МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

"МИРЭА-Российский технологический университет"

Институт комплексной безопасности и специального приборостроения

(наименование факультета)

КБ-2 «Прикладные информационные технологии»

(наименование кафедры)

**КУРСОВАЯ РАБОТА**

**по дисциплине**

«Базы данных и экспертные системы»

(наименование дисциплины)

**Тема курсовой работы** «Материально-техническое обеспечение учебного процесса»

*(наименование темы)*

Студент группы

БИСО-02-19 Биченков В. П.

*(учебная группа) (Фамилия И.О)*

Руководитель курсовой Кульков Д. А.

работы *(Фамилия И.О)*

Рецензент (при наличии)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*должность, звание, ученая степень (Фамилия И.О)*

Работа представлена к « » июня 2021 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

защите *(Подпись студента)*

«Допущен к защите» « » июня 2021 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(Подпись руководителя)*

Москва, 2021

**ОГЛАВЛЕНИЕ**

Задание (стр. 3)

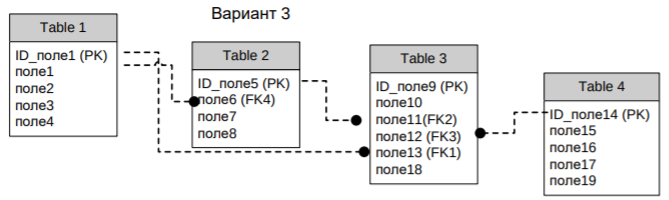
Введение (стр. 4)

1. Реализация таблиц и связей (стр. 5)
2. Реализация вопросов (стр. 7)
3. Реализация индексов (стр. 8)
4. Реализация триггеров (стр. 9)
5. Реализация процедур (стр. 10)
6. Реализация составной транзакции (стр. 25)
7. Реализация курсора на обновление (стр. 26)
8. Реализация функций (стр. 28)
9. Реализация распределения прав (стр. 29)

2

**ЗАДАНИЕ**

|  |  |  |  |
| --- | --- | --- | --- |
| Предметная область | Примерные отношения предметной области | № схемы (вариант схемы) | Триггер (таблиц а №…) |
| Информационно-кассовая система | Учет покупок на кассе (цены, списание со склада), формирование чеков. Информация о кассире | 3 | 4 |



Разработать клиент-серверное приложение, серверная часть которой реализована на

любом SQL сервере, представляющая собой модель предметной области в

соответствии с вариантом задания. В рамках заданной предметной области реализовать

заданную (по варианту) схему отношений, т.е. выделить сущности и их атрибуты, так

чтобы связи между сущностями соответствовали представленной схеме.

3

**ВВЕДЕНИЕ**

В эпоху мировой цифровизации, повсеместного распространения интернета и социальных сетей, базы данных стали практически неотъемлемой частью любой информационной системы. На их основе строится множество сайтов и приложений, которыми мы пользуемся каждый день.

В действительности, такие обыденные действия, как пролистывание ленты в Instagram, отправка сообщения знакомому ВКонтакте или же добавление нового друга на Facebook было бы невозможным без хранения всех этих данных на серверах компании. То же самое касается и игр, ведь практически в каждой из современный тайтлов есть возможность сохранять свои достижения.

Именно поэтому так важно понимать структуру построения баз данных и уметь работать с ними. Такой опыт пригодится как разработчику программного обеспечения, так и, например, аналитикам данных или же специалистам по информационной защите.

В данной работе я постарался разобраться со структурой работы c базами данных на пример PostgreSQL. Это свободно распространяемая СУБД, разработанная как open source проект в Калифорнийском университете Беркли.

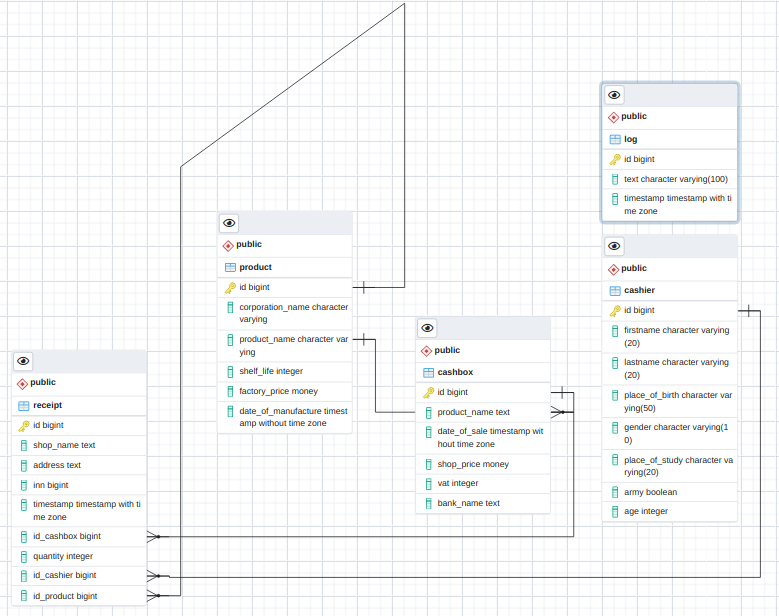
Преимуществом Postgres-а является его универсальность. СУБД доступна на большом количестве операционных систем, включая главных игроков рынка – Windows, MacOS и Linux.

