Создаем БД shop

postgres=# CREATE DATABASE shop;

Подключаемся к БД shop

postgres=# \c shop

Вы подключены к базе данных "shop" как пользователь "postgres".

shop=#

В SQL есть такая команда \d которая проверяет, что у нас сейчас есть в БД, какие таблицы и какие сигмансы последовательности: Did not find any relations. Сейчас соответственно у нас БД пустая.

shop=# \d

Отношения не найдены.

shop=#

Создадим сущность customer, указываем поля, SERIAL – означает, что поле будет уникальное 1,2,3…автоматически плюсоваться, name varchar(255), строчка длиной максимум 255 символов, phone varchar(30), email varchar(255), так мы создали таблицу customer

shop=# create table customer(

shop(# id SERIAL PRIMARY KEY,

shop(# name varchar(255),

shop(# phone varchar(30),

shop(# email varchar(255)

shop(# );

CREATE TABLE

shop=#

Проверим что сейчас есть в БД \d, мы создали таблицу customer, и seq последовательность customer\_id\_seq -–это последовательность уникальных идентификаторов в таблице customer, служебная сущность в postgreSQL

\d

Посмотрим описание таблицы customer, мы видим поля, типы default значения у этих полей, есть индекс \_pkey PRIMARY KEY "customer\_pkey" PRIMARY KEY, btree (id). PRIMARY KEY - это первичный ключ, ссылка на каждый уникальный объект в нашей таблице. Customer – это клиенты в нашем магазине. Id – это PRIMARY KEY. У каждого клиента будет уникальный первичный ключ, уникальный идентификатор цифровой, по которому мы точно сможем идентифицировать нашего каждого покупателя в нашем интернет-магазине

\d customer

По аналогии создадим таблицу product

shop=# create table product(

shop(# id serial primary key,

shop(# name varchar(255),

shop(# description text,

shop(# price integer);

CREATE TABLE

Проверим что сейчас есть в БД \d, появилась вторая таблица

shop=# \d

Создадим еще одну таблицу product\_photo – фотографии для наших продуктов, product\_id – это цифровое поле, тип этого поля integer и оно является ссылкой на таблицу product его поля id, это и есть тот самый Relation, та самая связь между таблицами.

Поле product\_id в сущности Product-photo (в таблице Product-photo) ссылается на сущность (таблицу Product) поля id.

shop=# create table product\_photo(

shop(# id serial primary key,

shop(# url varchar(255),

shop(# product\_id integer references product(id));

CREATE TABLE

Проверим что сейчас есть в БД \d, появилась третья таблица

shop=# \d

shop=# \d product\_photo

"product\_photo\_product\_id\_fkey" FOREIGN KEY (product\_id) REFERENCES product(id)

Обратим внимание на fkey. FOREIGN KEY – это внешний ключ, связь между двумя сущностями (таблицами) в БД PostgreSQL. В Данном случае фотография ссылается на товар.

Создадим еще одну сущность (таблицу) cart – корзина

shop=# create table cart(

shop(# customer\_id integer references customer(id),

shop(# id serial primary key);

CREATE TABLE

Cart – это корзина или заказ в интернет магазине, у которого есть идентификатор, и у каждого заказа есть клиент. Сущность(таблица) Cart поле Customer\_id ссылается на сущность (таблицу) customer поле (id).

shop=# \d cart

"cart\_customer\_id\_fkey" FOREIGN KEY (customer\_id) REFERENCES customer(id)

Нам нужно теперь создать промежуточную таблицу, отношение многое ко многим между нашим заказом и товарами в этом заказе

shop=# create table cart\_product(

shop(# cart\_id integer references cart(id),

shop(# product\_id integer references product(id));

CREATE TABLE

Посмотрим описание таблицы cart\_product с двумя полями, которые ссылаются на внешние таблицы, на таблицу продуктов и на таблицу корзин.

