Databases Group Project

Team: Можаров Егор, Васильев Борис, Дрягин Артём, Гюрсой Александр-Эфе (DSBA-192)

Оглавление

[Project description 3](#_Toc91383437)

[Technical requirements 3](#_Toc91383438)

[Functional dependencies 3](#_Toc91383439)

[Data restrictions 4](#_Toc91383440)

[Preliminary database schema 4](#_Toc91383441)

[Reasons for Normalization 4](#_Toc91383442)

[Database Creation 5](#_Toc91383443)

[Quires for DB 7](#_Toc91383444)

[Conclusion 10](#_Toc91383445)

# Project description

In this project, our team implemented a trading platform database. It displays information on all orders placed, customer information, product statuses and characteristics, as well as information about postomats (their location and status)

# Technical requirements

1. Client information
   1. User registration
   2. User authorization
2. Order information
   1. Make an order
   2. Watch information about order
   3. Decline order
3. Item information
   1. Watch information about order
   2. Calculate total profit from items
4. Postomat information
   1. Watch information about postomat
   2. Reserve postomat for order

# Functional dependencies

Client: **Client-ID**->Name, Surname, Country, City, Address, Mail, Password, Login, Date-Registration

Order: **Purchase -ID**->Purchase-status, Client-ID, Date-Registration, Postomat-ID, Date-Finish, Date-Accept, Date-Decline

