**Report about ICT project (website which specializes in electronics)**

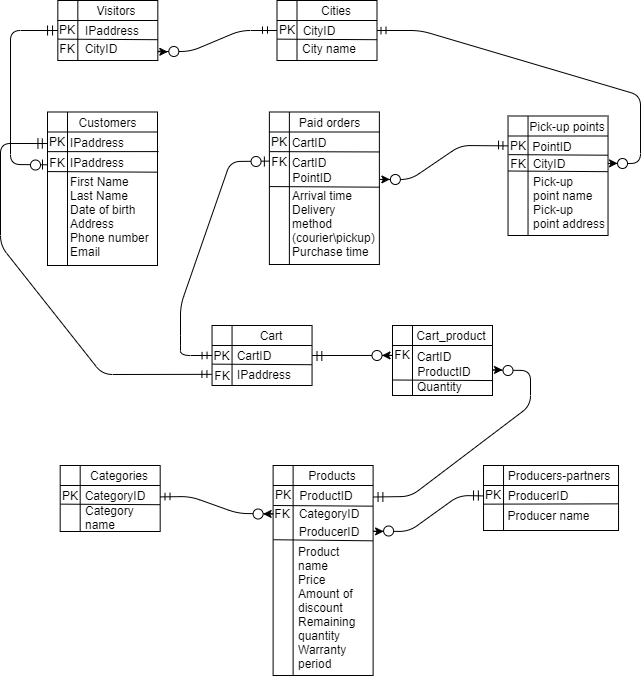
Zakir Maradzhapov, Darkan Serik

**ABSTRACT**

We decided to create database for a web site that sells electronic goods. Actually, if we rename some columns, our RED table will fit not only to electronic web shop, but also to many of other web sites that sell other types of products.

We made ERD with 10 tables. Initially there were more tables but during the work we understood it is possible to merge some of them, at the same time making ERD easier to understand and reducing its overall load.

This is the ERD itself:



The purpose of our database is to store information about products, its producers, pick-up points, as well as about customers and their purchases. The database also makes sure that each order arrives to the correct pick-up point, and sets discounts and the warranty period for products. Thanks to this, employees are not confused in a huge amount of data, and their work becomes much faster and more efficient.

**Literature review, real cases**

In 2010, brick-and-mortar retail stores of many brands (as well as ordinary entrepreneurs) began to close everywhere across America. This phenomenon was called Retail Apocalypse. In General, this phenomenon occurred due to the change in the purchasing habits of Americans towards online shopping. Also, the reason for Retail Apocalypse was the aggressive policy of such large companies as Amazon and Walmart. Retail stores simply could not compete with such giants and were forced out of the market, completely bankrupt. A particular jump in closures was noticeable in 2019. This year, about 9,300 stores closed, almost 60% more stores than in 2018. However, closing retail outlets does not always lead to bankruptcy. Many brands are closing less profitable stores and investing their available resources in outlets that bring in more money.

References:

<https://en.wikipedia.org/wiki/Retail_apocalypse>

<https://www.national.biz/the-retail-apocalypse-is-amazon-prime-day-to-blame-for-store-closures/>

However, it’s not just Amazon that poses a threat to stores. 2020 was an incredibly tough year, especially for retail shops. Huge number of them was closed due to lockdown caused by COVID-19. At the same time, a lot of franchise even go bankrupt, for example, New York & Co(378 stores), Pizza Hut(300 Stores), Papyrus(254 stores) etc. From the other hand, web stores and delivery services have become extremely popular. Compared with 2019, online purchases increase by an average of 9%, while in-store purchases decreased by 11%.

References:

<https://www.digitalcommerce360.com/article/coronavirus-impact-online-retail/>

<https://www.forbes.com/sites/walterloeb/2020/07/06/9274-stores-are-closing-in-2020--its-the-pandemic-and-high-debt--more-will-close/?sh=35c46f2b729f>

**METHODOLOGY**

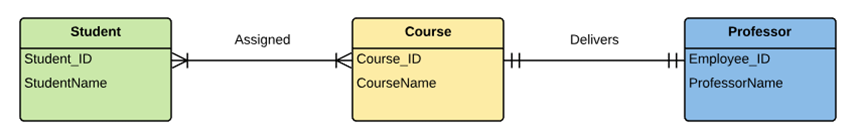
USED METHODS

Creating our database, we decided to make it in ERD(entity-relationship diagram) format.

We think this is the most effective method of visualization due to the many factors:

1. It is easily understandable by everyone who knows ERD symbols, so we don’t have to spend time to explain how our database works.
2. ERD is very flexible so we can effortlessly change some table and column names, relations and even logic during the short period.
3. You don’t have to study hard to build a simple ERD and learn its basic rules.

Example of ERD:



After making our ERD, our goal was to make sure that not a single piece of data was lost. Another goal was to make the data fit together. To do this, we wrote the code for our ERD in postgresql. In addition to being extremely reliable, postgresql also helps us enter data efficiently. To check inserted data for correctness, we used several CHECK constraints, some of which were implemented using functions.



To draw our ERD, we used site, which is called draw.io. The site provide us with all necessary tools using which we could easily depict tables, write names of their columns and also depict tables’ relation and special symbols. Moreover, the website automatically saves every changes in a project and allows us to save in lot of formats (jpg, png etc.)



BUSINESS RULES

(all the cities, pick-up points, products, its categories and producers are included to their tables beforehand)

When someone visits our site, our database automatically includes him or her to 'Visitors' table and adds appropriate city ID. When a visitor decided to add a product to cart, the database also adds him or her to 'Customers' table (now a person is recorded in 2 tables at the same time). After that, the database receives the necessary data from the person and creates a new line in 'Cart' table with person's IP address (when a customer quits the site, database deletes his or her cart from ‘Cart’ table). 'Cart\_product' table, in turn, records what products the person added to cart. If the person has bought the products in the cart, the database adds the cart to 'Orders' table. After that, the database gets the information about the delivery method and to what pick-up point it must be sent, records purchase time(in case we need to check whether the warranty was expired) and arrival time. Also, using functions and CHECK constraints, the database makes sure that everything is filled in correctly and the data matches each other (otherwise, the order will probably be sent to the pick-up point from the totally different city). If the person has chosen delivery by a courier, we should deliver the order straight to the customers, so we don’t have to add any point\_ID to the ‘Orders’ table. Person’s city and address are located in ‘Visitors’ and ‘Customers’ tables respectively and can be easily retrieved.