Посмотрим сколько у нас всего таблиц

shop=# \d

Заполним наши таблицы данными. Создадим какое-то колличество customer, т.е. вставляем в таблицу customer поля name, phone, email, (id можно не указывать – он будет указываться автоматически PostgreSQL) values – какие поля мы вставляем

shop=# \d customer

Записываем данные в поля customer

shop=# insert into customer(name, phone, email) values ('Micke', '404-246', 'micke@gmail.com');

INSERT 0 1

shop=# insert into customer(name, phone, email) values ('Nick', '404-322', 'nick@gmail.com');

INSERT 0 1

выберем теперь всех людей которые есть в таблице customer

shop=# SELECT \* FROM customer;

замечательно, добавили двух людей в таблицу customer

id | name | phone | email

----+-------+---------+-----------------

1 | Micke | 404-246 | micke@gmail.com

2 | Nick | 404-322 | nick@gmail.com

создадим два продукта, нужно заполнить поля name, description, price

shop=# \d product

shop=# insert into product(name, description, price) values ('iPhone', 'iPhone 7 Plus', '550');

INSERT 0 1

shop=# insert into product(name, description, price) values ('Apple Watch', 'Series 7', '450');

INSERT 0 1

посмотрим, что у нас есть в продуктах

shop=# SELECT \* FROM product;

id | name | description | price

----+-------------+---------------+-------

1 | iPhone | iPhone 7 Plus | 550

2 | Apple Watch | Series 7 | 450

добавим фотографии iPhone и Apple watch

shop=# \d product\_photo

здесь url и ссылка product\_id, вставим product\_photo(url, product\_id)

id Iphone – здесь 1, а фотографию Apple Watch мы вставлять не будем.

shop=# insert into product\_photo (url, product\_id) values ('iPhone\_photo', 1);

shop=# SELECT \* FROM product\_photo;

id | url | product\_id

----+--------------+------------

1 | iPhone\_photo | 1

Чтоб выбрать отдельные поля, а не SELECT \* - все. Мы можем выбрать только поле URL

shop=# select url from product\_photo;

url

--------------

iPhone\_photo

или

shop=# select url, product\_id from product\_photo;

url | product\_id

--------------+------------

iPhone\_photo | 1

Например, нам нужно получить(достать) все фотографии товаров с названиями этих товаров, т.е. мы хотим получить какую-то таблицу сформировать на выходе, которая будет содержать в себе url изображений(первая колонка), вторая колонка(название товара). Чтоб это сделать нам нужно использовать JOIN – его задача соединить разные таблицы БД в некую финальную таблицу, которую мы получаем на выходе

pp – это alias, можем указывать каждой таблице некий alias

shop=# SELECT \* FROM product\_photo pp;

id | url | product\_id

----+--------------+------------

1 | iPhone\_photo | 1

shop=# SELECT pp. \* FROM product\_photo pp;

id | url | product\_id

----+--------------+------------

1 | iPhone\_photo | 1

оn – это по каким полям мы соединяем таблицы

shop=# select pp. \* from product\_photo pp left JOIN product p on p.id=pp.product\_id;

Соединим две таблицы, нужно отметить что, в полученной таблице id, url, product\_id – это поля из таблицы product photo, а поле name – это поле из таблицы product.

Используя left join мы получили финальную таблицу, данные из двух таблиц. Это и есть задача оператора JOIN, в связке с LEFT JOIN.

SELECT делаем из первой таблицы product\_photo и второй таблицы product

Product\_photo – это левая таблица

Product – правая таблица

Главная таблица у нас левая – соответственно выбираются строки где присутствуют записи в левой таблице (главной таблице)

shop=# select pp.\*, p.name from product\_photo pp left JOIN product p on p.id=pp.product\_id;

Т.к. у нас есть один продукт для которого мы не создали фотографию, если мы посмотрим продукты, у нас два продукта

shop=# select \* from product;

а если мы посмотрим product\_photo, то у нас фотография только для iphone\_photo

shop=# select \* from product\_photo;

Попробуем добавить фотографию товара на несуществующий продукт, для этого нам нужно удалить внешний ключ FOREIGN KEY, который предназначен, чтоб БД была целостная, чтоб не было специально таких ситуаций (как бы мы хотим создать понимание работы LEFT JOIN – RIGHT JOIN), как работают разные модели соединения таблиц.