Postomat: **Postomat-ID**->Status

Item: **Item-ID**->Order-ID, Item-Status, Item-Type, Name, Price

Oreder\_status **id**->Status

Postomat\_location: **id**->Country, Address, City, id

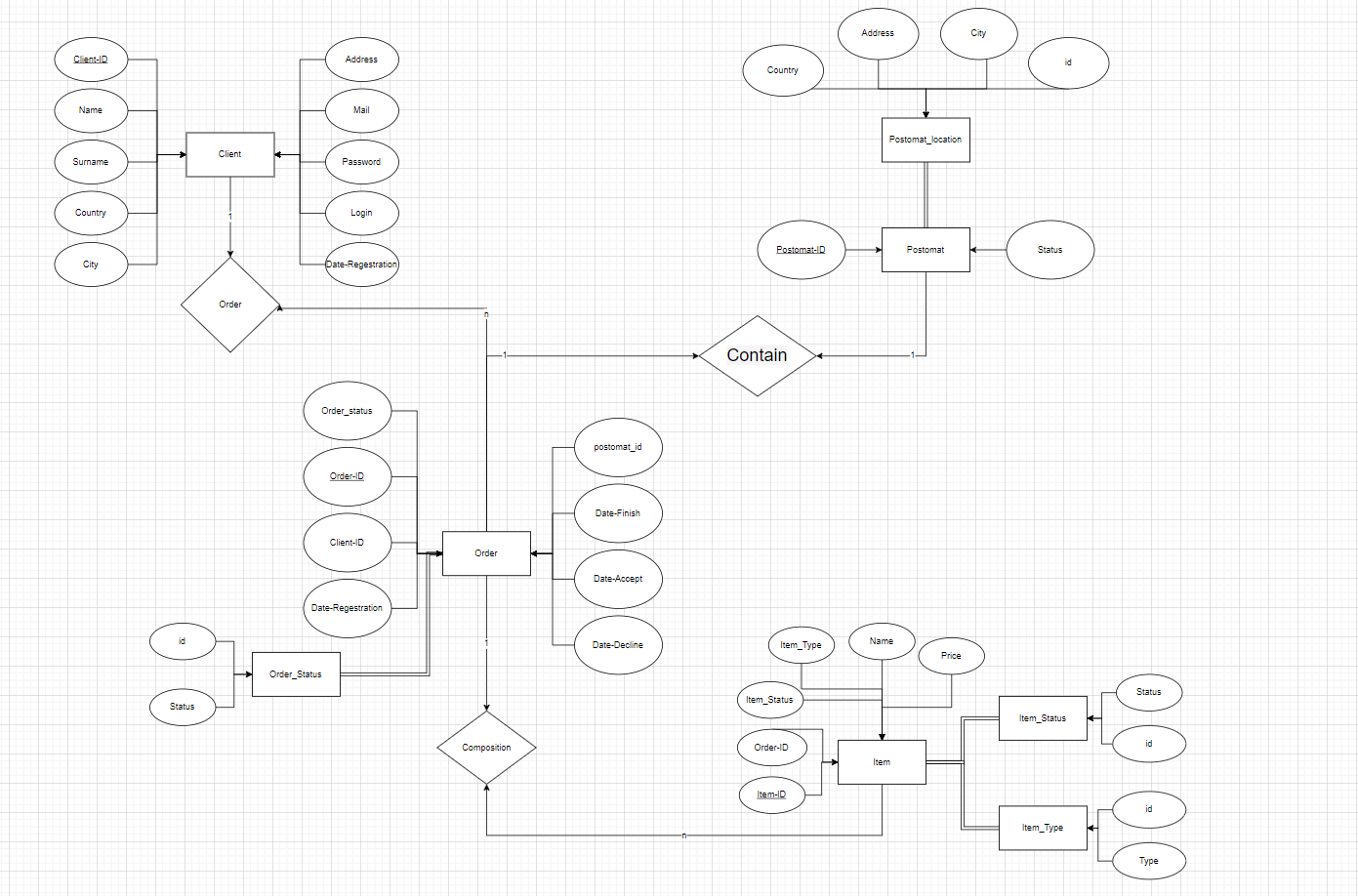
Item\_Status: **id**-> Status

Item\_Type: **id**-> Type

# Data restrictions

* Each client can make more than 0 orders
* Each item has only 1 status and only 1 type
* Each order can change status during the process delivering
* Each postomat can change status but in moment can have only one status
* Each postomat can have only one location and cannot change it

# Preliminary database schema



# Reasons for Normalization

Normalization itself is needed for more convenient structuring of data, as well as their preservation. In our case, if we want to delete an order by order-id, then we will not delete the information on our client, otherwise, when storing all the information in one place, the client's personal data could suffer. In addition, when using normalization, we avoid the problem of data redundancy. In addition to the actual advantages, we can include such positive aspects as the acceleration of queries due to a more uniform distribution of data, more convenient storage of the data itself in terms of access to them

# Database Creation

CREATE TABLE IF NOT EXISTS Client(

Client\_ID int AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Name VARCHAR(255) NOT NULL,

Surname VARCHAR(255) NOT NULL,

Country VARCHAR(255) NOT NULL,

City VARCHAR(255) NOT NULL,

Address VARCHAR(255) NOT NULL,

Mail VARCHAR(255) NOT NULL,

Password VARCHAR(255) NOT NULL,

Login VARCHAR(255) NOT NULL,

Date\_Regestration DATE NOT NULL

);

CREATE TABLE IF NOT EXISTS Purchase(

Purchase\_ID int AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Status\_ID INT REFERENCES Purchase\_Status(id) On DELETE CASCADE NOT NULL,

Client\_ID INT NOT NULL,

Postomat\_ID INT,

Date\_Regestration DATE NOT NULL,

Date\_Finish DATE,

Date\_Accept DATE,

Date\_Decline DATE

);

CREATE TABLE IF NOT EXISTS Purchase\_Status(

id INT AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Status VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS Item(

Item\_ID INT NOT NULL PRIMARY KEY,

Purchase\_ID INT REFERENCES Purchase(Purchase\_ID) On DELETE CASCADE,

Item\_Status INT REFERENCES Item\_Status(id) On DELETE CASCADE NOT NULL,

Item\_Type INT REFERENCES Item\_Type(id) On DELETE CASCADE NOT NULL,

Name VARCHAR(255) NOT NULL,

Price INT NOT NULL

);

CREATE TABLE IF NOT EXISTS Item\_Status(

id INT AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Status VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS Item\_Type(

id INT AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Type VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS Postomat(

Postomat\_ID INT AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Postomat\_location REFERENCES Postomat\_Location(id) On DELETE CASCADE NOT NULL,

Status VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS Postomat\_Location(

id INT AUTO\_INCREMENT NOT NULL PRIMARY KEY,

Country VARCHAR(255) NOT NULL,

City VARCHAR(255) NOT NULL,

Address VARCHAR(255) NOT NULL

);