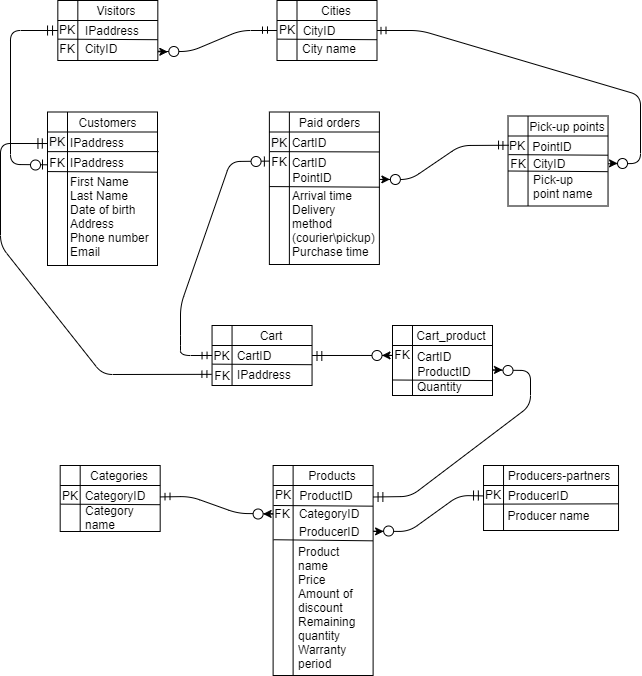
ERD’S STRUCTURE

Overall, ERD have 10 tables. Let’s shortly discuss each of them.

Firstly, we start from 3 main tables:

1. Customers

This table contains all the necessary information about customers, such as customers’ IP address, name and surname, their birth date, address, phone number and email. ‘Address’ column is used in case the customer chose delivery by a courier. As for other columns, we need them in order to contact the client if necessary. The table is connected with tables ‘Visitors’ and ‘Customers’ via IP address, which initially was a primary key of ‘Visitors’ table.



CREATE TABLE Customers(

IP\_address VARCHAR(15) PRIMARY KEY,--as every customer is a visitor, but not every visitor is a customer

f\_name VARCHAR(32) not null,

l\_name VARCHAR(32) not null,

birth\_date DATE not null,

address VARCHAR(100) not null,--in case a customer shoose delivery by a courier

phone\_num DEC(16) not null,

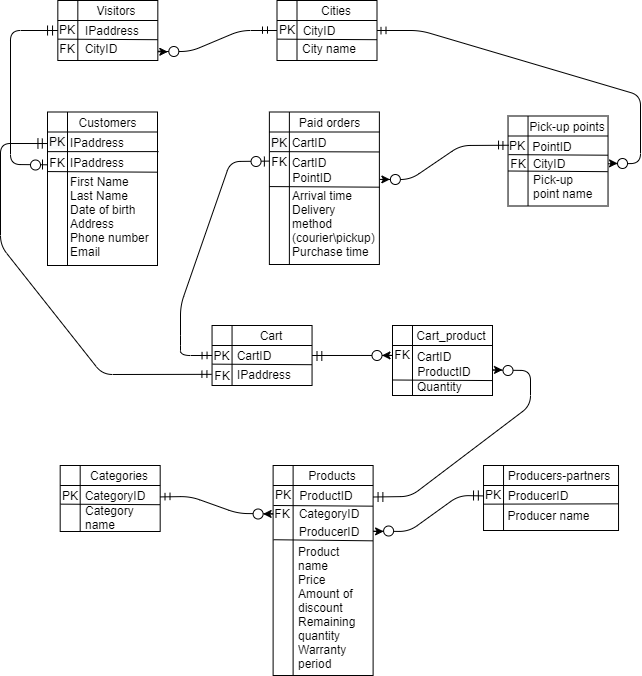
--email CHAR(50) not null,--we will add this column afterwards

FOREIGN KEY (IP\_address) REFERENCES Visitors(IP\_address) /\*as a customer and visitor is the same person, but he or she is recorded in 2 tables\*/

);

2. Paid orders

If products in ‘Cart’ were purchased, the cart’s ID must be added to ‘Paid orders’ table. This table is connected with ‘Cart’ table via cart\_ID, which is also a primary key of ‘Paid orders’ table. Regarding ‘Pick-up points’ table, they are linked via point ID. Apart from columns mentioned earlier, the table records arrival time, purchase time and delivery method. Also, using purchase time, we can identify whether the warranty of a product was expired.



CREATE TABLE Orders(

cart\_ID INT PRIMARY KEY,

point\_ID INT,/\*(is null only if delivered by a courier) in case the order is delivered by a courier,

we don't have to go to the pick-up point\*/

purchase\_time DATE not null,--we need purchase time in order to know when the warranty of bought products ends

arrival\_time DATE not null,

delivery\_method CHAR(7) not null, --only two variants exist('courier' or 'pickup')

--CONSTRAINT arrival\_after\_purchase CHECK (arrival\_time > purchase\_time),--we will add this CHECK constraint afterwards

CONSTRAINT correct\_city CHECK (get\_city\_of\_customer(cart\_ID) = get\_city\_of\_point(point\_ID)),/\*a CHECK constraint that

verifies that every paid order is shipped to the correct pick-up point, which must be in the same city as the buyer

of the order\*/

CONSTRAINT correct\_delivery\_method CHECK ((point\_ID is null AND delivery\_method = 'courier')