Посмотрим еще раз описание таблицы product\_photo

shop=# \d product\_photo

У нас есть такой FOREIGN KEY, давайте его удалим

shop=# alter table product\_photo drop constraint product\_photo\_product\_id\_fkey;

ALTER TABLE

Проверим таблицу

shop=# \d product\_photo

Теперь мы можем создать фотографию на несуществующий в БД продукт, т.е. у нас есть два продукта с идентификаторами 1 и 2

shop=# SELECT \* FROM product;

shop=# SELECT \* FROM product\_photo;

Сейчас мы создадим фотографию на несуществующий продукт, у нас нет product\_id 100, поэтому создаем фотографию для несуществующего продукта

shop=# insert into product\_photo (url, product\_id) values ('unknown\_photo', 100);

Получим две фотографии из таблицы product\_photo;

shop=# SELECT \* FROM product\_photo;

Теперь проверим как отработает наш запрос с LEFT JOIN. У нас есть две таблицы product\_photo и product.

product\_photo – это левая таблица, главная таблица в которой должны присутствовать те записи, которые попадают в финальную выборку и правая таблица – это product. Если в правой таблице не находится подходящих записей, то подставляются NULL в выходные колонки результата.

Т.е. iphone\_photo ссылается на iphone(т.к. он у нас существует, поле name заполнено iphone)

unknown\_photo ссылается на несуществующий продукт, соответственно поле name пустое NULL.

shop=# select pp.\*, p.name from product\_photo pp left join product p on p.id=pp.product\_id;

id | url | product\_id | name

----+---------------+------------+--------+--------------

1 | iPhone\_photo | 1 | iPhone

2 | unknown\_photo | 100 |\_\_\_\_\_\_\_

Как работает RIGHT JOIN. Теперь у нас главная таблица стала правая таблица product, поэтому product name iPhone и Apple Watch, но по продукт Apple Watch у нас не создана фотография, поэтому url и product\_id у нас NULL

shop=# select pp.\*, p.name from product\_photo pp RIGHT JOIN product p on p.id=pp.product\_id;

id | url | product\_id | name

----+--------------------+----------------+--------------------

1 | iPhone\_photo | 1 | iPhone

| \_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_ | Apple Watch

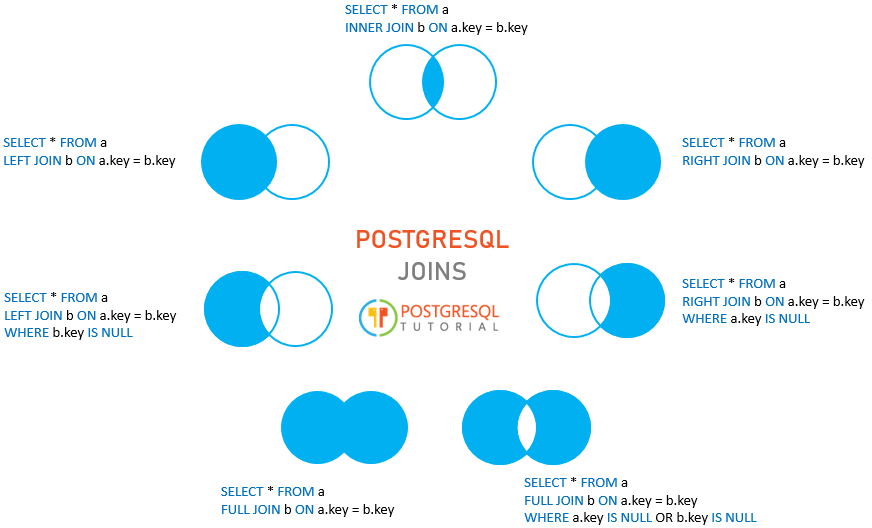
Модеь – как заполнить LEFT OUTER JOIN, те записи которые находятся в левой таблице, попадают в финальную выгрузку, а для тех которых нет соответствия в правой таблице, просто добавляются NULL.

LEFT OUTER JOIN и LEFT JOIN – это тоже самое, идентичные названия.

RIGHT OUTER JOIN – главная становится правая таблица и для нее записи всегда заполнены, а для левой таблицы могут дополняться NULL, если там соответствия не найдены.

Обычно используется запрос LEFT JOIN.

<https://www.postgresqltutorial.com/postgresql-joins/>



INNER JOIN – записи которые присутствуют в обеих таблицах.

