**ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ ЗВ’ЯЗКУ ІМ. О.С. ПОПОВА**

Навчально-науковий інститут інфокомунікацій та програмної інженерії

Кафедра інформаціонних технологій

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

З дисципліни «Бази Даних»

на тему **РОЗРОБКА ІНФОРМАЦІОННОЇ СИСТЕМИ УПРАВЛІННЯ МЕРЕЖЕЮ МАГАЗИНІВ З ПРОДАЖУ ПОБУТОВОЇ ТЕХНІКИ**

Виконав: студент 3 курсу, групи ПІ-3.01

Караянов І. О.

Керівник проф. Малахов Е. В.

Національна шкала \_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_ Оцінка: ECTS\_\_\_\_\_\_\_

Одеса – 2019

**CONTEXT**

[INTRODUCTION 3](#_Toc10644498)

[1 FORMULATION OF THE PROBLEM 5](#_Toc10644499)

[2 INFORMATION SYSTEM DESIGN 6](#_Toc10644500)

[3 DOMAIN INFORMATION MODELING 7](#_Toc10644501)

[3.1 Tables 7](#_Toc10644502)

[3.2 Normalization 10](#_Toc10644503)

[3.3 ER Diagram 13](#_Toc10644504)

[4 SOFTWARE SELECTION 15](#_Toc10644505)

[5 APPLICATION MODEL 17](#_Toc10644506)

[5.1 Hierarchy of pages . 17](#_Toc10644507)

[5.2 Hierarchy of components 17](#_Toc10644508)

[6 DATABASE CREATION 19](#_Toc10644509)

[7 DATABASE QUERIES FOR SOLVING ASSIGNED TASKS 20](#_Toc10644510)

[8 SOFTWARE IMPLEMENTATION OF THE CLIENT APPLICATION 22](#_Toc10644511)

[8.1 Connection with Database 22](#_Toc10644512)

[8.2 Connection from client to server 23](#_Toc10644513)

[8.3 Retrieving data from Database into client 24](#_Toc10644514)

[8.4 Mutation of database data 26](#_Toc10644515)

[9 USER GUIDE WITH TEXT EXAMPLE AND ILLUSTRATION 29](#_Toc10644516)

[LIST OF SOURCES USED 39](#_Toc10644517)

[Appendix A ER DIAGRAM 40](#_Toc10644518)

[Appendix B CREATING TABLES 41](#_Toc10644519)

[Appendix C CREATING VIEWS, ROLES AND THEIR PRIVILEGES 47](#_Toc10644521)

[Appenix D QUERIES SOLVING TASKS 51](#_Toc10644523)

# INTRODUCTION

Every application is the digital interpretation or automatization of the some system and tasks that thing people do by yourself. Purpose of any application is to make usual thing faster or check and count every process hence automatize of computerize system

Current application solved the following problem. The problem of automatization of selling products.

Because I has experience in this sphere.

Analyzing the subject domain of the network of household shops the main purpose can be easily defined. This is provide people with these goods. The process of sale are starting from producers. They make products and supply them to the warehouse (also called as primary reserve) providing order with a bill. The bill is list of ordered products and their price. Then another truck supply products from the warehouse to shop documented by the shop delivery bill. And after retrieving products on particular shop sellers provide products to clients. Each sale must be documented in view of check for wholeness of the system. The same situation for bills and shop delivery bills.

If you look through this algorithm then you can see all data are related. But it is not closed system. The Producer always can supply new products, or old products but with different prices.

Also the look of domain is that:

1. The net is located in one city
2. It has the only one warehouse
3. It has many shops
4. In each shop works several sellers and a manager
5. Managers regulate supply on Primary reserve and shop reserve

To develop this application working with information system of the network of household shops the student must:

1. Define tasks on the subject domain of the information system
2. Make a schema of the information system (shattering it by layers)
3. Information modeling of the subject domain (developing the database)
4. To choose software the application will be developed
5. Make the application model (class diagram / pages)
6. Create database: tables, views, functions, triggers etc.
7. Create queries to solve standalone tasks
8. Develop program realization of client application
9. Create user interface guide with illustrations

Developing application the information system must be considered. The application will interpret which can work with information system for a network of household appliances stores.

The application must differ roles of service (managers, sellers etc.) in purpose of access and protection and task distribution.

Next, here will be described structuring of database model and recreating all the essenses we need with postrgesql, creating webserver using node.js, postgrephile, express, and graphql and creating frontend part or client using JS and React.

# 1 FORMULATION OF THE PROBLEM

Defining tasks for users of the information system describing in table 1.1.

Table 1.1 – User tasks

|  |  |
| --- | --- |
| User | Tasks |
| 1. Seller | 1.1 Sell products |
| 1.2 View all products on his shop |
| 1.3 Print check |
| 1.4 Undo the last sale |
| 1.5 View the sale history on his shop |
| 1.6 Login |
| 1. Manager | 2.1 View all products on his shop |
| 2.2View all products on wirehouse |
| 2.3 Order products from the warehouse to shop |
| 2.4 View most sold products based on all shops |
| 2.5 Order most sold from manufacturer to warehouse |
| 2.6 Login |

# 2 INFORMATION SYSTEM DESIGN

The application will be developed using 3-layer architecture, because the client ask queries not directly.

The client started on post 3000 and interpret only interface. The client wrapped in Apollo provider where determined link to the webserver. Client will provide queries to webserver.

Webserver will listen on another port for queries and translate in to SQL and ask Database and respond with result to client in JSON .

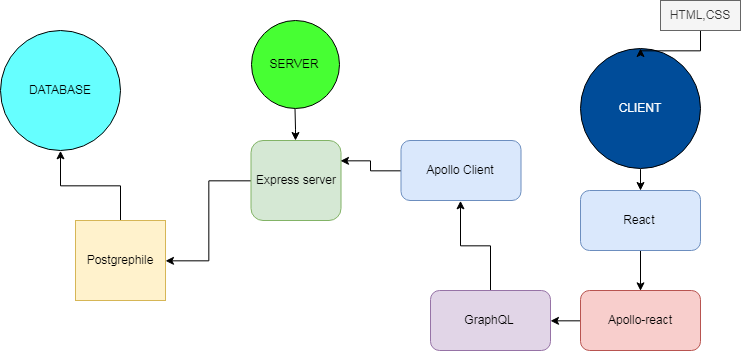


Figure 2.1 – Three layer architecture

A Client can be either hardware or software. It can be the source as well as a destination of data to or from a Database. Usually the client is a human operated workstation, data source (Sensor) or another database.

A Database is software that provides data storage and operates on a hardware Server. A database is usually controlled by a Data Base Management System (DBMS) software. A Database stores, provides access to the stored data and allows for data manipulation to meet client requests.

A Server is a Physical computer where the Database resides. It runs the DBMS and provides communication connections between the Client and the database.

# 3 DOMAIN INFORMATION MODELING

## 3.1 Tables

Modeling the subject domain (SB) it is convenient to start from defining views of objects. So the essence of these objects will be considered. Essence is the set of the similar objects. Every essence possess some number of main attributes defining itself

Essences in our subject domain follows from tasks in table 1.1. They will be represented as tables below.

The producer essence represents an entity that create and supply products to the warehouse. The ‘Producers table’ represented in table 3.1. Considering ‘producers table ‘ attributes the relation is reduced to the normal Boyce Codd shape (NBCS). Each attribute directly depends from potential key, there is no transitivity. The same form determined for each table 3.1-3.11.

Table 3.1 - Producers

|  |  |  |  |
| --- | --- | --- | --- |
| **Producers** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| id | Producer ID | Serial | Primary key |
| mark | Producer or company name | Varchar | Unique, not null |
| email | Producer Email to contact | Varchar | Unique, not null |
| Mrh\_name | Merchandiser (Representor name) | Varchar | Unique, not null |
| phone | Contact number | Varchar | Unique- |

‘Bill table’ represented in table 3.2.

Table 3.2-Bill

|  |  |  |  |
| --- | --- | --- | --- |
| **Bill** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Id | Supply Bill ID | Serial | Primary key |
| Billdate | Date of creating | Timestamp |  |
| ProducerId | Producer ID | Int | Foreign key |

Table 3.3 - BillItem

|  |  |  |  |
| --- | --- | --- | --- |
| **BillItem** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| BillId | Bill ID references  Table 3.1 | Int | Foreign key ,  primary key |
| Price | Price from producer | Numeric | Not null |
| ProductId | Product ID references  Table 3.4 | Int | Foreign key,  primary key |
| Quantity | Supplied quantity | Int | Not null |

Table 3.4 - Products

|  |  |  |  |
| --- | --- | --- | --- |
| **Products** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Id | Product id | Serial | Primary key |
| Model | Product name | Vachar | Not null |
| Category | Category | Tech\_category | Not null |
| Description | Product optional info | Vachar |  |
| Color | Color | Vachar | Not null |
| volume | Size(large small…) | Vachar |  |
| Turn\_in\_sec | Optional for washer | Int |  |

Table 3.5 - Shop

|  |  |  |  |
| --- | --- | --- | --- |
| **Shop** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Id | Shop ID | Serial | Primary key |
| Title | Title of shop | Varchar | Not null |
| City | Name of the city | Varchar | Not null |
| Street number | Street number | Int | Not null |
| Mngr\_name | Manager name | Varchar | Not null |
| Mngr\_lastname | Manager lastname | Varchar | Not null |

Table 3.6 - ShopDeliveryBill

|  |  |  |  |
| --- | --- | --- | --- |
| **ShopDeliveryBill** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Id | Id of delivery Bill on particular shop | Serial | Primary key |
| Data | The date | Timestamp | Not null |
| Shop\_id | Shop ID | Int | Not null |

Table 3.7 - PrimaryReserve

|  |  |  |  |
| --- | --- | --- | --- |
| **PrimaryReserve** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Destribution\_id | ‘shopdeliverybill’id references table 3.6 | Int | Foreign key, primary key, Not null |
| Product\_id | Product ID reference ‘product table’ table 3.4 | Int | Foreign key,  primary key Not null |
| Quantity | Quantity | Int | Not null |

Table 3.7 - Seller

|  |  |  |  |
| --- | --- | --- | --- |
| **Seller** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| ID | Seller ID | Int | Primary key |
| Name | Name | Varchar | Not null |
| Last\_Name | Name | Varchar | Not null |
| Phone | Contact number | Varchar | - |
| Shop\_id | Shop id references table 3.5 | Int | Foreign key,not null |

Table 3.8 - BuyCheck

|  |  |  |  |
| --- | --- | --- | --- |
| **BuyCheck** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Id | Check ID | Serial | Primary key |
| sellerId | Seller ID references ‘seller table’ 3.7 | Int | Foreign key |
| data | Date/time of sale | Timestamp | Not null |

Table 3.9 - CheckContent

|  |  |  |  |
| --- | --- | --- | --- |
| **CheckContent** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Check\_id | Check ID references  Table 3.1 | Int | Foreign key, primary key |
| Product\_id | Product ID references  Table 3.4 | Int | Foreign key, primary key |
| Count | Count of single position in the check | Int | Int,not null |

Table 3.10 - SellerPassword

|  |  |  |  |
| --- | --- | --- | --- |
| **SellerPassword** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Login | Login | Varchar | Primary key |
| Password | Password | Varchar | Not null |
| Seller\_id | Seller ID references table 3.7 | Int | Primary key,  Foreign key |

Table 3.11- ShopPassword

|  |  |  |  |
| --- | --- | --- | --- |
| **ShopPassword** | | | |
| **Attribute** | **Description** | **Type** | **Constraint** |
| Login | Login | Varchar | Primary key |
| Password | Password | Varchar | Not null |
| Shop\_id | Shop ID references table 3.5 | Int | Primary key,  Foreign key |

## 3.2 Normalization

For correct work with database the anomaly need to be prevented. Hence, the Relation must be normalized. The tables above have already been normalized. All Normalizations are represented on figures 3.1-3.3.