Кроме того, важным плюсом PostgreSQL является поддержка и регулярные обновления. Так, например, последняя актуальная версия (13.3) стала доступна разработчикам совсем недавно – в 2020 году. Более того, 21 мая 2021 вышла первая beta-версия следующеего поколения – PostgreSQL 14.

1. Постоянные таблицы и связи между ними, количество таблиц и наличие связей должно соответствовать заданию, допускается увеличение числа таблиц и их полей для более адекватного представления предметной области.

Для начала создадим ER Диаграмму для нашей базы данных:

4



/\* Завод по производству товаров \*/

**CREATE TABLE** product(

id BIGSERIAL **PRIMARY KEY** NOT NULL,

corporation\_name VARCAHR(50) NOT NULL,

product\_name VARCHAR(20) NOT NULL UNIQUE,

shelf\_life INT NOT NULL,

factory\_price MONEY NOT NULL,

date\_of\_manufacture TIMESTAMP DEFAULT CURRENT\_TIMESTAMP NOT NULL );

/\* Касса \*/

**CREATE TABLE** cashbox(

id BIGSERIAL **PRIMARY KEY** NOT NULL,

product\_name VARCHAR NOT NULL,

date\_of\_sale TIMESTAMP DEFAULT CURRENT\_TIMESTAMP NOT NULL,

shop\_price MONEY NOT NULL,

vat INT NOT NULL,

bank\_name VARCHAR NOT NULL 5

**FOREIGN KEY**(product\_name) PREFERENCES product(product\_name) ON DELETE CASCADE );

/\* Кассир \*/

**CREATE TABLE** cashier(

id BIGSERIAL **PRIMARY KEY** NOT NULL,

firstname VARCHAR(20) NOT NULL,

lastname VARCHAR(20) NOT NULL,

date\_of\_birth DATE NOT NULL,

place\_of\_birth VARCHAR(50) NOT NULL,

gender VARCHAR(10) NOT NULL,

place\_of\_study VARCHAR(20) NOT NULL,

army BOOLEAN NOT NULL);

/\* Чек \*/

**CREATE TABLE** receipt(

id BIGSERIAL PRIMARY KEY NOT NULL,

shop\_name VARCHAR(20) NOT NULL,

address VARCHAR(100) NOT NULL,

inn BIGINT NOT NULL,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

id\_cashbox BIGSERIAL NOT NULL UNIQUE,

quantity INT NOT NULL,

id\_cashier BIGSERIAL NOT NULL UNIQUE,

id\_factory BIGSERIAL NOT NULL UNIQUE,

**FOREIGN KEY**(id\_cashbox) PREFERENCES cashbox(id) ON DELETE CASCADE,

**FOREIGN KEY**(id\_cashier) PREFERENCES cashier(id) ON DELETE CASCADE,

**FOREIGN KEY**(id\_product) PREFERENCES product(id) ON DELETE CASCADE );

/\* Лог \*/

**CREATE TABLE** log(

id BIGSERIAL NOT NULL **PRIMARY KEY**, 6

text VARCHAR(100),

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP);

2. В приложении реализовать не менее пяти запросов, включая (для демонстрации навыков работы)

1. Составной многотабличный запрос с параметром, включающий соединение таблиц и CASE-выражение;

/\* Выводит кол-во купленных продуктов с одним названием \*/

SELECT

product\_name,

SUM (CASE

WHEN id\_product = 13 THEN 1

ELSE 0

END

) AS suquantitym\_of\_products

FROM product, receipt WHERE receipt.id\_product = product.id AND product.id = 13

GROUP BY product\_name

1. На основе обновляющего представления (многотабличного VIEW), в котором критерий упорядоченности задает пользователь при выполнении;