OR (point\_ID is not null AND delivery\_method = 'pickup')),

/\*if an order will be delivered by a courier, a customer doesn't have to to to the pick-up point, so point\_ID shouldn't

be mentioned. In case the customer decided to pick up the order in the available pick-up point, point\_ID is mandatory\*/

FOREIGN KEY (Cart\_ID) REFERENCES Cart(Cart\_ID), /\*link the Orders table with Cart table so we can know what products

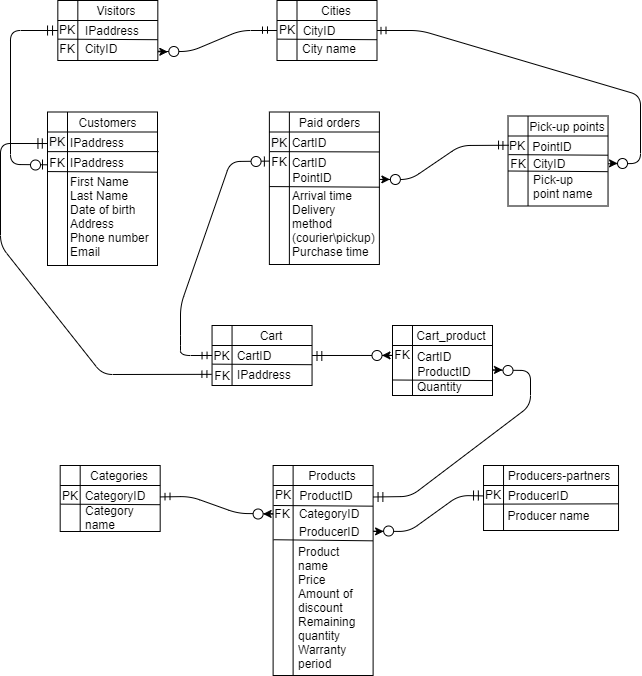
an order cointains, the products's price etc\*/

FOREIGN KEY (Point\_ID) REFERENCES Pickup\_points(Point\_ID)--to what pick-up point the order must be delivered

);

3. Products

‘Products’ table contains information about product’s name, its producer and category, price, amount of discount, quantity in stock and warranty. Also, it gives unique ID to every product. The table is linked with ‘Cart\_product’, ‘Producers-partners’ and ‘Category’ tables via product ID, producer ID and category ID respectively. In price column, we wrote the prices that already includes the discount.



CREATE TABLE Products(

product\_ID INT PRIMARY KEY,

product\_name VARCHAR(50),

category\_ID INT not null,

producer\_ID INT not null,

price INT not null CONSTRAINT no\_free\_products CHECK (price > 0),/\*(the price in the column have already

included discounts) we need no\_free\_products CHECK constraint in order not to skip mistakes that make products free\*/

discount\_amount DECIMAL(3,2) CONSTRAINT allowed\_disc\_range CHECK (discount\_amount > 0 AND discount\_amount < 1),

/\* We need allowed\_disc\_range CHECK constraint in order to prevent a 100%(1.00) discount on a product.

In case a discount wasn't set to the product, discount\_amount is null\*/

remain\_quantity INT not null,

warranty\_period VARCHAR(50),/\*(is null only if a product doesn't have a warranty)

also unclear is DATE (or DAY) data type appropriate to this column?\*/

FOREIGN KEY (category\_ID) REFERENCES Categories(category\_ID),

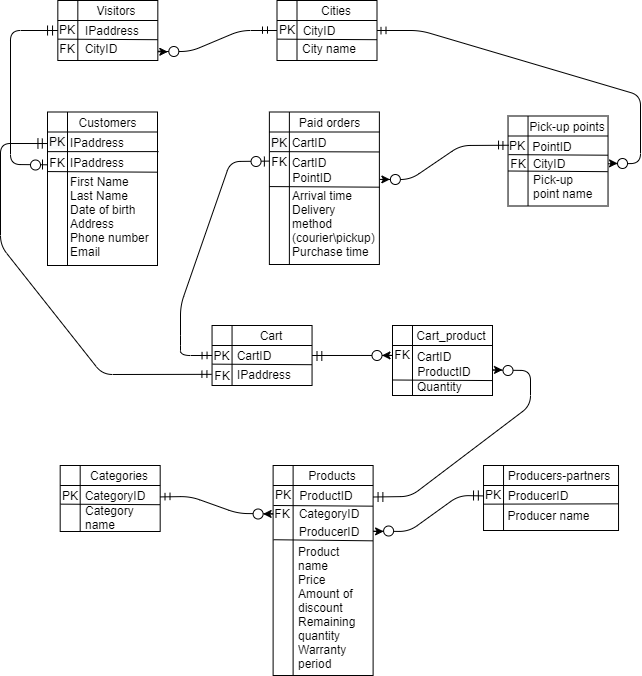
FOREIGN KEY (producer\_ID) REFERENCES Producers(producer\_ID)

);

Let’s discuss another 7 tables that are also crucial:

4. Producers’

‘Producers’ table contains information about name of every producers our shop cooperates with. Also, the table gives unique ID to every producer, by which the table is linked with ‘Products’ table. The table can contain data about producers that wasn’t mentioned in ‘Products’ table yet.