Figure 3.1 represent M-N relation between ‘Producer’ and ‘Products’ that conclude to appearing of new third mid table ‘BillList’ that was normalized to Boyce Codd Form due to several kay with different attributes dependence transitivity. The figures 3.2 and 3.3 will have same principle of normalization.

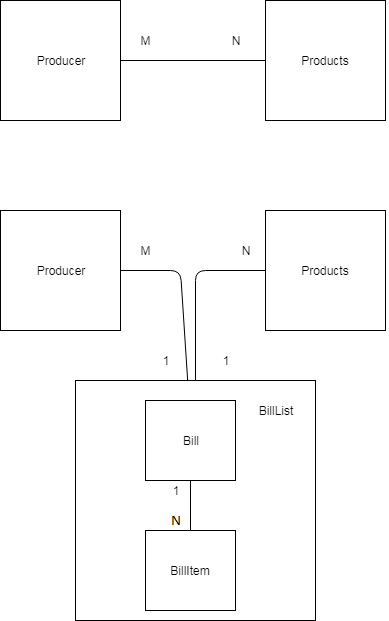


Figure 3.1 – Normalization of M-N- relation 1

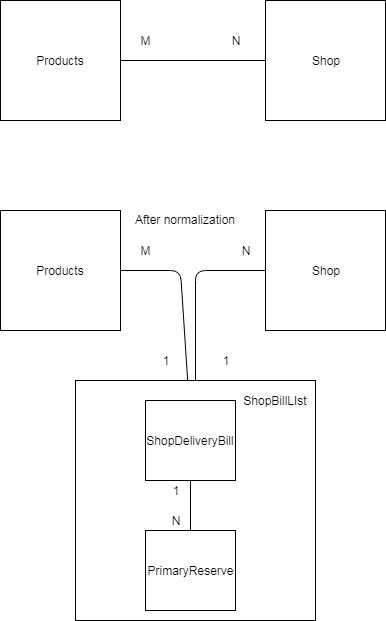


Figure 3.2 – Normalization of M-N- relation 2

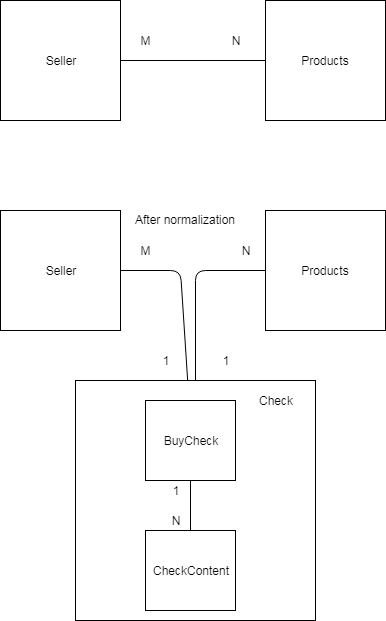


Figure 3.3 – Normalization of M-N- relation 3

Another types of relations as ‘1 to N’ or ‘1 to 1’ are represented on ER diagram in Appendix A.

## 3.3 ER Diagram

The ER or (Entity Relational Model) is a high-level conceptual data model diagram. Entity-Relation model is based on the notion of real-world entities and the relationship between them.

ER modeling helps you to analyze data requirements systematically to produce a well-designed database.

The ER diagram of the relationship entities from table 3.1-3.8 represented in Appendix A.

# 4 SOFTWARE SELECTION

PostgreSQL. PostgreSQL is a general purpose and object-relational database management system, the most advanced open source database system. Will be used in this web application application due to large community, ambitious number of framework and node packages that help easy work and convenient with postgres.

PgAdmin. PgAdmin is the most popular and feature rich Open Source administration and development platform for PostgreSQL. It is used to create database and its entries (tables, view, types, functions etc.).

Node.js.Node.js is an open-source and cross-platform JavaScript runtime environment. Will be used to run webserver and any ‘.js’ file from CLI.

NPM. NPM is node package manager that is provided with Node.js. Help you to install and manage tools, features for web development. Used for running JS files in CLI.

Express.js. Express.js is Node library that set up server and manage REST API Express will set up server through last the database and client will connect.

Postgrephile. The postgrephile is the JS library that make out of your Database schema into GraphQL schema automatically. Using express REST API, it connects to database and provide it to the server as GraphQL scheme.

GraphQL. GraphQL is a query language that gives clients the power to ask for exactly what they need. Using JSON view of data with nesting. Nesting data represented in foreign key principle.

GraphiQL. The browser tool that are ran on the server of application where you can easy view and test all of your GraphQL queries and mutations.

Apollo. Apollo is the node module that provide GraphQL scheme to react.

JavaScript. JavaScript is high-level web development language. The most popular language for web development. It is used to create Webserver and Client to handle client actions and React.

React. React is component’s based JavaScript framework. Use to create multiple pages user interface.

React Context API. ‘React Context API’ is react local store. It used for storing cache needed on different orders of react component.

HTML. Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. HTML define the client and react component’s structure.

CSS and LESS. CSS is Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. Less is the style library that enable you to write nested structures that then converted into CSS.

# APPLICATION MODEL

## 5.1 Hierarchy of pages

The hierarchy of pages is represented on figure 5.1.

Figure 5.1 – Hierarchy of page

The figure 5.1 shows page routes of the application. For unauthorized user any route will redirect to ‘/Auth’.

## 5.2 Hierarchy of components

JavaScript is not Object oriented language .Class a not used but react framework use ‘component oriented model’. Main Components represented in figure 5.2 the diagram shows Hierarchy of components in applications. Greens are components that was extended, and yellow ones are extended from parent or pattern ‘react’ component.

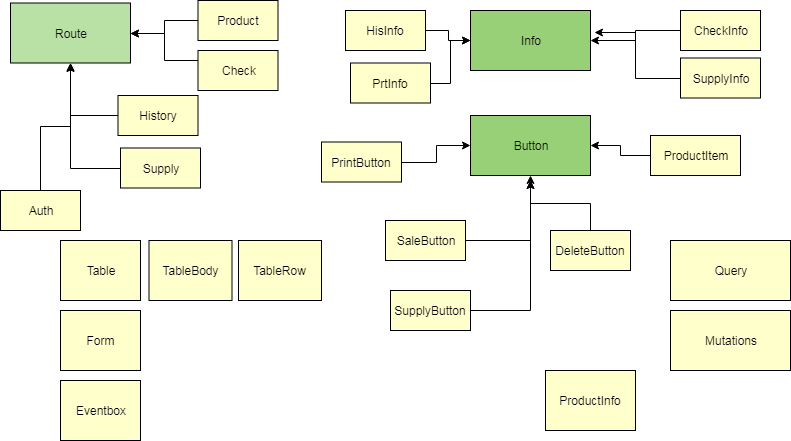


Figure 5.2 – Diagram of main components

# 6 DATABASE CREATION

The database has tables, custom type, primary keys, foreign keys, not null, default constraint for serial properties on relations, on delete cascade for table Bill, table ‘buycheck’, table ‘primary reserve’. They all represented in Appendix B. Queries for creating of views, roles and grants in Appendix C.

# 7 DATABASE QUERIES FOR SOLVING ASSIGNED TASKS

For solving task 1.1. In table 1.1 was create trigger ‘SaleBlock on checkcontent’, ‘trigger function ‘SaleBlockFun’ and function ‘Sale(sellerid int, int[] products, int[] prcount)’. Where the first argument is seller ID, the second is array of products id that sold. In addition, the third is the array of the quantity in order of products IDs.

Example:

Select \* from Sale (1, [3, 16], [1, 1])

For solving task, 1.2 in table 1.1, the ‘SELECT’ query to view ‘productsview’ was implemented:

Select \* from productsview

For solving task 1.3 in table 1.1, the return type of function ‘Sale’ is table that represent check need info. This is used to format and print check in React.

For solving task 1.4 in table 1.1 delete query to `buycheck was made.

Delete from buycheck where id = N

/\*N is a number of check should be deleted.\*/

For solving task, 1.5 in table 1.1 was made two queries:

Select id from buycheck where id = N;

/ Result of below will be used as ‘N’ in second one\*/

Select \* from checkview where checkview\_id = N

For solving task 2.1 in table 1.1, the ‘select’ query to view ‘productsview’ was implemented:

SELECT \* FROM productsview

For solving task 2.2 in table 1.1, there was created function ‘get\_wirehouse\_by\_category (varchar cat)’ was created, where argument represent the particular category.

Example:

Select \* from get\_wirehouse\_by\_category (‘washer’)

For solving task 2.3 in table 1.1 were created ‘trigger SupplyBlock before insert on primaryreserve’, SupplyBlockFun’ trigger function and ‘Supply(shopid int, int[] products, int[] prcount)’ function. Arguments:

1. Shop to supply;
2. Array of products;
3. Array of quantity to supply.

Example:

Select \* from supply (1, [2, 11], [3, 3])

For solving task 2.4 in table 1.1, there was created ‘getDeliveryList()’ function.

Example:

Select \* from getDeliveryList()

For solving task 2.5 in table 1.1, there was created ‘postTopBill()’ function, and ‘random\_between’ that generates new prices from producers.

Example

Select postTopBill()

For solving task 1.1 and 2.6 there was created tables of passwords, two roles, and following functions: ‘login’, ‘checkauthmanager’, ‘checkauthseller’. To Login the following query must be implemented:

Select \* from login('login’, ‘password')

All above functions, stored procedures and triggers represented in appendix D.

# 8 SOFTWARE IMPLEMENTATION OF THE CLIENT APPLICATION

## 8.1 Connection with Database

To connect to database was created ‘server’ folder Where server files were placed. To connect to DB ‘app.js', and ‘.env’ files were set up.

//app.js file

// import from package.json

const express = require("express");

const { postgraphile } = require("postgraphile");

const ConnectionFilterPlugin = require("postgraphile-plugin-connection-filter");

const graphqlHTTP = require('express-graphql');

const cors=require('cors')

// init server

var app = express();

//import .env

require('dotenv').config()

// connect to datadase dblab2 to public scheme throuph postgrephile

// emitting data in JSON to client

app.use(

postgraphile( process.env.DATABASE\_URL, null, {

appendPlugins: [ConnectionFilterPlugin],

graphiql: true,// grapgQL tool

enableCors: true// enable requests from client to server for the graphQL

})

);

app.use(cors()) // enable requests from client to server

// url to graphQL manager.

app.use('/graphql', graphqlHTTP({

graphiql: true

}));

//start the server

app.listen(process.env.PORT ,()=>{

console.log( 'link on outer', process.env.DATABASE\_URL)

});

Listing 8.1 – Connect to DB

The environment file on listing 8.1 stores database URL, role, role’s password and the server post number

//.env files

DATABASE\_URL="postgres:manager:1111@localhost:5432/dblab2"

USER=manager

PASS=1111

PORT=5000

# DATABASE\_URL="postgres:seller:1234@localhost:5432/dblab2"

# USER=seller

# PASS=1234

# PORT=5000

Listing 8.1 – Environment file

## 8.2 Connection from client to server

Go to index.js file on the React application and wrap your ‘App’ component into Apollo Provider that connected to webserver. Apollo client contain link on webserver on route where graphQL scheme is defined

//imports

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';// css style

import App from './App';// App component

import \* as serviceWorker from './serviceWorker';

import ApolloClient from 'apollo-boost' // class that connect to graphQL

import {ApolloProvider} from 'react-apollo'// Provide graphQL scheme to the application

const client = new ApolloClient({

uri : 'http://localhost:5000/graphql'// server link to connect to graphQL schema

})

//renders app

ReactDOM.render(

<ApolloProvider client ={client}>

<App />

</ApolloProvider>,

document.getElementById('root')

);

Listing 8.2 – Connection from client to server

## 8.3 Retrieving data from Database into client

The graphQL scheme of application are reduced from Postgres one due to postgrephile.