/\* Выводит заводы с количеством привязанных к ним покупок \*/

**CREATE VIEW** [**public**.num\_of\_products](http://public.num_of_products) **AS**

**SELECT DISTINCT**

[product.id](http://factory.id), corporation\_name, product\_name,

(**SELECT** COUNT(\*) **FROM** receipt

**WHERE** id\_product = product.id) **AS** quantity

**FROM** product, receipt **WHERE** [product.id](http://factory.id) = [receipt.id\_product](http://receipt.id_factory);

/\* Сортирует по количеству покупок \*/

**SELECT** \* **FROM** num\_of\_products **ORDER BY** quantity

1. Запрос содержащий коррелированные и некоррелированные подзапросы в разделах SELECT, FROM и WHERE (в каждом хотя бы по одному);

/\* Показывает полную цену в чеке \*/ 7

SELECT receipt.id, receipt.quantity,

(SELECT shop\_price FROM cashbox WHERE cashbox.id = receipt.id\_cashbox),

(SELECT product\_name FROM product WHERE product.id = receipt.id\_product),

cashbox.shop\_price \* receipt.quantity AS full\_price

FROM cashbox, receipt

where receipt.quantity > (select sum(receipt.quantity)/count(receipt.quantity) from receipt)

1. Многотабличный запрос, содержащий группировку записей, агрегативные ф-ции и параметр, используемый в разделе HAVING;

/\* Максимальное количество продукта больше 4 \*/

**SELECT** id\_product, **MAX**(quantity), product\_name

**FROM** receipt, product

**WHERE** [product.id](http://product.id) = receipt.id\_product

**GROUP BY** id\_product, product\_name

**HAVING** (**MAX**(quantity) >= 4)

1. Запрос, содержащий предикат ANY (SOME) или ALL.

SELECT quantity, timestamp

FROM receipt

WHERE id\_product = any(select id from product where corporation\_name='H&M')

3. Создать индексы для увеличения скорости выполнения запросов.

**CREATE INDEX** c\_firstname

**ON** cashier **USING** btree(firstname);

**CREATE INDEX** c\_lastname

**ON** cashier **USING** btree(lastname);

**CREATE INDEX** c\_date\_of\_birth

**ON** cashier **USING** btree(date\_of\_birth);

**CREATE INDEX** c\_place \_of\_birth)

**ON** cashier **USING** btree(place\_of\_birth); 8

**CREATE INDEX** c\_gender

**ON** cashier **USING** btree(gender);

**CREATE INDEX** c\_place\_of\_study

**ON** cashier **USING** btree(place\_of\_study);

**CREATE INDEX** c\_army

**ON** cashier **USING** btree(army);

4. В таблице (в соответствии с вариантом) предусмотреть поле, которое заполняется автоматически по срабатыванию триггера при добавлении, обновлении и удалении данных, иметь возможность продемонстрировать работу триггера при работе приложения. Триггеры должны обрабатывать только те записи, которые были добавлены, изменены или удалены в ходе текущей операции (транзакции).

**CREATE FUNCTION** *add\_to\_log*() **RETURNS** trigger

**LANGUAGE** plpgsql

AS

$$

**DECLARE**

mstr varchar(30);

firs varchar(100);

last varchar(100);

retstr varchar(254);

**BEGIN**

**IF TG\_OP** = 'INSERT' **THEN**

firs = **NEW**.firstname;

last = **NEW**.lastname;

mstr := 'Добавлен новый сотрудник: ';

retstr := mstr || firs || ' ' || last;

**INSERT INTO** log(text, timestamp) values (retstr,**NOW**());

**RETURN NEW**; 9

**ELSIF TG\_OP** = 'UPDATE' **THEN**

firs = **NEW**.firstname;

last = **NEW**.lastname;

mstr := 'Изменен сотрудник: ';

retstr := mstr || firs || ' ' || last;

**INSERT INTO** log(text,timestamp) values (retstr,**NOW**());

**RETURN NEW**;

**ELSIF TG\_OP** = 'DELETE' **THEN**

firs = **OLD**.firstname;

last = **OLD**.lastname;

mstr := 'Уволен сотрудник: ';

retstr := mstr || firs || ' ' || last;

**INSERT INTO** log(text,timestamp) values (retstr,**NOW**());

**RETURN OLD**;

**END IF**;

**END**;

$$;

5. Операции добавления, удаления и обновления реализовать в виде хранимых процедур (с параметрами) хотя бы для одной таблицы; для остальных допустимо использовать возможности связывания полей ввода в приложении с полями БД.