# Quires for DB

**--Adding new values to Client**

INSERT INTO Client VALUES(1, "Egor", "Mozharov", "Russia", "Moscow", "Nelidovskya 23", "egor.mozharov@mail.ru", "Egor134", "Egor123", "2020-10-10");

SELECT \* from Client;

**--Adding new values to Purchase**

INSERT INTO Purchase VALUES(1, 1, 1, 1, "2020-10-15", "2020-11-14", "2020-10-15", NULL);

SELECT \* from Purchase;

**--Adding new values to Purchase\_Status**

INSERT INTO Purchase\_Status VALUES(1, "Delivered");

SELECT \* from Purchase\_Status;

**--Adding new values to Item**

INSERT INTO Item VALUES(1, 1, 1, 1, "AiphoneX", 7000);

INSERT INTO Item VALUES(2, NULL, 2, 1, "AiphoneX", 5000);

INSERT INTO Item VALUES(3, 1, 1, 1, "AiphoneX", 7000);

SELECT \* from Item;

**--Adding new values to Item\_Status**

INSERT INTO Item\_Status VALUES(1, "Sold");

INSERT INTO Item\_Status VALUES(2, "Warehouse");

SELECT \* from Item\_Status;

**--Adding new values to Item\_Type**

INSERT INTO Item\_Type VALUES(1, "Electronic Device");

SELECT \* from Item\_Type;

**--Adding new values to Postomat**

INSERT INTO Postomat VALUES(1, 1, "Work");

SELECT \* from Postomat;

**--Adding new values to Postomat\_Location**

INSERT INTO Postomat\_Location VALUES(1, "Russia", "Moscow", "Nelidiv 25A");

SELECT \* from Postomat\_Location;

**--Update Item\_Type table**

UPDATE Item\_Type SET Type = 'Phone' WHERE id = 1;

SELECT \* from Item\_Type;

**--Update Client table**

UPDATE Client SET Address = 'Nelo 12' WHERE Name = 'Egor' and Surname = 'Mozharov';

SELECT \* from Client;

**--Update Postomat table**

UPDATE Postomat SET status = "Broken" WHERE Postomat\_ID = 2;

SELECT \* from Postomat;

**--Delete Item with id=2**

DELETE FROM Item WHERE item\_id = 2;

SELECT \* from Item

**--Select all Purchase which was delivered and amount of Item in one Purchase more than 1**

SELECT \* FROm Purchase LEFT JOIN Item ON Item.purchase\_id = Purchase.purchase\_id GROUP BY Purchase.purchase\_id HAVING COUNT(Item.Item\_ID) >= 2;

**--CalculatePostomat\_Location total Price for item which was sold**

SELECT SUM(Item.Price) FROM Item LEFT JOIN Item\_Status ON Item\_Status.id = Item.Item\_Status WHERE Item\_Status.Status = "Sold"

**--Calculate current number of working postomats**

SELECT COUNT(Postomat.Postomat\_ID) FROM Postomat WHERE Postomat.Status = "Work"

**--Calculate how much the clients living in Moscow spent in total**

SELECT SUM(Item.Price) FROM Item

FULL JOIN Item\_Status ON Item\_Status.id = Item.Item\_Status

FULL JOIN Purchase ON Purchase.purchase\_id = Item.Purchase\_ID

FULL JOIN Client ON Client.client\_id = Purchase.Client\_ID

WHERE Item\_Status.Status = "Sold" AND Client.City = "Moscow"

**--Select all distinct Purchase\_Status**

SELECT DISTINCT(Purchase\_Status.Status) FROM Purchase\_Status

**--Calculate the number of Client who made at least 1 purchase**

SELECT COUNT(Client.Client\_ID) FROM Client

CROSS JOIN Purchase ON Purchase.Client\_ID = Client.Client\_ID

# Conclusion

During the execution of the project work, our team managed to fulfill all the tasks assigned to us, namely: Normalization, Technical requirements, Preliminary database schema, Functional dependencies, create database and make queries for it. Moreover, teamwork was worked out, the division of responsibilities was improved, practical skills were improved when writing queries, as well as the types of normalization were studied in more detail, as well as the ways and reasons for its application.

Source code: [Github](https://github.com/EgorMozharov/HW_DB_Group_Project)