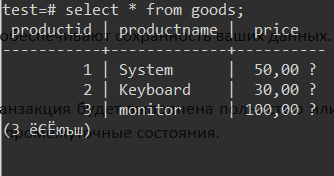
1. Напишите одну функцию и один триггер. Код и скрины БД вставить.
2. Напишите транзакцию и объясните в чем отличие от обычного запроса.

Транзакция — это осуществление одного или нескольких изменений базы данных. Для управления транзакциями используются следующие команды.

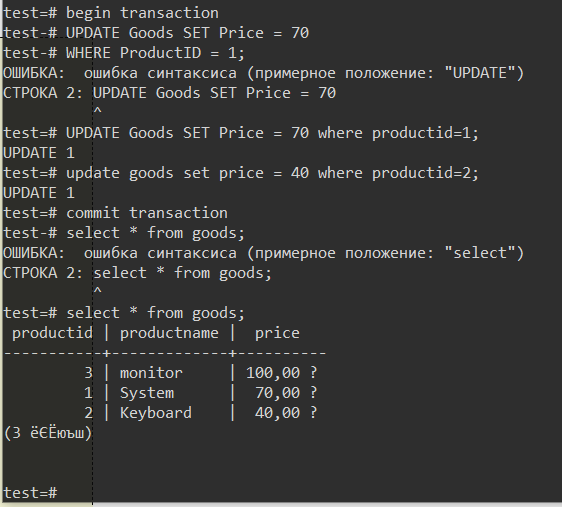
* COMMIT — сохранить изменения.
* ROLLBACK — отменить изменения.
* SAVEPOINT — создает точки сохранения в группах транзакций. (RELEASE SAVEPOINT удаляет)
* SET TRANSACTION — помещает имя в транзакцию.

Транзакции используются только с командами INSERT, UPDATE, DELETE.

Допустим есть таблица:



После внесения изменений через транзакцию:



1. Что такое ACID?

Требования ACID — набор требований, которые обеспечивают сохранность ваших данных. Что особенно важно для финансовых операций.

* Atomicity – Атомарность:

Атомарность гарантирует, что каждая транзакция будет выполнена полностью или не будет выполнена совсем. Не допускаются промежуточные состояния.

* Consistency – Согласованность:

Транзакция, достигающая своего нормального завершения (EOT — end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных.

* Isolation – Изоляция:

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.

* Durability – Надежность:

Если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.