CREATE TABLE Producers(

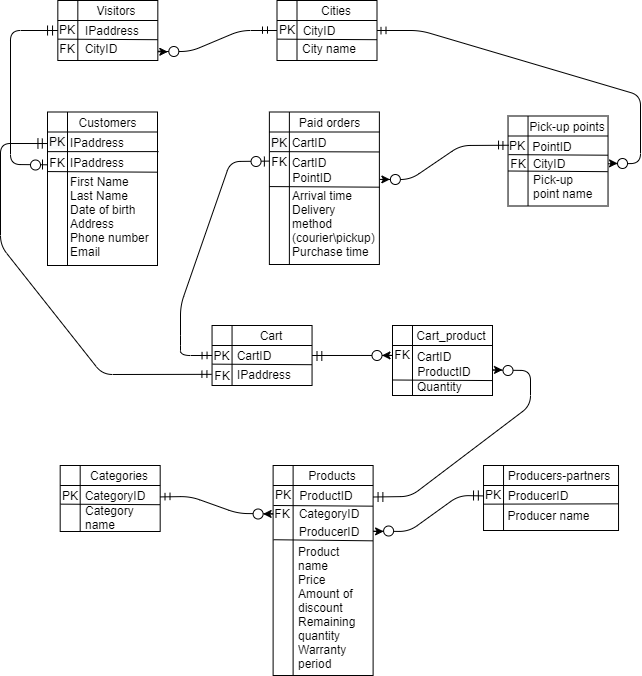
producer\_ID INT PRIMARY KEY,

producer\_name VARCHAR(32) not null

);

5. Categories

‘Categories’ table contains information about name of every category our shop sells. Also, the table gives unique ID to every category, by which the table is linked with ‘Products’ table. The table can contain data about categories that wasn’t mentioned in ‘Products’ table yet.



CREATE TABLE Categories(

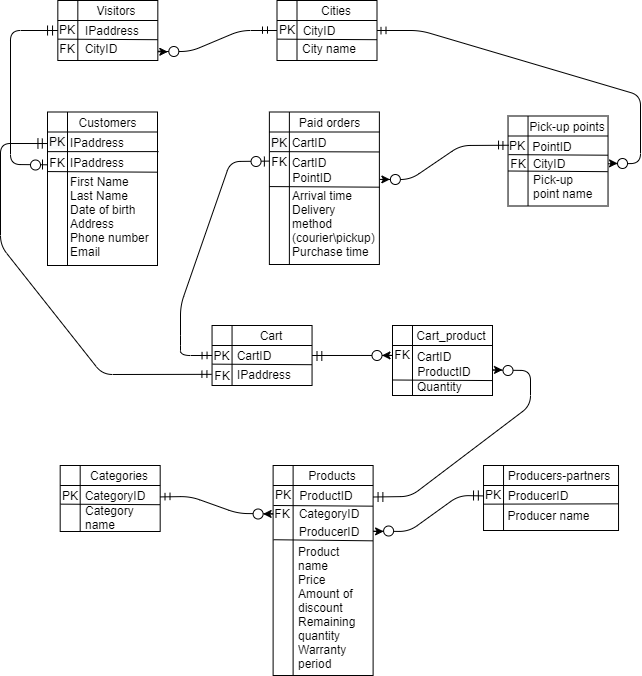
category\_ID INT PRIMARY KEY,

category\_name VARCHAR(32) not null

);

6. Cart

‘Cart’ table contains IP addresses of customers and creates unique cart ID for each of them. It takes customers’ IDs from ‘Customers’ table and passes Cart’s ID to Cart\_product table. The important thing is that in order to become a customer, a person just need to add something to a cart. After that, a customer can have an empty cart, and person’s cart exists until he or she leave the site



CREATE TABLE Cart(

cart\_ID INT PRIMARY KEY,

IP\_address VARCHAR(15),--we will add not null constraint to this column afterwards (IP\_address VARCHAR(15) not null)

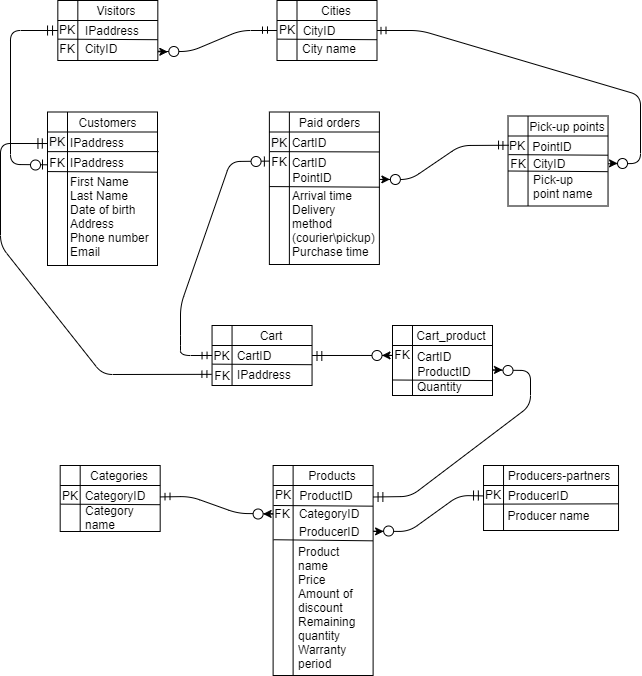
FOREIGN KEY (IP\_address) REFERENCES Customers(IP\_address)/\*we link ID address with Customer table

because visitors refers to people who haven't added anything to cart yet\*/

);

7. Cart\_product

This table shows what kind of products a cart from ‘Cart’ table contains and also shows the products’ quantity. The ‘Cart\_product’ table is linked with ‘Cart’ and ‘Products’ tables by one to one relation and via cart ID and product ID respectively.



CREATE TABLE Cart\_product(

product\_ID INT not null,

cart\_ID INT not null,

quantity INT not null CONSTRAINT existing\_product CHECK (quantity > 0),/\*because a customer must buy at least