Different is then every name is camel-cased. The result of GraphQL query will be in JSON.

Example of this query in the listing 8.2.

//queries.js

export const getWirehouseByCategory = gql`

query ($category:String!)

{

getWirehouseByCategory(cat: $category) {

edges {

node {

prodId

model

mark

price

avalCount

}

}

}

}`

Listing 8.2 – GraphQL query

In client folder, the query.js file was set up. The result of reusable query represented on figure 8.1, so where ‘$category’ there parameter of function should stand.

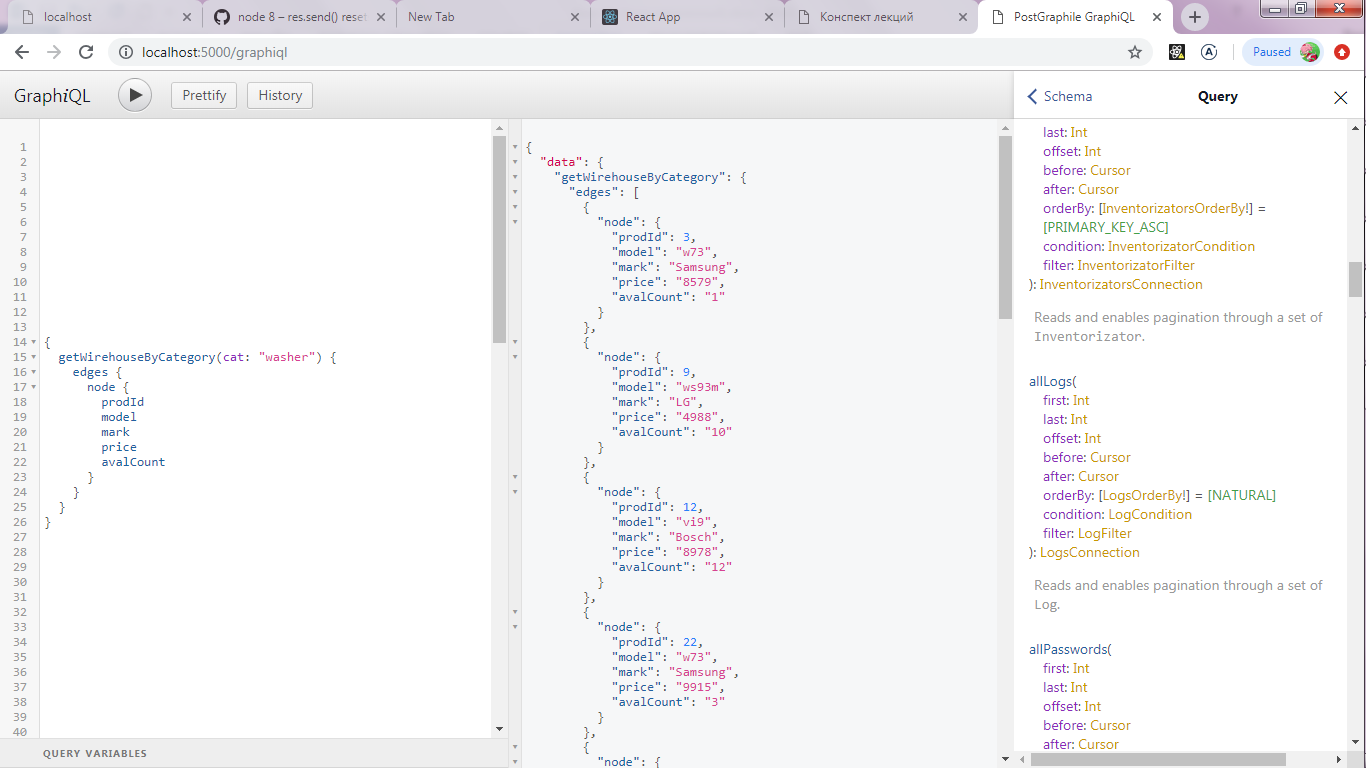


Figure 8.1 – GraphiQL query

The ‘queries.js’ file is a set of GraphQL’ queries. The one elbow similar to the next PostgreSQL query:

Select prod\_id, model, mark, price, aval\_count from ‘get\_wirehouse\_by\_category(category)’

/\* where category any varchar.\*/

To bring result of GraphQL query into ‘client-side’. The component context must be wrapped into Query component. The example is represented in listing 8.3

The query component has two attributes:

1. The reference to the query
2. The variables

//CategoryReserve.js

import React from 'react'

//Query components

import { Query} from 'react-apollo'

//query

import {getWirehouseByCategory} from '../../queries'

import ProductList from './ProductList'

//CategoryReserve component

const CategoryReserve = ({category}) =>

<Query query={getWirehouseByCategory} variables = {{category}} >

{

({loading,error,data} )=>{

if (loading) return "Loading...";

if (error) return <p>

No access

</p>;

else return (

<React.Fragment>

<h4>{category}</h4>

<ProductList edges = {data.getWirehouseByCategory.edges} />

</React.Fragment>

)

}

}

</Query>

export default (CategoryReserve)

Listing 8.3 – Query component in React

## 8.4 Mutation of database data

To make changes on database the GraphQL mutations should be used.

Mutations in GraphQL scheme define any query where update, delete, insert keywords, and functions with volatile status.

Example of mutations:

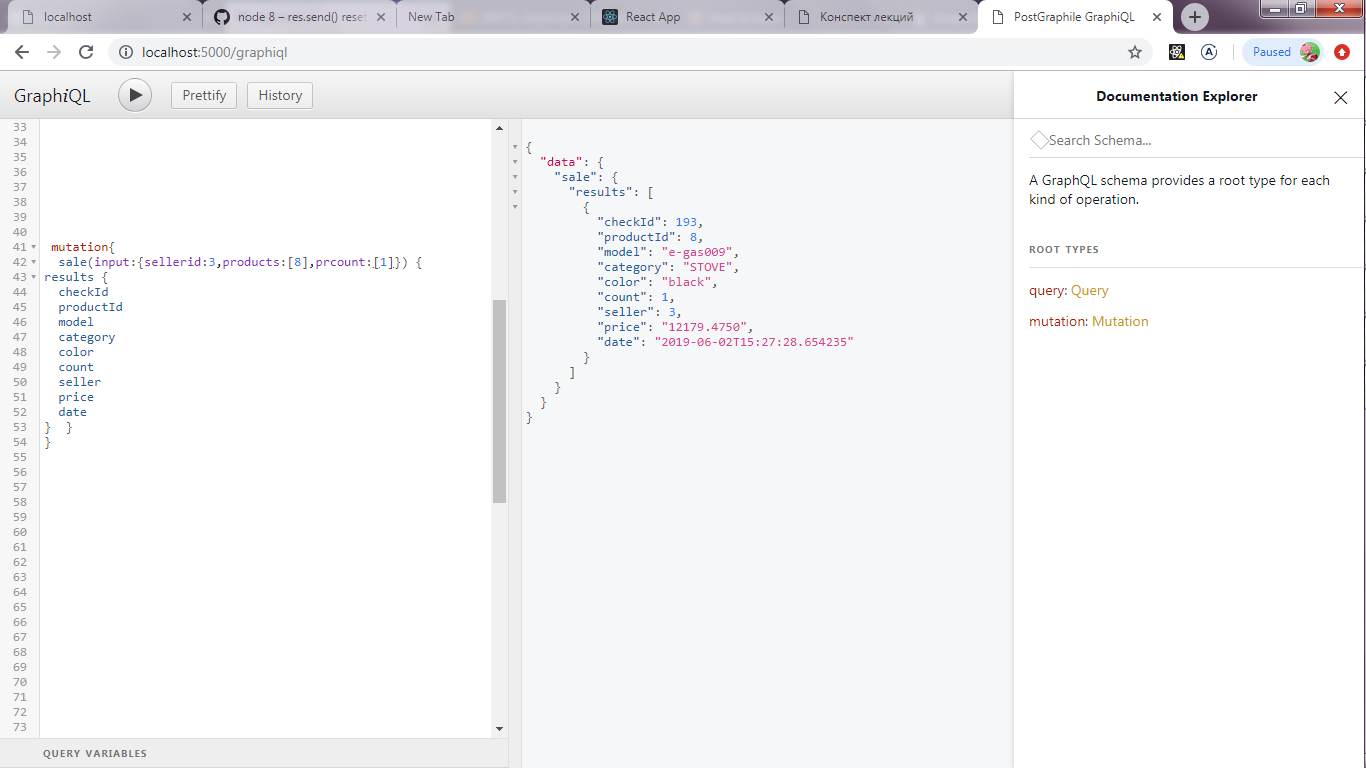


Figure 8.2 – GraphQL mutation

The mutation elbow similar to

Select \* from Sale(3,[8],[1])

In application the mutations are set on client in mutatuion.js file.

Example:

export const ADD\_SALE = gql`

mutation AddSale($seller: Int!, $products:[Int!], $prcount:[Int!]){

sale(input:{sellerid:$seller,products:$products,prcount:$prcount}) {

results {

checkId

productId

model

category

color

count

seller

price

date

} }

}

`;

Listing 8.2 – GraphQL mutation

To provide ability on client, ‘Mutation’component should be exported from ‘react-apollo’.

import React , {useState} from 'react'

import {Mutation} from 'react-apollo'

import {ADD\_SALE} from '../../mutations'

import {MyContext} from '../../Provider'

import {ReduceFromArrOfObj} from '../../library'

function SaleButton({mutation,check,}) {

let arrIDs = ReduceFromArrOfObj(check,'id');

let arrCounts = ReduceFromArrOfObj(check,'count');

return (

<MyContext.Consumer>

{

({userId,shop,updateMsg,setCheck,setLastCheck})=>

<Mutation mutation = {ADD\_SALE}>

{

(mutate, {data}) =>{

console.log('mutate',mutate)

return( <button type='submit' className = 'btn blue ' onClick = {()=>{

updateMsg(`Sold in shop ${shop} `)

mutate({variables:{seller: parseInt(userId),products:arrIDs,prcount:arrCounts}}).then(({data})=>{

console.log('SALEDATA',data.sale.results)

setLastCheck( data.sale.results)

setCheck([])

},error=>{

updateMsg('ERROR! product miscount!')

console.log(error)

})

}}>SALE</button>)

}

}

</Mutation>

}

</MyContext.Consumer>

)

}

export default SaleButton

Listing 8.4 – Mutation component in React

The arguments for mutative function was set on the call.