Просто нужно запомнить мнемоническую картинку с JOIN

shop=# select pp.\*, p.name from product\_photo pp INNER JOIN product p on p.id=pp.product\_id;

id | url | product\_id | name

----+--------------------+----------------+--------

1 | iPhone\_photo | 1 | iPhone

Удалим вторую фотографию, которая ссылается на несуществующий продукт

shop=# SELECT \* FROM product\_photo;

shop=# DELETE FROM product\_photo WHERE id=2;

DELETE 1

Теперь у нас только одна запись в таблице product\_photo

shop=# SELECT \* FROM product\_photo;

Обновим url фотографии в таблице product\_photo. Таким образом обновили поле url таблицы product\_photo, установив поле url в значение iphone\_image\_2 для всех записей id у которых = 1, для ровно одной записи, таким образом обновление данных в таблице и удаление данных из таблицы выполняется простым способом. UPDATE такую-то таблицу, обновить значение таких-то полей для записей у которых id = 1

shop=# update product\_photo set url='iphone\_image\_2' where id=1;

UPDATE 1

Заполним больше данных, чтоб поработать с более сложными SQL запросами SELECT

Проверим таблицу cart, она у нас должна быть пустая

shop=# SELECT \* FROM cart;

Например создадим для Micke заказ

shop=# SELECT \* FROM customer;

Сейчас мы создадим один заказ для Micke

shop=# insert into cart (customer\_id) values (1);

INSERT 0 1

Проверим созданный заказ

shop=# SELECT \* FROM cart;

Нам необходимо заполнить заказ его продуктами, т.к. у нас есть два продукта Iphone и Apple Watch, далее добавим оба этих продукта в корзину для Micke

shop=# SELECT \* FROM product;

У нас есть таблица cart\_prpduct

shop=# \d cart\_product

Добавим первый заказ id =1, первая корзина, продукт 1 (1,1), и первая корзина продукт 2 (1, 2)

shop=# insert into cart\_product (cart\_id, product\_id) values (1,1), (1,2);

Проверим таблицу cart\_product, у нас добавилось два продукта в первую корзину

shop=# SELECT \* FROM cart\_product;

cart\_id | product\_id

---------+------------

1 | 1

1 | 2

Например нужно получить имена клиентов с общей суммой их заказов. Т.к. у нас есть таблица customer, в нем есть поле name

shop=# SELECT \* FROM customer;

Нам нужно сформировать таблицу с двумя колонками, первое поле – это имя клиентов, вторая колонка – это общая сумма заказов этого клиента. Для этого нам нужно выполнить JOIN нескольких таблиц.

Для поля customer дадим alias c. Получим первую колонку.

shop=# SELECT c.name FROM customer c;

name

-------

Micke

Nick

Выполняем LEFT JOIN, таблицы cart к таблице customer

shop=# SELECT c.name FROM customer c LEFT JOIN cart on cart.customer\_id=c.id;

name

-------

Micke

Nick

Добавим в выходное поле cart\_id id

shop=# SELECT c.name, cart.id FROM customer c LEFT JOIN cart on cart.customer\_id=c.id;

name | id

-------+----

Micke | 1

Nick |

Идентификатор корзины card\_id. У Micke есть одна корзина, у Nick пока корзины нет, значит NULL

shop=# SELECT c.name, cart.id as card\_id FROM customer c LEFT JOIN cart on cart.customer\_id=c.id;

name | card\_id

---------+---------

Micke | 1

Nick |

Для Micke добавилось две строки в выходной результат, потому что у нас в cart\_product для Micke есть две записи, т.е. у него в корзине есть два продукта.

shop=# SELECT c.name, cart.id as card\_id FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id;

name | card\_id

----------+---------

Micke | 1

Micke | 1

Nick |

Выведем два продукта в корзине для Micke, product\_id 1 и 2

shop=# SELECT c.name, cart.id as cart\_id, cp.product\_id FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id;

name | cart\_id | product\_id

----------+----------+------------

Micke | 1 | 1

Micke | 1 | 2

Nick | |

Выполняем LEFT JOIN, таблицы product поля price. Т.е. у нас есть выходная таблица в которой есть имена клиентов, id всех корзин этих клиентов(т.е. всех заказов этих клиентов), следующая колонка product\_id – это товар клиента, и price – это цена товара

shop=# SELECT c.name, cart.id as cart\_id, cp.product\_id, p.price FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id;

name | cart\_id | product\_id | price

----------+----------+----------------+-------

Micke | 1 | 1 | 550

Micke | 1 | 2 | 450

Nick | | |

Наша задача сейчас просуммировать все price по клиентам и сгруппировать данные, чтоб у нас было две колонки name и суммарный price.