1 product in order to add it into the cart\*/

CONSTRAINT not\_out\_of\_stock CHECK (in\_stock(product\_ID) >= quantity),

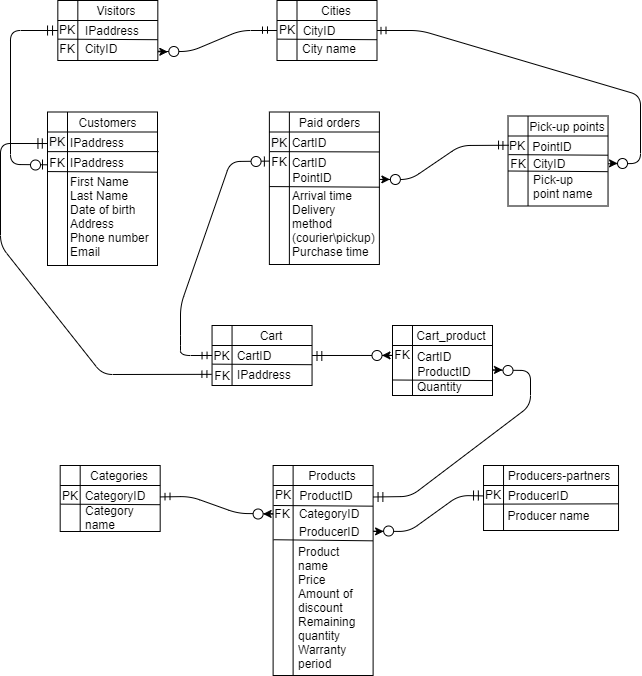
FOREIGN KEY (product\_ID) REFERENCES Products(product\_ID),

FOREIGN KEY (cart\_ID) REFERENCES Cart(cart\_ID)

);

8. Visitors

A ‘Visitors’ table gathers IP addresses of all people who visited our website. Also, it asks them to choose their city. The table is connected with ‘Cities’ tables, which contains all the cities, which were offered to visitors to choose from. Also, the ‘Visitors’ table is linked with ‘Customers’ table via IP address and has zero to one relation with it.



CREATE TABLE Visitors(

IP\_address VARCHAR(15) PRIMARY KEY,

city\_ID INT not null,

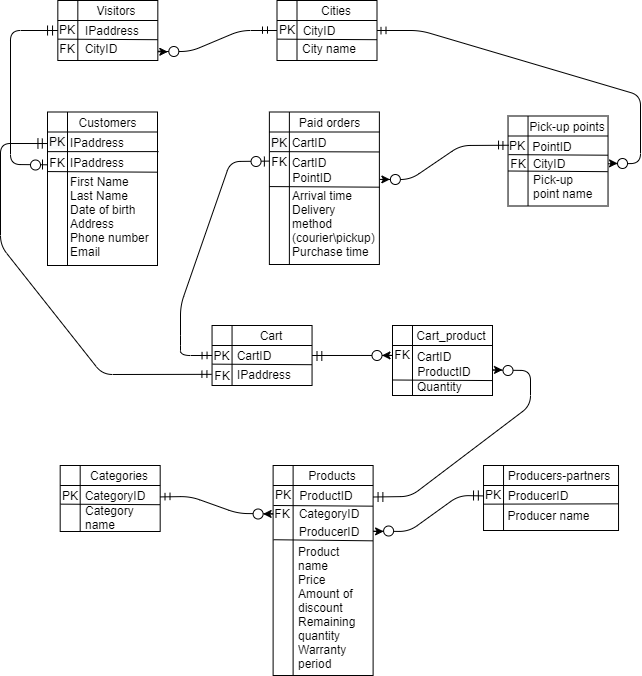
gender CHAR(6) not null,--we will delete this column afterwards

FOREIGN KEY (city\_ID) REFERENCES Cities(city\_ID)--city of a visitor

);

9. Cities

A ‘Cities’ table contains certain cities, which were inserted to the table beforehand, and also creates a unique ID for each of them. The table is linked with ‘Visitors’ table and ‘Pick-up points’ table via city ID and by many to zero relation.



CREATE TABLE Cities(

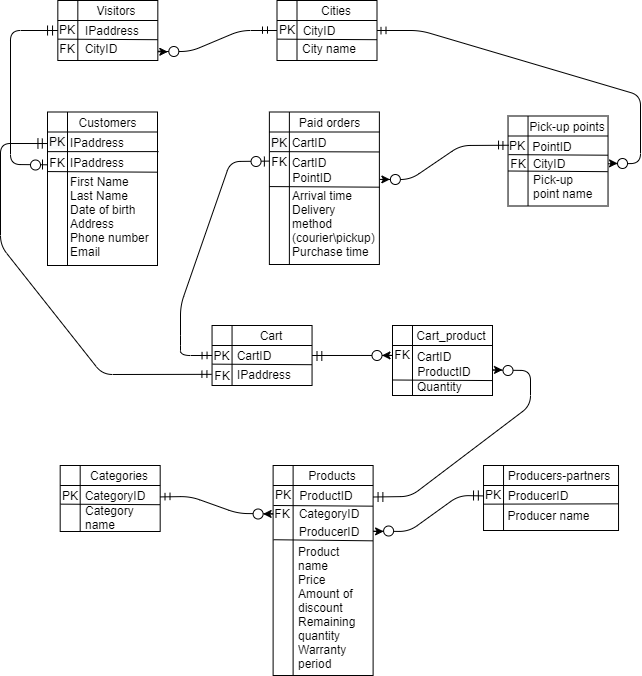
city\_ID INT PRIMARY KEY,

city\_name VARCHAR(32) not null

);

10. Pick-up points

A ‘Pickup\_points’ table keeps the data about every point where a customer can pick up the order, such as a pick-up point’s name and its address. Also, the table assigns unique Id to every point. The table is connected with ‘Cities’ table by one to one relation (via city\_id) and ‘Orders’ table by many to zero relation (via point\_id).



CREATE TABLE Pickup\_points(

point\_ID INT PRIMARY KEY,

city\_ID INT not null,

pickup\_name VARCHAR(32) not null,--we will rename this column afterwards(point\_name)

point\_address VARCHAR(50) not null,

FOREIGN KEY (city\_ID) REFERENCES Cities(city\_ID)--in what city pick-up point is located

);

VALUES

In cases where we can use data generator(using mockaroo.com), we took data from it. But most frequently, we generated the data by our own. It was necessary because mockaroo generates fully random values, without any links between them. Instead of this, in many tables we should insert the data that refers to data from another table. For example, in ‘Orders’ table the product with name ‘PlayStation 5’ must have the producer ID which refers to ‘Sony’, not to ‘Microsoft’.