# 9 USER GUIDE WITH TEXT EXAMPLE AND ILLUSTRATION

The manager login screen on figure 7.1, where:

1. Determine role that can login.
2. Login input field
3. Password input field
4. Submit button that login use with with particular login, if it is correct for current role

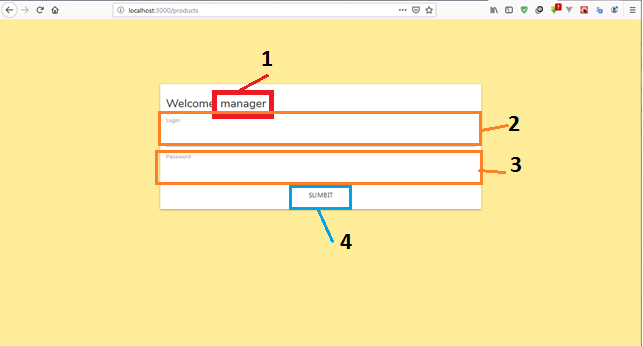


Figure 7.1 –Manager login

The seller login screen on figure 7.2, where the instruction is the same as for figure 7.1.

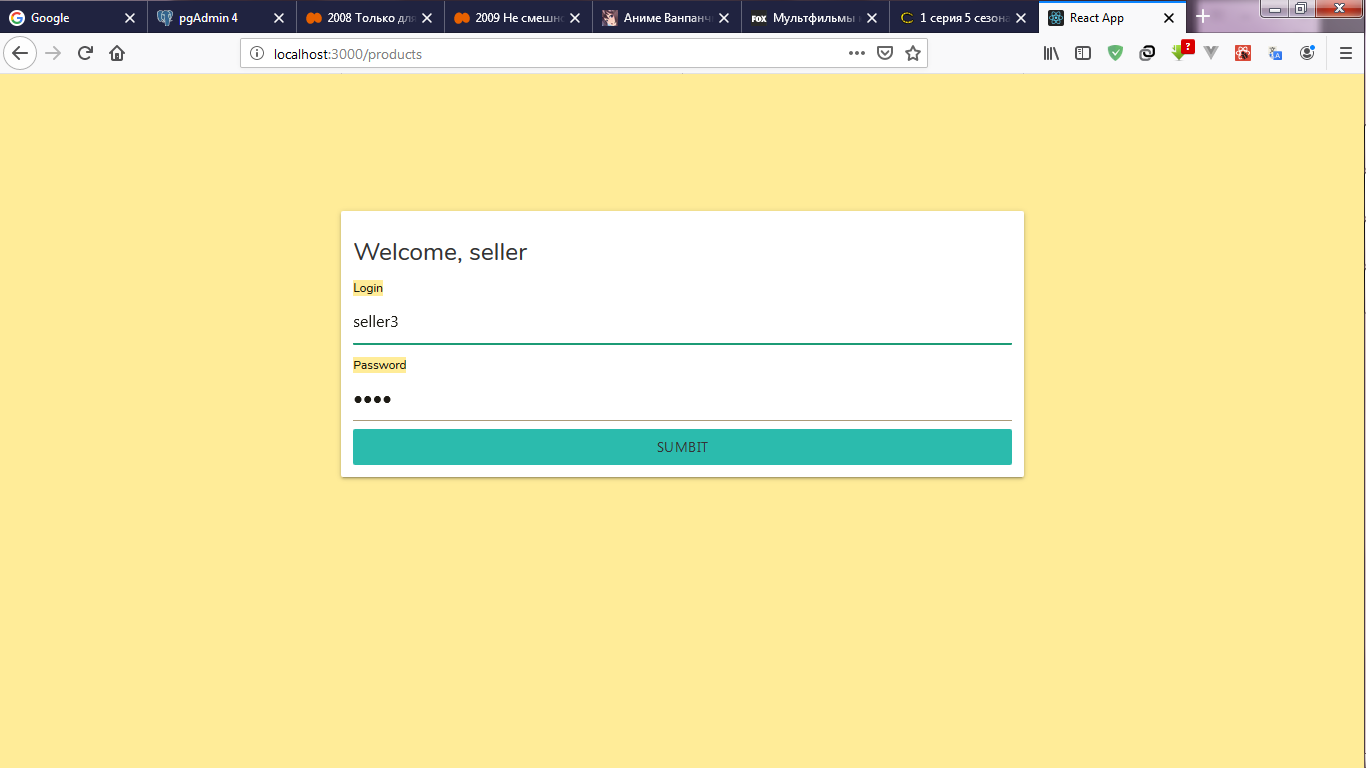


Figure 7.2 –Seller login

Products page is represented on figure 7.3, where:

1. Button, that ask you to logout
2. Products that available in current shop, current shop is based on logined seller .
3. Navigation link
4. Navbar
5. Last event based on user action
6. Info block
7. Product item as button that select the product, shows information about it and allows actions in it
8. ’Add selected atem to check
9. Link to ‘Check’ page in figure 7.4
10. Number input determine how many selected products add to check
11. Info about selected product
12. Priduct’s quantity of selcted products on the shop
13. Seller info (role, name, shop)

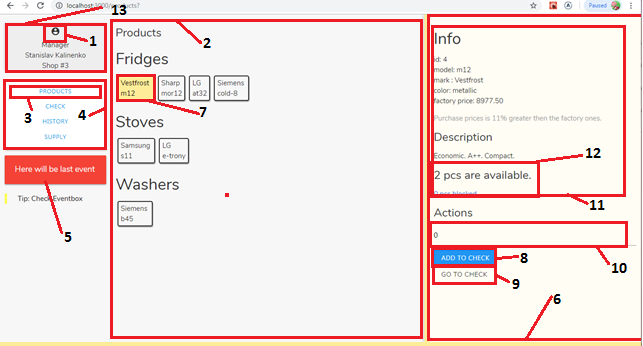


Figure 7.3 –Products page

Check page before ‘sale’ represented on figure 7.4, where

1. Check block
2. Check view with items will be sold as table and actions
3. Clear button, that delete all position from check view
4. Table columns
5. Check position, for each product in check
6. Delete position button
7. ‘Sale’ button (on click sale all products in check)
8. Link to ‘products’ page on figure 7.3
9. Position total sum
10. Total sum for all positions in check

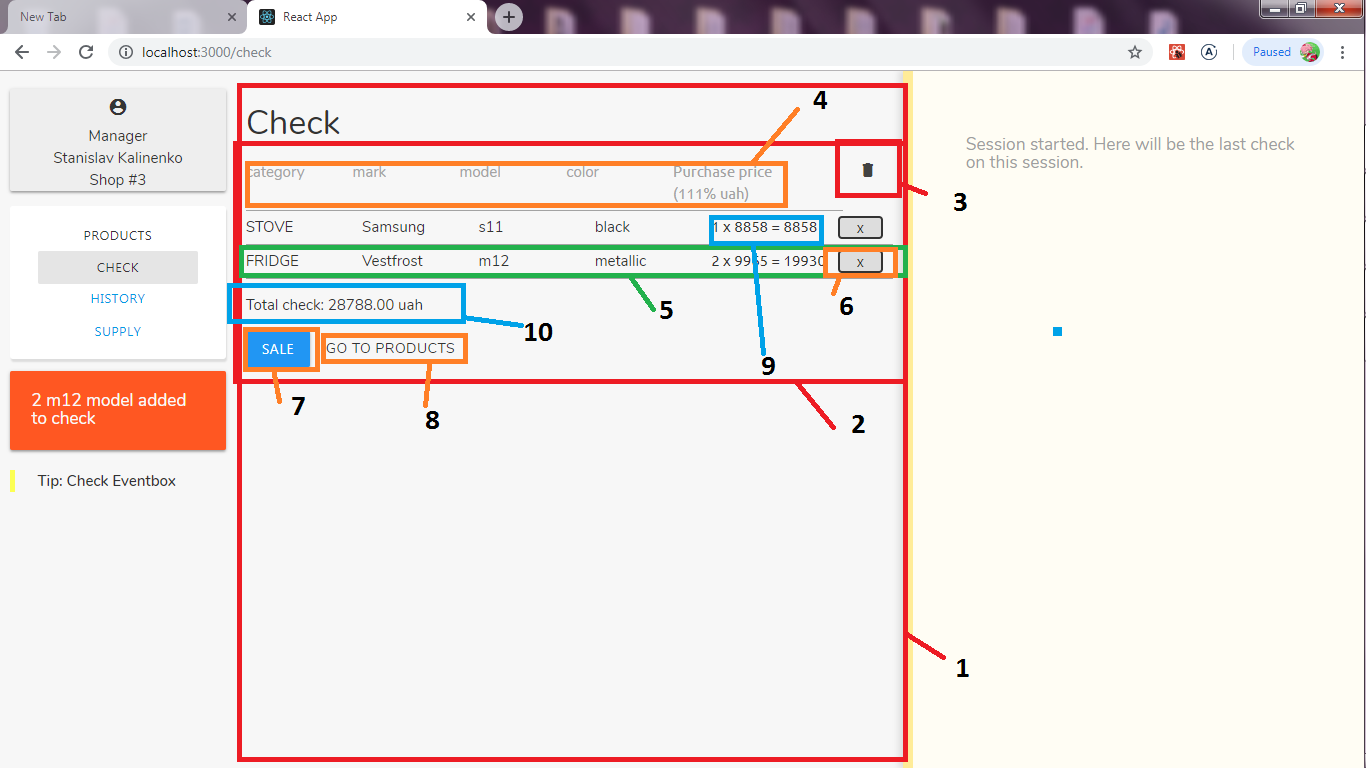


Figure 7.4 –Check page before ‘sale’

The check after ‘sale’ event occurs on ‘sale’ button click the check view will be cleaned and next screen view will be seen as on figure 7.5.

1. Last check block
2. Check that made after ‘sale’ button clicked
3. ‘Undo’ button that cancel this check
4. ‘Print’ button, prints this check

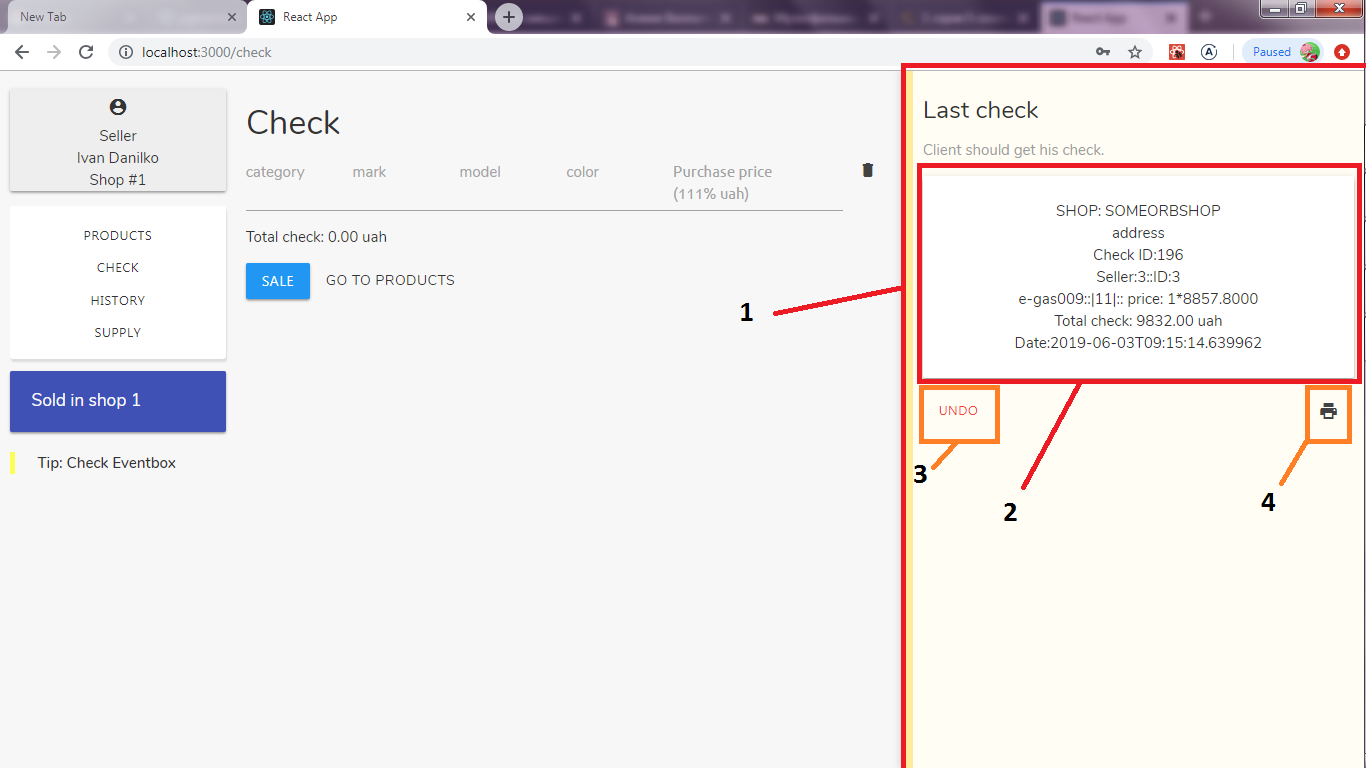


Figure 7.5 –Check page after ‘sale’