shop=# SELECT c.name, p.price FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id;

name | price

----------+-------

Micke | 550

Micke | 450

Nick |

Сгруппируем данные по имени GROUP BY NAME

shop=# SELECT c.name, p.price FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id GROUP BY name;

ОШИБКА: неоднозначная ссылка на столбец "name"

СТРОКА 1: ....id left join product p on p.id=cp.product\_id GROUP BY name;

Ошибка, т.к. непонятно, что делать с ценой, неодназначная ссылка(ambiguous) поле name есть в customer и в product. Значит Micke у нас купил на 1000. Nick купил у нас NULL

shop=# SELECT c.name, sum(p.price) FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id GROUP BY c.name;

name | sum

----------+------

Nick |

Micke | 1000

У него в корзине есть два товара

shop=# SELECT \* FROM product;

id | name | description | price

----+-------------+---------------+-------

1 | iPhone | iPhone 7 Plus | 550

2 | Apple Watch | Series 7 | 450

Если мы хотим, чтоб у Nick было 0 а не NULL, воспользуемся coalesce. Так мы получили сумму всех заказов по каждому клиенту

shop=# SELECT c.name, coalesce(sum(p.price), 0) as orders\_sum FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id GROUP BY c.name;

name | orders\_sum

----------+------------

Nick | 0

Micke | 1000

Нам нужно отсортировать этот список, чтоб наверху были самые топовые клинты

shop=# SELECT c.name, coalesce(sum(p.price), 0) as orders\_sum FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id GROUP BY c.name order by orders\_sum;

name | orders\_sum

-------+------------

Nick | 0

Micke | 1000

shop=# SELECT c.name, coalesce(sum(p.price), 0) as orders\_sum FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id GROUP BY c.name order by orders\_sum desc;

name | orders\_sum

-------+------------

Micke | 1000

Nick | 0

Например, нам нужно получить, только тех клиентов, которые у нас, что-то купили, у которых orders\_sum > 0

shop=# SELECT c.name, coalesce(sum(p.price), 0) as orders\_sum FROM customer c LEFT JOIN cart on cart.customer\_id=c.id LEFT JOIN cart\_product cp on cp.cart\_id=cart.id left join product p on p.id=cp.product\_id GROUP BY c.name having sum(p.price)>0;

name | orders\_sum

-------+------------

Micke | 1000

В чем отличие WHERE от having? WHERE – фильтрует строки, having – фильтрует группы. Т.е. having – это агрегирующая функция sum(сумма), count(кол-во строк)

Мы берем пишем какой-то SELECT WHERE – получаем строчки, делаем GROUP BY – получаем группы, сгруппированные значения, и если нам нужно сгруппированные значения еще дополнительно отфильтровать мы используем HAVING

Рассмотрим запрос сортировку по имени

shop=# select \* from customer order by name;

id | name | phone | email

----+-------+---------+-----------------

1 | Micke | 404-246 | micke@gmail.com

2 | Nick | 404-322 | [nick@gmail.com](mailto:nick@gmail.com)

# Если ORDER BY неправильно работает можно рассмотреть как правильно создавать БД в правильной кодировке

SELECT \* FROM customer order by name using ~<~;

Если нам нужно выполнить ограничение одной записи LIMIT

shop=# select \* from customer order by name LIMIT 1;

id | name | phone | email

----+-------+---------+-----------------

1 | Micke | 404-246 | micke@gmail.com

Если нам нужно выполнить получение второй записи LIMIT, OFFSET задает смещение, это часто используется для пагинации, для разбивки результатов какой-то выдачи на разные страницы.

shop=# select \* from customer order by name LIMIT 1 OFFSET 1;

id | name | phone | email

----+------+---------+----------------

2 | Nick | 404-322 | nick@gmail.com

Эти аспекты одинаково работают в управлении любой БД

Функции для работы с датами, как из даты получить или достать год, месяц, как складывать разные интервалы времени ORACLE DB , PotgreSQL