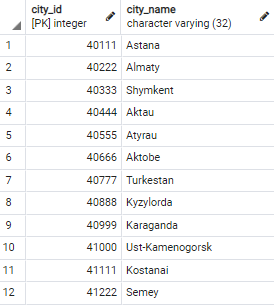
**CASE STUDY/RESULTS**

After writing code for tables in postgresql, let’s start to insert values into it (all the data was simulated).

Firstly, we should fill by data all the tables that don’t have any foreign keys and thus don’t depend on other tables.

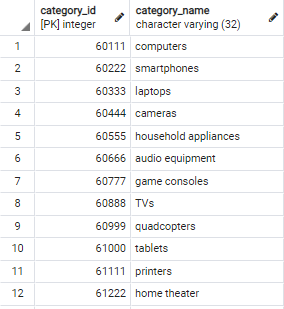
1. City

INSERT INTO Cities(city\_ID, city\_name) VALUES



In order to show operating principles of relations with ‘Pick-up points’ and ‘Visitors’ tables, we won’t use ID of Turkestan(40777) in ‘Pick-up points’ table and ID of Semey(41222) in ‘Visitors’ table

2. Categories

INSERT INTO Categories(category\_ID, category\_name) VALUES

In order to show operating principles of relations, we won’t use ID of ‘quadcopters’(60999) in ‘Products’ table.

3. Producers-partners

INSERT INTO Producers(producer\_ID, producer\_name) VALUES

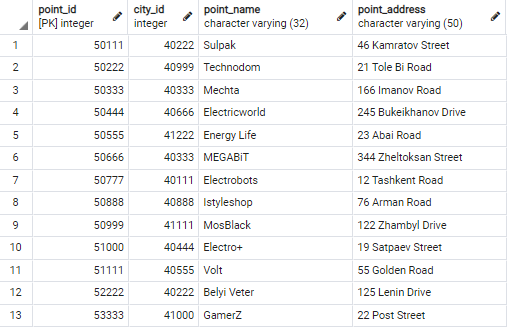


In order to show operating principles of relations, we won’t use ID of ‘Razer’(20008) in ‘Products’ table.

After that, we can start filling by data other 7 tables:

4. Pick-up points

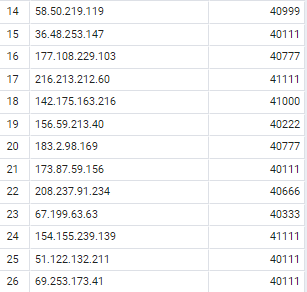
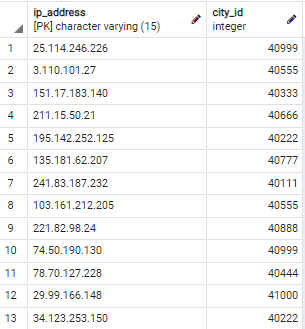
INSERT INTO Pickup\_points(point\_ID, city\_ID, point\_name, point\_address) VALUES



In order to show operating principles of relations, we won’t use all 13 points in ‘Paid orders’ table.

5. Visitors

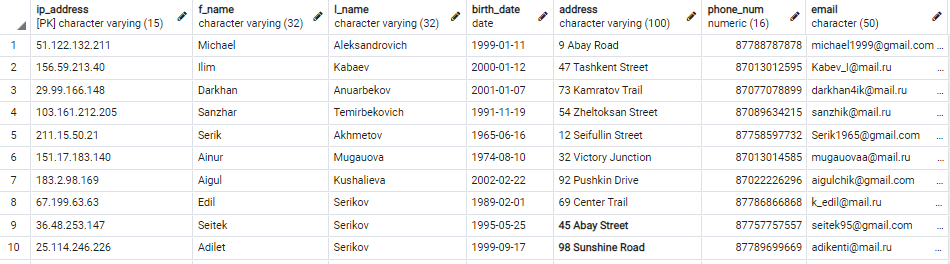
INSERT INTO Visitors(IP\_address, city\_ID) VALUES



We inserted IP addresses which were generated by a robot. Also, Semey(41222) wasn’t used. We won’t use every IP in ‘Customers’ table, because not every visitor is a customer.

6. Customers

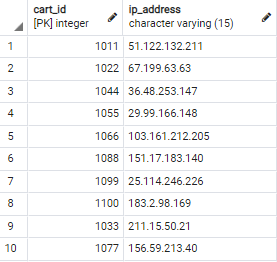
INSERT INTO Customers(IP\_address, f\_name, l\_name, birth\_date, address, phone\_num, email) VALUES



IP address of every customer will be used in ‘Cart’ table, i.e. every customer will have a cart.

7. Cart

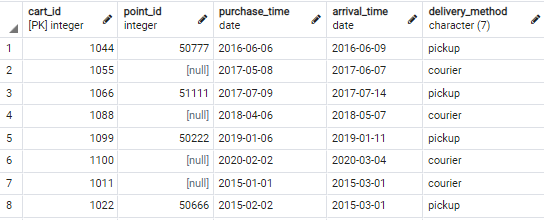
INSERT INTO Cart(cart\_ID, IP\_address) VALUES



Cart may not contain any products (In case a person added something to cart and then deleted it, the person’s cart still exists).