The history page represented on the figure 7.6, where:

1. History block where by check id ut can be watched
2. Number input field for seaching check by id
3. Search result block

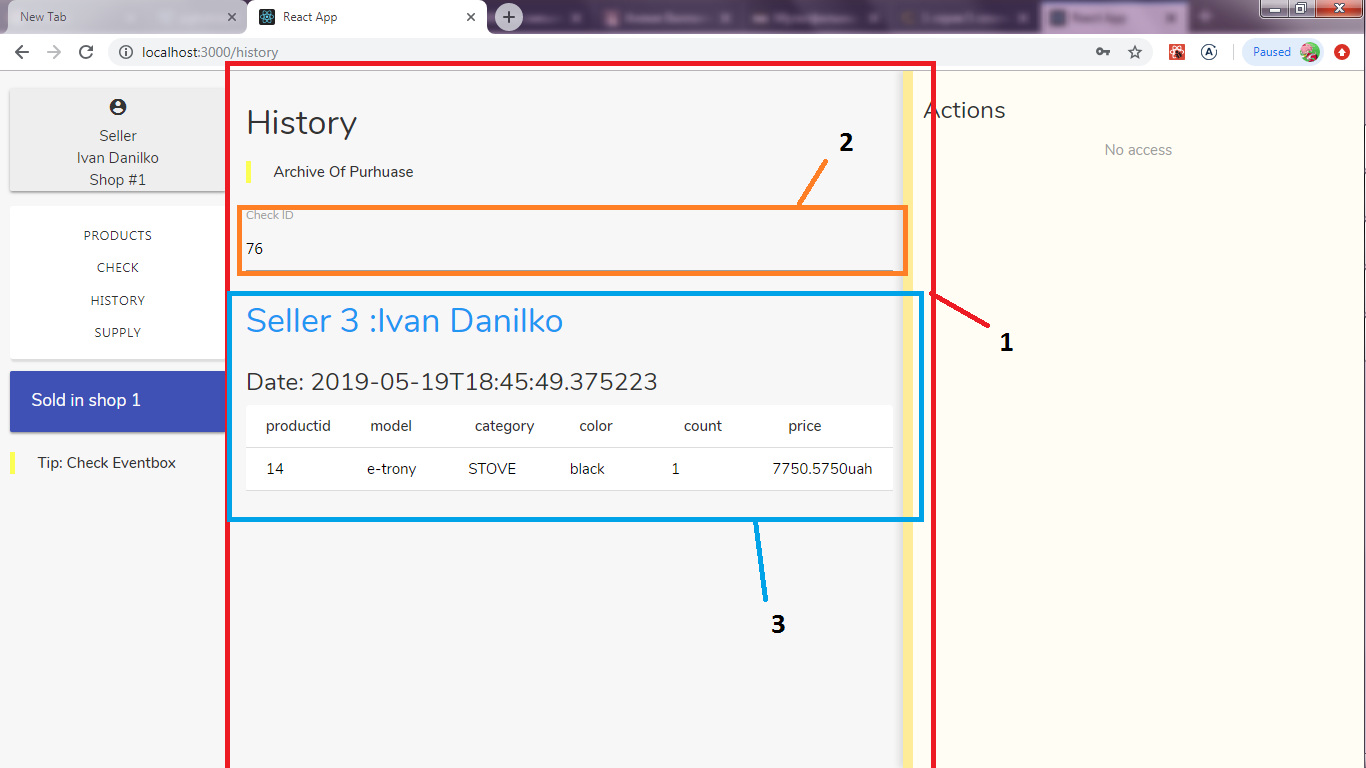


Figure 7.6 –History page

Seller supply page is represented on figure 7.8. There is no access to any action for seller.

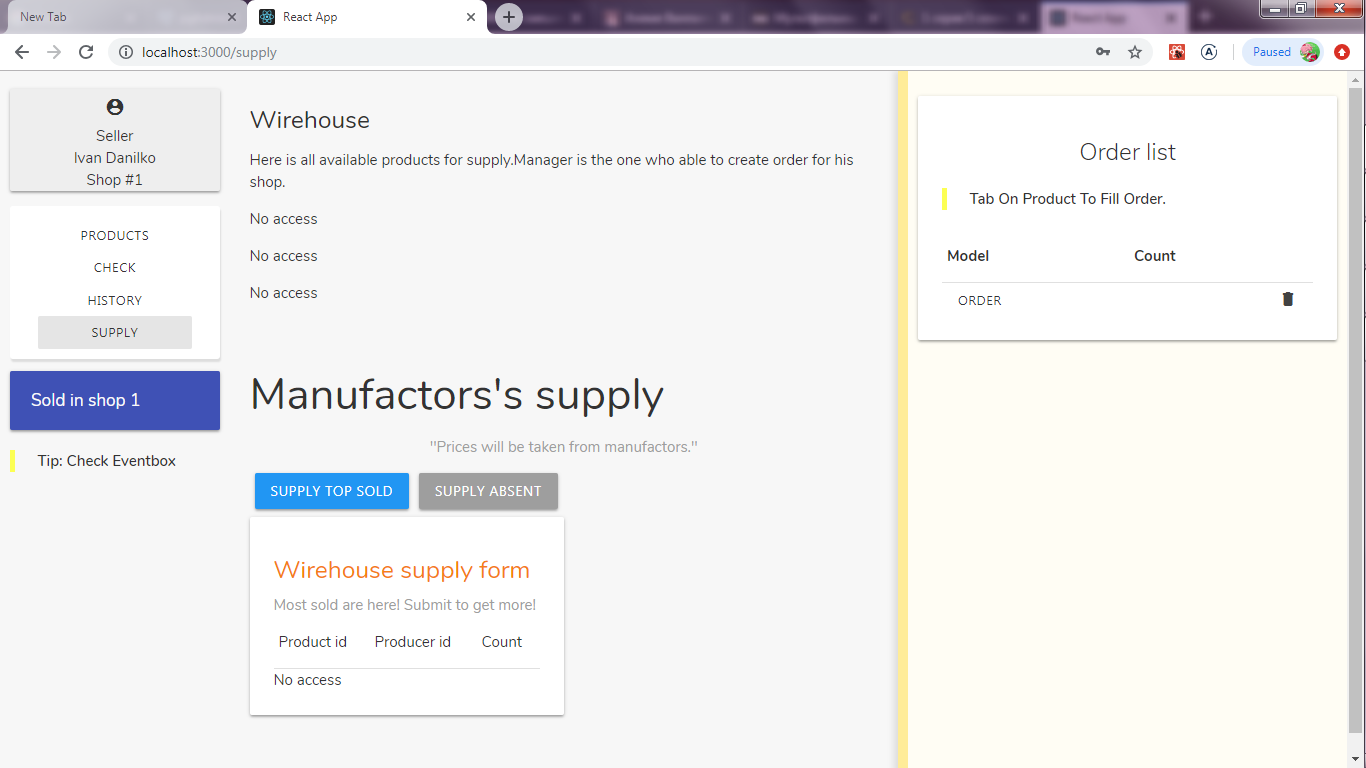


Figure 7.7 – Seller supply page

Manager supply page is represented on figures 7.8-7.9, where

1. Products supply list block ordered by category
2. Info-action block
3. Info aboult selected product item
4. ‘Add to the order list selected item’ form (products will be supplied to the shop there logined user works)
5. Selected model
6. Avaliable count on the warehouse
7. Number input, define how many items order to the shop
8. ‘Add’ button. Adds selected item to Order list
9. Order list
10. Position on the order list
11. ‘Order’ button that submit order to the shop defined on logined user
12. ‘Clear’ button that removes all postions from the order list
13. Separated category of products
14. Product item
15. Selected product item
16. Manufactor supply block, here is a suppliment on the warehouse
17. Button, that responsible for for suppliment that products were sold on last week
18. List of products were were sold on last week

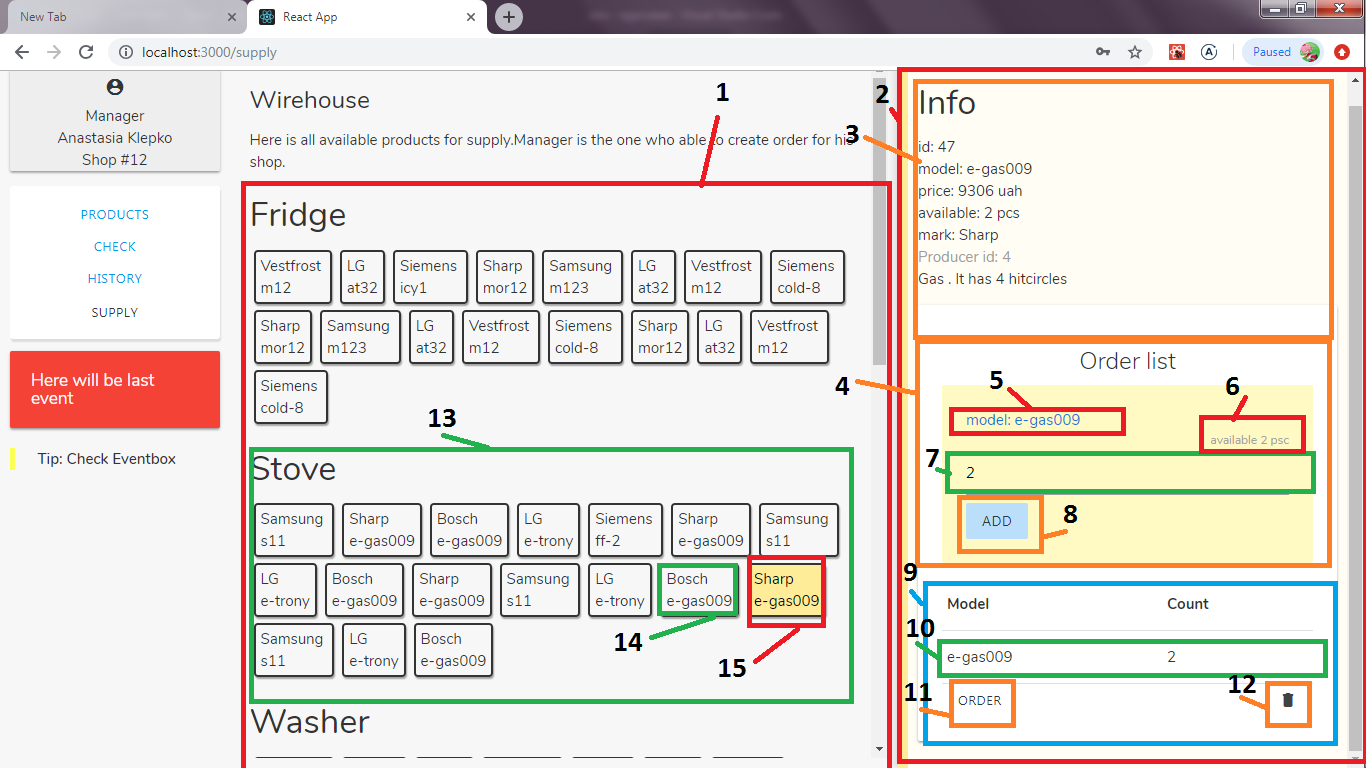


Figure 7.8 – Manager Supply page ()

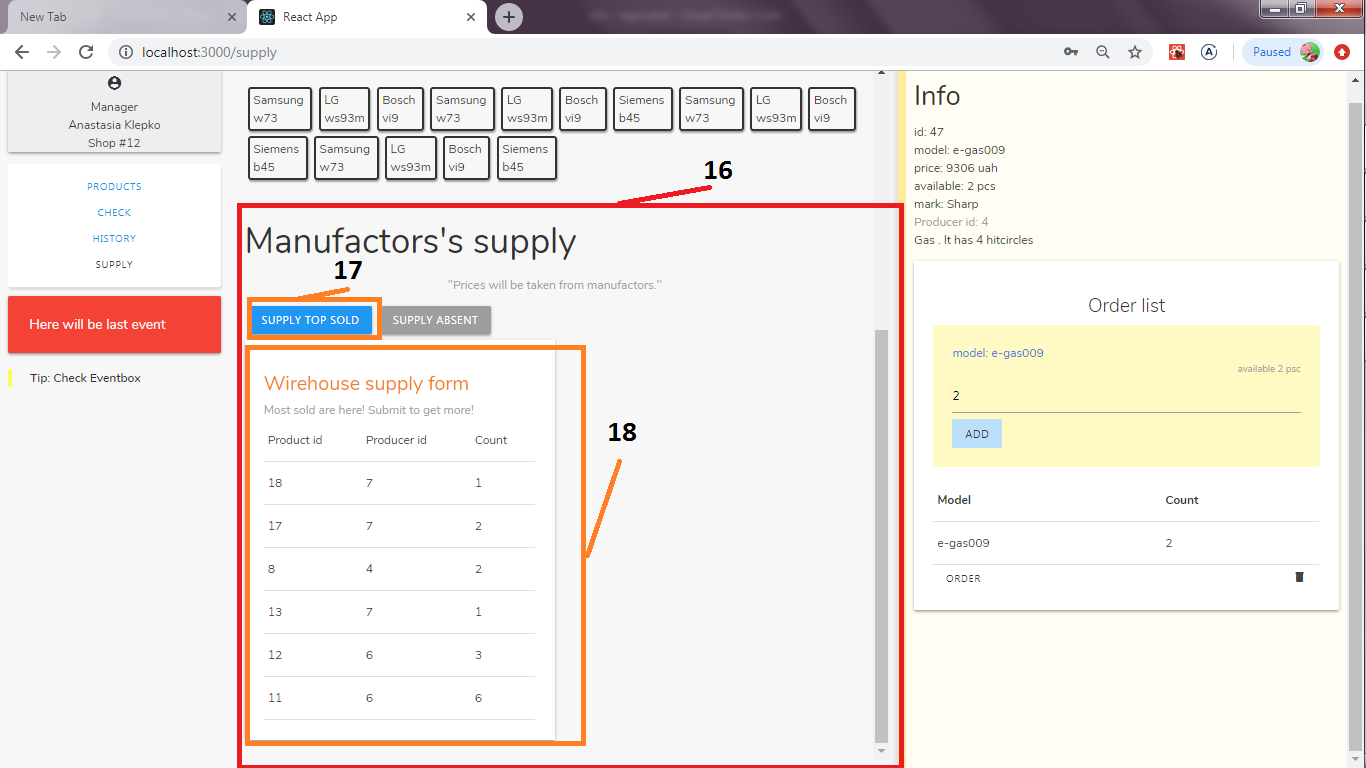


Figure 7.9 –Manager supply page 2

**CONCLUSION**

There was done the application to work with the network of shops those sale household goods. To enrich this result was analyzed the information system, defined tasks on the information system, created the model of subject domain, created database.

In future it might be done:

1. Role entry in from one server
2. Translating login and password in view of JWT (secure encrypted web tokens)
3. Printing of all products in the particular shop
4. Admin to manage workers, and their salaries
5. Supplier that can contact to the manufactory and gather order by themselves

# LIST OF SOURCES USED

1 Web – [Node.js **Introduction**]-https://www.w3schools.com/nodejs/nodejs\_intro.asp

2 Web – [What is PostgreSQL] - http://www.postgresqltutorial.com/what-is-postgresql/

3 Web – [Apollo Docs] - <https://www.apollographql.com/docs/react/>

4 Web – [Using PostGraphile as a Library] – https://www.graphile.org/postgraphile/usage-library/

5 Web – [PostGraphile Chat] – https://discordapp.com/channels/489127045289476126/493799100014264331

6 Web – [React Docs] – https://uk.reactjs.org/docs/getting-started.htmlWeb – [name] – clink

# Appendix A

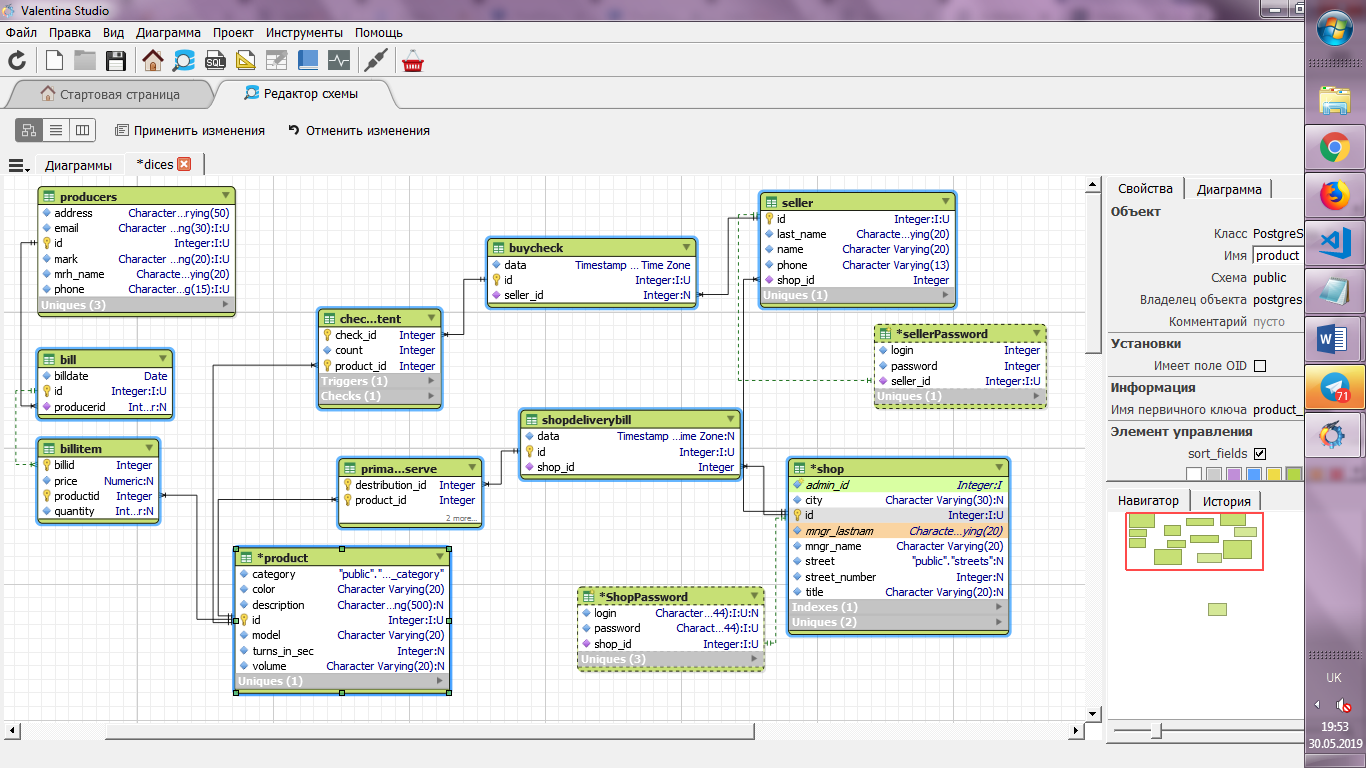


Figure A1 - The ER diagram

# Appendix B

# CREATING TABLES

**/\* TYPE: cat\_category \*/**

CREATE TYPE public.tech\_category AS ENUM

('fridge', 'washer', 'stove');

**/\* TYPE: streets \*/**

CREATE TYPE public.streets AS ENUM

('pr. Shevchenco', 'Govorova', 'Ekaterinenska', 'Sadova', 'Gretska', 'Literaturna', 'small arnautska', 'Filatova', 'large arnautska');

/\* TYPE auth\*/

CREATE TYPE public.auth AS

(

role\_name character varying,

role\_id integer,

shop\_id integer

);

**/\* Table: producers \*/**

CREATE TABLE public.producers

(

id integer NOT NULL DEFAULT nextval('producers\_id\_seq'::regclass),

mrh\_name character varying(20) COLLATE pg\_catalog."default" NOT NULL DEFAULT 'merchandiser'::character varying,

mark character varying(20) COLLATE pg\_catalog."default" NOT NULL,

phone character varying(15) COLLATE pg\_catalog."default" NOT NULL,

email character varying(30) COLLATE pg\_catalog."default" NOT NULL,

address character varying(50) COLLATE pg\_catalog."default" NOT NULL DEFAULT 'UA'::character varying,

CONSTRAINT producers\_pkey PRIMARY KEY (id),

CONSTRAINT producers\_email\_key UNIQUE (email)

,

CONSTRAINT producers\_mark\_key UNIQUE (mark)

,

CONSTRAINT producers\_phone\_key UNIQUE (phone)

)

CREATE TABLE Department

(DeptKod INTEGER NOT NULL CHECK(DeptKod>0) PRIMARY KEY,

DeptName CHAR(40) NOT NULL UNIQUE,

PKod INTEGER REFERENCES Department(DeptKod));

**/\* Table: bill \*/**

CREATE TABLE public.bill

(

id integer NOT NULL DEFAULT nextval('bill\_id\_seq'::regclass),

billdate date NOT NULL,

producerid integer,

CONSTRAINT bill\_pkey PRIMARY KEY (id),

CONSTRAINT bill\_producerid\_fkey FOREIGN KEY (producerid)

REFERENCES public.producers (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

**/\* Table: product \*/**

CREATE TABLE public.product

(

id integer NOT NULL DEFAULT nextval('product\_id\_seq'::regclass),

model character varying(20) COLLATE pg\_catalog."default" NOT NULL,

color character varying(20) COLLATE pg\_catalog."default" NOT NULL,

category tech\_category NOT NULL,

volume character varying(20) COLLATE pg\_catalog."default",

description character varying(500) COLLATE pg\_catalog."default",

turns\_in\_sec integer,

CONSTRAINT product\_pkey PRIMARY KEY (id)