8. Orders

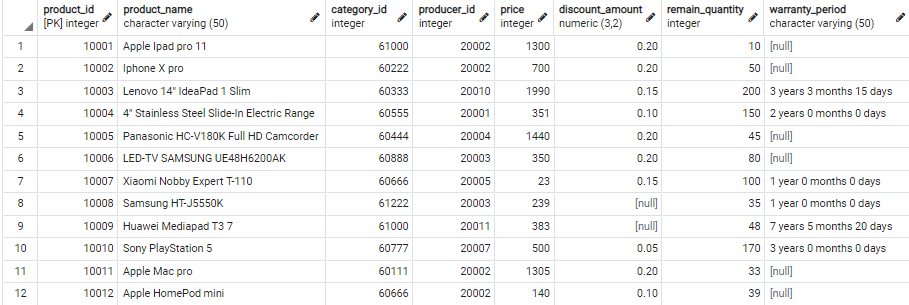
INSERT INTO Orders(cart\_ID, point\_ID, purchase\_time, arrival\_time, delivery\_method) VALUES

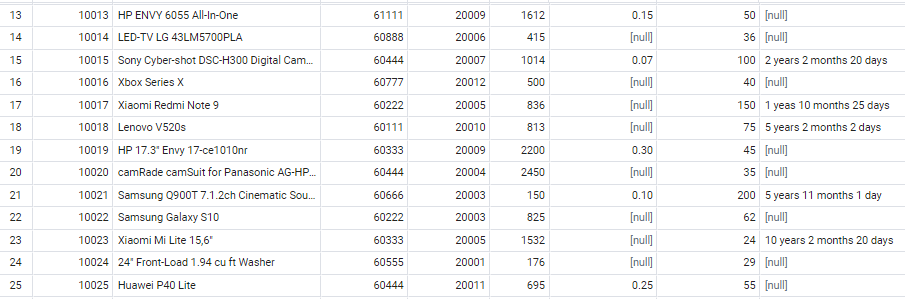


There wasn’t some of the customers, because not every customer buys products in cart. Also, the value of point\_id will be null if the order is delivered by a courier.

9. Products

INSERT INTO Products(product\_ID, product\_name, category\_ID, producer\_ID, price, discount\_amount, remain\_quantity, warranty\_period) VALUES

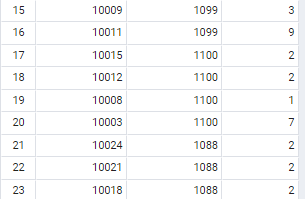
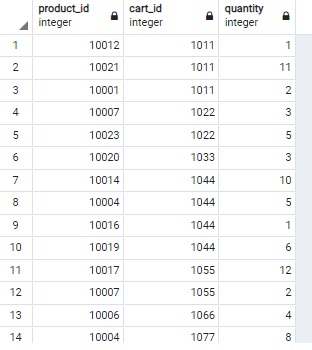




In this table, ‘Razer’ (2008) from ‘Producers’ and ‘quadcopters’(60999) from ‘Categories’ table weren’t used. Also, if a product doesn’t have a discount or warranty period, the value must be null.

10. Cart\_product

INSERT INTO Cart\_product(product\_ID,cart\_ID, quantity) VALUES



In this table, cart IDs(in case the person bought couple products) and product(in case the product was bought by few people) IDs can repeat several times.

After inserting all the values, let’s create some test cases

1. Select cities and addresses of clients that have chosen delivery by a courier.

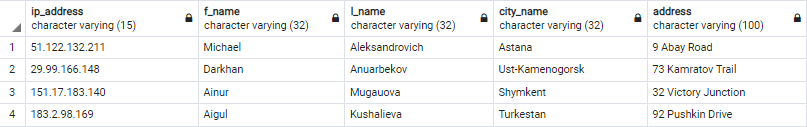
select Customers.ip\_address, Customers.f\_name, Customers.l\_name, Cities.city\_name, Customers.address FROM Orders

inner join Cart on Cart.cart\_id = Orders.cart\_ID

inner join Customers on Customers.ip\_address = Cart.ip\_address

inner join Visitors on Customers.ip\_address = Visitors.ip\_address

inner join Cities on Cities.city\_id = Visitors.city\_id WHERE Orders.delivery\_method = 'courier';

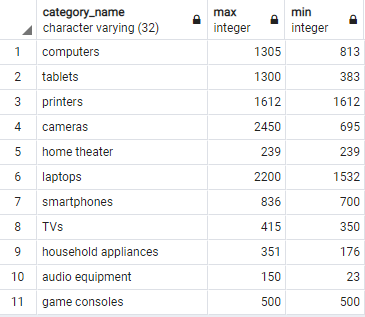


Data from this table will help us to determine to what city and address our courier should deliver the purchased order.

2. Select category name and one of its goods with max price

select distinct c.category\_name, max(p.price), min(p.price) from products p, categories c

where p.category\_id = c.category\_id group by c.category\_name



Information from this table will help us to determine the highest and lowest price segment in each category. It will also clarify each category’s price range and the average price of products.

3. Select paid products purchased without discount

select p.product\_id as Product\_ID,

p2.producer\_name as Producer\_Name,

o.cart\_id as Order\_ID,

o.purchase\_time as Order\_Purchased\_Date,

o.arrival\_time as Arrival\_Date,

c2.f\_name,

c2.l\_name

from products p

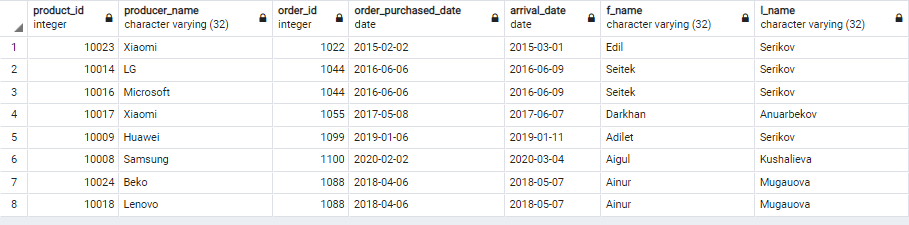
inner join producers p2 on p2.producer\_id = p.producer\_id

inner join cart\_product cp on p.product\_id = cp.product\_id

inner join cart c on cp.cart\_id = c.cart\_id

inner join orders o on c.cart\_id = o.cart\_id

inner join customers c2 on c2.ip\_address = c.ip\_address where p.discount\_amount is null



Information from this table are the products without discount selling well. In case In case some of them are difficult to sell, we can set a discount to the products.

4. Count of sold products for each producer

select

count(o.cart\_id),

p2.producer\_name

from products p

inner join producers p2 on p2.producer\_id = p.producer\_id

inner join cart\_product cp on p.product\_id = cp.product\_id

inner join cart c on cp.cart\_id = c.cart\_id

inner join orders o on c.cart\_id = o.cart\_id

group by p2.producer\_name



Data from this table can help to understand us products of which producer are most (or least) popular on our site.