)

**/\* Table: billitem \*/**

CREATE TABLE public.billitem

(

quantity integer,

price numeric(9,2),

billid integer NOT NULL,

productid integer NOT NULL,

CONSTRAINT billitem\_pkey PRIMARY KEY (billid, productid),

CONSTRAINT billitem\_billid\_fkey2 FOREIGN KEY (billid)

REFERENCES public.bill (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE,

CONSTRAINT billitem\_productid\_fkey FOREIGN KEY (productid)

REFERENCES public.product (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

)

**/\* Table: Shop \*/**

CREATE TABLE public.shop

(

id integer NOT NULL DEFAULT nextval('shop\_id\_seq'::regclass),

title character varying(20) COLLATE pg\_catalog."default" DEFAULT 'FW shop'::character varying,

mngr\_name character varying(20) COLLATE pg\_catalog."default" NOT NULL,

mngr\_lastname character varying(20) COLLATE pg\_catalog."default" NOT NULL,

street streets,

street\_number integer,

city character varying(30) COLLATE pg\_catalog."default" DEFAULT 'Odessa'::character varying,

CONSTRAINT shop\_pkey PRIMARY KEY (id),

CONSTRAINT shop\_mngr\_name\_mngr\_lastname\_key UNIQUE (mngr\_name, mngr\_lastname)

,

CONSTRAINT shop\_street\_street\_number\_key UNIQUE (street, street\_number)

)

**/\* Table: Seller \*/**

CREATE TABLE public.seller

(

id integer NOT NULL DEFAULT nextval('seller\_id\_seq'::regclass),

name character varying(20) COLLATE pg\_catalog."default" NOT NULL,

last\_name character varying(20) COLLATE pg\_catalog."default" NOT NULL,

phone character varying(13) COLLATE pg\_catalog."default" NOT NULL,

shop\_id integer NOT NULL,

CONSTRAINT seller\_pkey PRIMARY KEY (id),

CONSTRAINT seller\_name\_last\_name\_key UNIQUE (name, last\_name)

,

CONSTRAINT seller\_shop\_id\_fkey FOREIGN KEY (shop\_id)

REFERENCES public.shop (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

/\* Table: BuyCheck \*/

CREATE TABLE public.buycheck

(

id integer NOT NULL DEFAULT nextval('buycheck\_id\_seq'::regclass),

seller\_id integer,

data timestamp without time zone NOT NULL,

CONSTRAINT buycheck\_pkey PRIMARY KEY (id),

CONSTRAINT buycheck\_seller\_id\_fkey FOREIGN KEY (seller\_id)

REFERENCES public.seller (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

**/\* Table: CheckContent \*/**

CREATE TABLE public.checkcontent

(

check\_id integer NOT NULL,

product\_id integer NOT NULL,

count integer NOT NULL,

CONSTRAINT checkcontent\_pkey PRIMARY KEY (check\_id, product\_id),

CONSTRAINT checkcontent\_check\_id\_fkey FOREIGN KEY (check\_id)

REFERENCES public.buycheck (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE,

CONSTRAINT checkcontent\_productid\_fkey FOREIGN KEY (product\_id)

REFERENCES public.product (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE,

CONSTRAINT checkcontent\_count\_check CHECK (count >= 0)

)

**/\* Table: ShopDeliveryBill \*/**

CREATE TABLE public.shopdeliverybill

(

id integer NOT NULL DEFAULT nextval('shopdeliverybill\_id\_seq'::regclass),

shop\_id integer NOT NULL,

data timestamp without time zone,

CONSTRAINT shopdeliverybill\_pkey PRIMARY KEY (id),

CONSTRAINT shopdeliverybill\_shop\_id\_fkey FOREIGN KEY (shop\_id)

REFERENCES public.shop (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

**/\* Table: PrimaryReserve \*/**

CREATE TABLE public.primaryreserve

(

destribution\_id integer NOT NULL,

product\_id integer NOT NULL,

quantity integer,

CONSTRAINT primaryreserve\_pkey PRIMARY KEY (destribution\_id, product\_id),

CONSTRAINT primaryreserve\_destribution\_id\_fkey FOREIGN KEY (destribution\_id)

REFERENCES public.shopdeliverybill (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE,

CONSTRAINT primaryreserve\_productid\_fkey FOREIGN KEY (product\_id)

REFERENCES public.product (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

)

**/\* Table: SellerPassword\*/**

CREATE TABLE public.sellerpassword

(

sellerid integer NOT NULL,

login character varying(32) COLLATE pg\_catalog."default" NOT NULL,

password character varying(32) COLLATE pg\_catalog."default",

CONSTRAINT sellerpassword\_pkey PRIMARY KEY (login, sellerid),

CONSTRAINT sellerpassword\_sellerid\_fkey FOREIGN KEY (sellerid)

REFERENCES public.seller (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

**/\* Table: ShopPassword \*/**

CREATE TABLE public.shoppassword

(

shopid integer NOT NULL,

login character varying(32) COLLATE pg\_catalog."default" NOT NULL,

password character varying(32) COLLATE pg\_catalog."default",

CONSTRAINT shoppassword\_pkey PRIMARY KEY (login, shopid),

CONSTRAINT shoppassword\_shopid\_fkey FOREIGN KEY (shopid)

REFERENCES public.shop (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

**/\*table Log as independent to login control \*/**

CREATE TABLE public.log

(

date timestamp without time zone,

rolename character varying COLLATE pg\_catalog."default",

rolename\_id integer

)

# Appendix C

# CREATING VIEWS, ROLES AND THEIR PRIVILEGES

**/\*view: viewfullchecksext \*/**

CREATE OR REPLACE VIEW public.viewfullchecksext AS

SELECT bc.id,

cc.product\_id,

cc.count,

bc.seller\_id,

sl.shop\_id,

bc.data

FROM checkcontent cc

JOIN buycheck bc ON bc.id = cc.check\_id

JOIN seller sl ON sl.id = bc.seller\_id

ORDER BY bc.id, sl.shop\_id;

**/\*view: productsview \*/**

CREATE OR REPLACE VIEW public.productsview AS

WITH t1 AS (

SELECT viewfullchecksext.shop\_id,

viewfullchecksext.product\_id,

sum(viewfullchecksext.count) AS sold

FROM viewfullchecksext

GROUP BY viewfullchecksext.shop\_id, viewfullchecksext.product\_id

ORDER BY (ROW(viewfullchecksext.shop\_id, viewfullchecksext.product\_id))

), t2 AS (

SELECT sb.shop\_id,

pm.product\_id,

sum(pm.quantity) AS deliver

FROM primaryreserve pm

JOIN shopdeliverybill sb ON sb.id = pm.destribution\_id

GROUP BY sb.shop\_id, pm.product\_id

ORDER BY (ROW(sb.shop\_id, pm.product\_id))

), t3 AS (

SELECT sh.id AS shop\_id,

pr\_1.id AS prod\_id,

pd\_1.mark,

pr\_1.model,

pr\_1.color,

pr\_1.category,

bi\_1.price,

pd\_1.id AS producer\_id

FROM product pr\_1

JOIN billitem bi\_1 ON bi\_1.productid = pr\_1.id

JOIN bill bill\_1 ON bill\_1.id = bi\_1.billid

JOIN producers pd\_1 ON bill\_1.producerid = pd\_1.id

JOIN primaryreserve pm ON pr\_1.id = pm.product\_id

JOIN shopdeliverybill sdb ON pm.destribution\_id = sdb.id

JOIN shop sh ON sh.id = sdb.shop\_id

GROUP BY sh.id, pr\_1.id, pd\_1.id, pr\_1.model, pr\_1.color, pr\_1.category, bi\_1.price

ORDER BY (ROW(sh.id, pr\_1.id, pd\_1.mark, pr\_1.model, pr\_1.color, pr\_1.category, bi\_1.price))

), jti AS (

SELECT t2.shop\_id,

t2.product\_id,

t2.deliver,

COALESCE(t1.sold, 0::bigint)::integer AS solded

FROM t2

LEFT JOIN t1 ON t1.shop\_id = t2.shop\_id AND t1.product\_id = t2.product\_id

), jt AS (

SELECT jti.shop\_id,

jti.product\_id,

jti.deliver,

jti.solded,

jti.deliver - jti.solded AS aval\_count

FROM jti

)

SELECT pr.category,

pd.id AS producer\_id,

pd.mark,

pr.id AS prod\_id,

pr.model,

pr.color,

jt.aval\_count,

bi.price,

jt.shop\_id

FROM product pr

JOIN billitem bi ON bi.productid = pr.id

JOIN bill ON bill.id = bi.billid

JOIN producers pd ON bill.producerid = pd.id

JOIN jt ON jt.product\_id = pr.id

GROUP BY jt.shop\_id, pr.id, bi.price, pd.id, pr.model, pr.category, pd.mark, jt.aval\_count;

**/\*view: wirehouse\_view \*/**

CREATE OR REPLACE VIEW public.wirehouse\_view AS

WITH t1 AS (

SELECT DISTINCT bi.productid,

pd.mark,

round(bi.price) AS price,

sum(bi.quantity) OVER (PARTITION BY bi.productid) AS all\_count

FROM billitem bi

JOIN bill ON bill.id = bi.billid

JOIN producers pd ON bill.producerid = pd.id

ORDER BY bi.productid

), t2 AS (

SELECT DISTINCT pr.id,

pr.model,

t1.mark,

pr.category,

t1.price,

t1.all\_count,

sum(pm.quantity) OVER (PARTITION BY pr.id) AS closed\_count

FROM t1

JOIN product pr ON pr.id = t1.productid

LEFT JOIN primaryreserve pm ON pm.product\_id = t1.productid

ORDER BY pr.id

), t3 AS (

SELECT t2.id AS prod\_id,

t2.model,

t2.mark,

t2.category,

t2.price,

t2.all\_count - COALESCE(t2.closed\_count, 0::bigint) AS aval\_count

FROM t2

)

SELECT t3.prod\_id,

t3.model,

t3.mark,

t3.category,

t3.price,

t3.aval\_count

FROM t3

WHERE t3.aval\_count > 0;

**/\*view checkview \*/**

CREATE OR REPLACE VIEW public.checkview AS

SELECT bc.id AS checkviewid,

pr.id AS productid,

pr.model,

pr.category,

pr.color,

cc.count,

sl.id AS seller\_id,

sl.name AS seller\_name,

sl.last\_name AS seller\_lastname,

bi.price + bi.price \* 0.11 AS pricetax,

bc.data AS checkdate

FROM buycheck bc

JOIN checkcontent cc ON bc.id = cc.check\_id

JOIN product pr ON pr.id = cc.product\_id

JOIN seller sl ON sl.id = bc.seller\_id

JOIN billitem bi ON bi.productid = pr.id;

/\*role: manager\*/

CREATE ROLE manager WITH PASSWORD 1111 LOGIN

**/\*role: seller\*/**

CREATE ROLE seller WITH PASSWORD 1234 LOGIN

**/\*grants on tables\*/**

Grant Select on productsview to seller;

Grant Select on productsview to manager;

GRANT ALL ON TABLE bill TO manager;

GRANT ALL ON TABLE billitem TO manager;

GRANT select ON TABLE buycheck TO manager;

GRANT INSERT, DELETE ON TABLE checkcontent TO seller;

GRANT SELECT, INSERT ON TABLE public.log TO manager;

GRANT SELECT, INSERT ON TABLE public.log TO seller;

GRANT ALL ON TABLE public.primaryreserve TO manager;

GRANT SELECT ON TABLE public.producers TO manager;

GRANT INSERT ON TABLE public.product TO manager;

GRANT SELECT ON TABLE public.seller TO manager;

GRANT SELECT ON TABLE public.seller TO seller;

GRANT SELECT ON TABLE public.sellerpassword TO seller;

GRANT SELECT ON TABLE public.sellerpassword TO manager;

GRANT SELECT ON TABLE public.shop TO manager;

GRANT SELECT ON TABLE public.shop TO seller;

GRANT ALL ON TABLE public.shopdeliverybill TO manager;

GRANT SELECT ON TABLE public.shoppassword TO manager;

GRANT SELECT ON TABLE public.shoppassword TO seller;

/\*grants on views\*/

GRANT SELECT ON TABLE public.checkview TO seller;

GRANT SELECT ON TABLE public.checkview TO manager;

GRANT SELECT ON TABLE public.productsview TO seller;

GRANT SELECT ON TABLE public.productsview TO manager;

GRANT SELECT ON TABLE public.wirehouse\_view TO manager;

/\*Grants for functions in appendix D\*/

# Appenix D

# QUERIES SOLVING TASKS

**/\*trigger function: SaleBlockFun\*/**

CREATE FUNCTION public.saleblockfun()

RETURNS trigger

AS $ $

declare avalcount integer;

declare shopid integer;

begin

with tc as ( select sl.shop\_id from buycheck bc join seller sl on bc.seller\_id = sl.id where bc.id = new.check\_id)

select tc.shop\_id into shopid from tc ;

with t as (

select \* from productsview pv where pv.shop\_id = shopid and pv.prod\_id= new.product\_id)

select aval\_count into avalcount from t ;

raise notice 'avalcount=(%)',avalcount;

raise notice 'shop\_id=(%)',shopid;

if(new.count > avalcount)

then

raise exception 'error';

end if;

return new;

end ;

$$ LANGUAGE 'plpgsql';

**/\*trigger: SaleBlock\*/**

CREATE TRIGGER saleblock

BEFORE INSERT

ON public.checkcontent

FOR EACH ROW

EXECUTE PROCEDURE public.saleblockfun();

**/\*function: Sale\*/**

CREATE OR REPLACE FUNCTION public.sale(

sellerid integer,

products integer[],

prcount integer[])

RETURNS TABLE(check\_id integer, product\_id integer, model character varying, category tech\_category, color character varying, count integer, seller integer, "sellerName" character varying, "sellerLastName" character varying, price numeric, date timestamp without time zone)

AS $ $

declare

fin INT;

checkid INT;

checkdate timestamp;

begin

INSERT INTO buycheck (id, seller\_id,data) VALUES (DEFAULT, sellerid, current\_timestamp )

RETURNING id,data into checkid, checkdate ;

raise notice 'checkid=%',checkid;

raise notice 'date=%',checkdate;

fin:= array\_length(products, 1);

for iter in 1..fin

loop

insert into checkcontent(check\_id,product\_id,count) values(checkid,products[iter],prcount[iter]);

end loop;

return query select \* from checkview where (checkview.checkviewid = checkid);

BEGIN

EXCEPTION

WHEN OTHERS THEN

BEGIN

raise notice 'rollback';

ROLLBACK;

END;

END;

end;

$$ LANGUAGE plpgsql;

**/\*function get\_wirehouse\_by\_category\*/**

CREATE OR REPLACE FUNCTION public.get\_wirehouse\_by\_category(

cat character varying)

RETURNS TABLE("prod\_Id" integer, model character varying, mark character varying, price numeric, aval\_count bigint)

LANGUAGE 'plpgsql'

AS $ $

declare cond tech\_category;

begin

cond = cat as tech\_category;

return query select w.prod\_Id,w.model,w.mark,w.price,w.aval\_count from wirehouse\_view w

where w.category = cond;

end;

$$ LANGUAGE plpgsql stable;

**/\*trigger function: supplyblockfun\*/**

CREATE FUNCTION public.supplyblockfun()

RETURNS trigger

AS $ $

declare avalcount integer;

begin

select aval\_count into avalcount from wirehouse\_view wv where wv.prod\_id = new.product\_id;

raise notice 'avalcount=(%)',avalcount;

if(new.quantity > avalcount)

then

raise exception 'error: AMOUNT MORE THAN AVALIBLE';

end if;

return new;

end ;

$$ LANGUAGE 'plpgsql';

**/\*trigger: supplyblock \*/**

CREATE TRIGGER supplyblock

BEFORE INSERT

ON public.primaryreserve

FOR EACH ROW

EXECUTE PROCEDURE public.supplyblockfun();

**/\*function: supply\*/**

CREATE OR REPLACE FUNCTION public.supply(

shopid integer,

products integer[],

prcount integer[])

RETURNS TABLE(destribution\_id integer, shop\_id integer, product\_id integer, model character varying, quantity integer, category tech\_category)

AS $$

declare

fin INT;

dist\_id INT;

dist\_date timestamp;

begin

INSERT INTO shopdeliverybill (id, shop\_id,data) VALUES (DEFAULT, shopid, current\_timestamp )

RETURNING id,data into dist\_id,dist\_date ;

raise notice 'dist\_id =%',dist\_id;

raise notice 'dist\_date=%',dist\_date ;

fin:= array\_length(products, 1);

for iter in 1..fin

loop

insert into primaryreserve (destribution\_id,product\_id,quantity) values(dist\_id,products[iter],prcount[iter]);

end loop;

return query select \* from Shop\_Supply\_View where Shop\_Supply\_View.destribution\_id = dist\_id;

BEGIN

EXCEPTION

WHEN OTHERS THEN

BEGIN

raise notice 'rollback';

ROLLBACK;

END;

END;

end;

$$ LANGUAGE 'plpgsql';

**/\*function: getDeliveryList\*/**

CREATE OR REPLACE FUNCTION public.getdeliverylist(

)

RETURNS TABLE(product\_id integer, producer\_id integer, prod\_count bigint)

AS $$

begin

return query select cv.productid, pd.id as producer\_id, sum(cv.count) from checkview cv join billitem bi

on cv.productid = bi.productid

join bill on bi.billid = bill.id

join producers pd on pd.id = bill.producerid

where ( date\_part('day',age(checkdate)) < 7)

group by(cv.productid,pd.id);

end;

$$ LANGUAGE 'plpgsql'

STABLE ;

**/\*function random\_between \*/**

CREATE OR REPLACE FUNCTION public.random\_between(

low integer,

high integer)

RETURNS integer

AS $ $

BEGIN

RETURN floor(random()\* (high-low + 1) + low);

END;

$$ LANGUAGE 'plpgsql';

**/\*function: postTopBill\*/**

CREATE OR REPLACE FUNCTION public.posttopbill()

RETURNS integer

AS $ $

DECLARE

producers int[];

new\_billid int;

len int;

len2 int;

prods int[];

counts int[];

rand bigint;

new\_prod\_id int;

newprod product%ROWTYPE;

begin

select array(select distinct producer\_id from getDeliveryList()) into producers;

len:= array\_length(producers, 1);

for ind in 1..len

loop

INSERT INTO bill(id, billdate,producerid) VALUES (DEFAULT, current\_date,producers[ind])

RETURNING bill.id into new\_billid;

select array( select product\_id from getDeliveryList() where producer\_id = producers[ind] )into prods;

select array( select prod\_count from getDeliveryList() where producer\_id = producers[ind]) into counts ;

len2 := array\_length(prods, 1);

for iter2 in 1..len2

loop

select \* from product where id = prods[iter2] into newprod;

insert into

product (id,model,color,category,volume,description, turns\_in\_sec)

values (default,newprod.model,newprod.color,newprod.category,newprod.volume,

newprod.description,newprod.turns\_in\_sec)

returning id into new\_prod\_id;

select random\_between(6000,10000)::bigint into rand;

INSERT INTO billitem (billid, price,productid,quantity)

VALUES (new\_billid, rand, new\_prod\_id,counts[iter2]);

end loop;

end loop;

return 1;

end;

$$ LANGUAGE plpgsql;

**/\* function: checkauthseller\*/**

CREATE OR REPLACE FUNCTION public.checkauthseller(

login character varying,

password character varying)

RETURNS integer

AS $$

declare

rec RECORD;

cur Cursor for Select \* from sellerpassword;

begin

Open cur;

LOOP

-- fetch row into the film

FETCH cur INTO rec;

-- exit when no more row to fetch

EXIT WHEN NOT FOUND;

-- build the output

IF rec.login = login and rec.password= password THEN

return rec.sellerid;

END IF;

END LOOP;

-- Close the cursor

CLOSE cur;

RETURN -1;

END;

$ $ LANGUAGE 'plpgsql'

STABLE ;

**/\* function: checkauthmanager\*/**

CREATE OR REPLACE FUNCTION public.checkauthmanager(

login character varying,

password character varying)

RETURNS integer

AS $$

declare

rec RECORD;

cur Cursor for Select \* from shoppassword;

begin

Open cur;

LOOP

-- fetch row into the film

FETCH cur INTO rec;

-- exit when no more row to fetch

EXIT WHEN NOT FOUND;

-- build the output

IF rec.login = login and rec.password= password THEN

return rec.shopid;

END IF;

END LOOP;

-- Close the cursor

CLOSE cur;

RETURN -1;

END;

$$ LANGUAGE 'plpgsql'

STABLE ;

**/\*function: Login\*/**

CREATE OR REPLACE FUNCTION public.login(

login character varying,

pass character varying)

RETURNS auth

AS $$

DECLARE

id\_var int;

id\_var2 int;

shop\_id int;

BEGIN

select checkauthmanager(login,pass) into id\_var;

IF id\_var > 0 THEN

SET role 'manager';

insert into log values (current\_timestamp,'manager' , id\_var);

return ( 'manager'::varchar, id\_var, id\_var );

END IF;

select checkauthseller(login,pass) into id\_var2;

IF id\_var2 > 0 THEN

SET role 'seller';

select seller.shop\_id from seller where id = id\_var2 into shop\_id;

insert into log values (current\_timestamp,'seller' , id\_var2);

return ( 'seller'::varchar,id\_var2,shop\_id );

END IF;

SET role 'norole';

return ( 'norole'::varchar,-1,-1);

END;

$$ LANGUAGE 'plpgsql';

**/\*Grants for functions\*/**

GRANT EXECUTE ON FUNCTION public.checkauthmanager(character varying, character varying) TO seller;

GRANT EXECUTE ON FUNCTION public.checkauthmanager(character varying, character varying) TO manager;

GRANT EXECUTE ON FUNCTION public.checkauthseller(character varying, character varying) TO seller;

GRANT EXECUTE ON FUNCTION public.checkauthseller(character varying, character varying) TO postgres;

GRANT EXECUTE ON FUNCTION public.get\_wirehouse\_by\_category(character varying) TO manager;

GRANT EXECUTE ON FUNCTION public.getdeliverylist() TO manager;

GRANT EXECUTE ON FUNCTION public.login(character varying, character varying) TO seller;

GRANT EXECUTE ON FUNCTION public.login(character varying, character varying) TO manager;

GRANT EXECUTE ON FUNCTION public.posttopbill() TO manager;

GRANT EXECUTE ON FUNCTION public.sale(integer, integer[], integer[]) TO seller;

GRANT EXECUTE ON FUNCTION public.supply(integer, integer[], integer[]) TO manager;