5. Select paid products and their quantity with customer's city names from Astana, Almaty, Shymkent

select p.product\_id as Product\_ID,

cp.quantity as Quantity,

p2.producer\_name as Producer\_Name,

o.purchase\_time as Order\_Purchased\_Date,

c3.city\_name

from products p

inner join producers p2 on p2.producer\_id = p.producer\_id

inner join cart\_product cp on p.product\_id = cp.product\_id

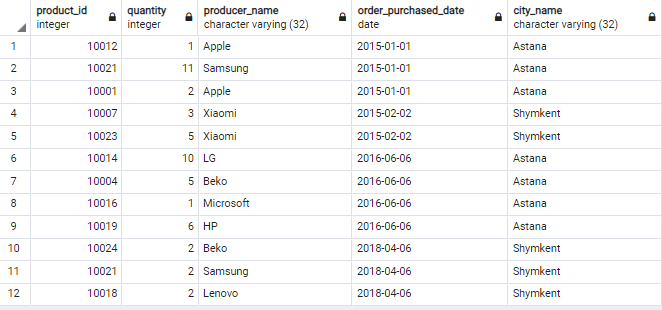
inner join cart c on cp.cart\_id = c.cart\_id

inner join orders o on c.cart\_id = o.cart\_id

inner join customers c2 on c2.ip\_address = c.ip\_address

inner join visitors v on c2.ip\_address = v.ip\_address

inner join cities c3 on v.city\_id = c3.city\_id and c3.city\_name in ('Astana', 'Almaty', 'Shymkent');



Information from this table can help us to make analysis, which will show what producers and products are popular in certain cities in different periods of time. Also, it will show us how many people from certain cities buy products from our site.

6. Select producers and their IDs whose products exist in Products table

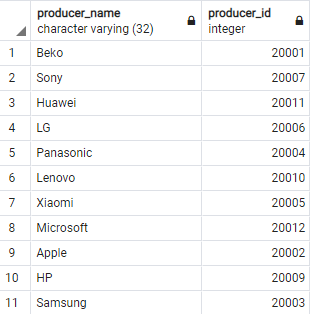
select distinct

p2.producer\_name,

p.producer\_id

from products p

inner join producers p2 on p.producer\_id = p2.producer\_id



This output shows us producers, whose products are selling in our shop. This information will show us which manufacturers we already work closely with.

***(Appendix)***

During the project, we encountered an interesting problem. In order to verify that every paid order is shipped to the correct pick-up point, which must be in the same city as the buyer of the order, we wanted to add a CHECK constraint to “Orders” table. Firstly, we tried to implement subquery in the CHECK constraint,

(Table Orders)  
CONSTRAINT pickup\_point\_in\_correct\_city CHECK  
((SELECT [Pickup\_points.city](https://vk.com/away.php?to=http%3A%2F%2FPickup_points.city&cc_key=" \t "_blank)\_ID FROM Orders INNER JOIN Pickup\_points ON Orders.point\_ID = Pickup\_points.point\_ID) =

(SELECT [Visitors.city](https://vk.com/away.php?to=http%3A%2F%2FVisitors.city&cc_key=" \t "_blank)\_ID FROM Orders INNER JOIN Cart ON Orders.cart\_ID = Cart.cart\_ID  
INNER JOIN Customers ON Cart.IP\_address = Customers.IP\_address INNER JOIN Visitors ON Customers.IP\_address = Visitors.IP\_address))

but an error popped up in postgresql “Subqueries are not allowed in this context. Only scalar expressions are allowed.”

Then we found out that functions can help with this, so we added a function(get\_city\_of\_point) that returns the city of a pick-up point and another function(get\_city\_of\_customer) that returns the city of a customer.

CREATE FUNCTION get\_city\_of\_point(pointID INT)

RETURNS INT

LANGUAGE plpgsql

AS

$$

DECLARE

correct\_city INT;

BEGIN

SELECT city\_ID INTO correct\_city FROM Pickup\_points WHERE point\_ID = pointID;

RETURN correct\_city;

END;

$$;

/\*we need that function in order to get the city of a given pick-up point in the following CHECK constraint

from ORDERS table (that’s because only scalar expressions are allowed in CHECK constraint)\*/

CREATE FUNCTION get\_city\_of\_customer(cartID INT)

RETURNS INT

LANGUAGE plpgsql

AS

$$

DECLARE

correct\_city INT;

BEGIN

SELECT city\_ID INTO correct\_city FROM Visitors WHERE IP\_address = (SELECT IP\_address FROM Cart WHERE cart\_ID = cartID);

RETURN correct\_city;

END;

$$;

/\*we need that function in order to get the city of a given customer in the following CHECK constraint

from ORDERS table (that’s because only scalar expressions are allowed in CHECK constraint)\*/

Using them, the CHECK constraint started to work correctly. This experience made it easy to create the next CHECK constraint using another function(in\_stock)

CREATE FUNCTION in\_stock(productID INT)

RETURNS INT

LANGUAGE plpgsql

AS

$$

DECLARE

quantity INT;

BEGIN

SELECT remain\_quantity INTO quantity FROM Products WHERE product\_ID = productID;

RETURN quantity;

END;

$$;

/\*we need that function in order to get the quantity of a product on the warehouse in the following CHECK constraint

from PRODUCTS table (that’s because only scalar expressions are allowed in CHECK constraint)\*/

Using all these functions and constraints, our database has become extremely reliable.

Sources where we found the solving of the problem:

(how to use queries in CHECK constraint)

<http://www.sql-tutorial.ru/ru/book_subqueries_in_check_constraints.html>

<https://stackoverflow.com/questions/13000698/sub-queries-in-check-constraint>

(how to create functions)

https://postgrespro.ru/docs/postgresql/9.6/sql-createfunction

https://www.postgresqltutorial.com/postgresql-create-function/