/\* Функция для добавления продукта \*/

create function *add\_product*(\_corporation\_name character varying, \_product\_name character varying, \_shelf\_life integer, \_factory\_price money) returns boolean

language plpgsql

as

$$ 10

DECLARE

return\_value BOOLEAN := TRUE;

product\_count INT;

BEGIN

if (\_corporation\_name IS null)

THEN

raise EXCEPTION 'Введите имя фирмы!';

return\_value := false;

end if;

if (\_product\_name IS null)

THEN

raise EXCEPTION 'Введите название продукта!';

return\_value := false;

end if;

if (\_shelf\_life IS null)

THEN

raise EXCEPTION 'Введите срок годности продукта!';

return\_value := false;

end if;

if (\_factory\_price IS null)

THEN

raise EXCEPTION 'Введите цену продукта на заводе!';

return\_value := false;

end if;

BEGIN

SELECT count(\*) 11

INTO product\_count

FROM product

WHERE product\_name = \_product\_name;

end;

if product\_count > 0

THEN

raise EXCEPTION 'Продукт с таким названием уже существует!';

return\_value := false;

end if;

if return\_value = true

THEN

INSERT INTO product(corporation\_name, product\_name, shelf\_life, factory\_price) VALUES

(\_corporation\_name, \_product\_name, \_shelf\_life, \_factory\_price);

end if;

RETURN return\_value;

end;

$$;

/\* Функция для добавления кассира \*/

create function *add\_cashier*(\_firstname character varying, \_lastname character varying, \_age integer, \_place\_of\_birth character varying, \_gender character varying, \_place\_of\_study character varying, \_army boolean) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

cashiers\_count INT; 12

BEGIN

if (\_firstname IS null)

THEN

raise EXCEPTION 'Введите имя кассира!';

return\_value := false;

end if;

if (\_lastname IS null)

THEN

raise EXCEPTION 'Введите фамилию кассира!';

return\_value := false;

end if;

if (\_age IS null)

THEN

raise EXCEPTION 'Введите возраст кассира!';

return\_value := false;

end if;

if (\_place\_of\_birth IS null)

THEN

raise EXCEPTION 'Введите место рождения кассира!';

return\_value := false;

end if;

if (\_gender IS null)

THEN

raise EXCEPTION 'Введите пол кассира!';

return\_value := false;

end if;

if (\_place\_of\_study IS null) 13

THEN

raise EXCEPTION 'Введите место учебы кассира!';

return\_value := false;

end if;

if (\_army IS null)

THEN

raise EXCEPTION 'Служил кассир или нет?';

return\_value := false;

end if;

BEGIN

SELECT count(\*)

INTO cashiers\_count

FROM cashier

WHERE lastname = \_lastname;

end;

if cashiers\_count > 0

THEN

raise EXCEPTION 'Такой кассир уже есть!';

return\_value := false;

end if;

if return\_value = true

THEN

INSERT INTO cashier(firstname, lastname, age, place\_of\_birth, gender, place\_of\_study, army) VALUES

(\_firstname, \_lastname, \_age, \_place\_of\_birth, \_gender, \_place\_of\_study, \_army);

end if;

RETURN return\_value;

end; 14

$$;

/\* Функция добавления кассы \*/

create function *add\_cashbox*(\_product\_name character varying, \_shop\_price money, \_vat integer, \_bank\_name character varying) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

product\_count INT;

BEGIN

if (\_shop\_price IS null)

THEN

raise EXCEPTION 'Введите магизинную цену!';

return\_value := false;

end if;

if (\_product\_name IS null)

THEN

raise EXCEPTION 'Введите название продукта!';

return\_value := false;

end if;

if (\_vat IS null)

THEN

raise EXCEPTION 'Введите НДС!';

return\_value := false;

end if;

if (\_bank\_name IS null)

THEN 15

raise EXCEPTION 'Введите название банка!';

return\_value := false;

end if;

BEGIN SELECT count(\*)

INTO product\_count

FROM cashbox

WHERE product\_name = \_product\_name;

end;

if product\_count > 0

THEN

raise EXCEPTION 'Продукт с таким названием уже существует!';

return\_value := false;

end if;

if return\_value = true

THEN

INSERT INTO cashbox(product\_name, shop\_price, vat, bank\_name) VALUES

(\_product\_name, \_shop\_price, \_vat, \_bank\_name);

end if;

RETURN return\_value;

end;

$$;

/\* Функция добавления чека \*/

create function *add\_receipt*(\_shop\_name text, \_address text, \_inn bigint, \_quantity integer, \_new\_cashbox text, \_new\_cashier text, \_new\_product text) returns boolean

language plpgsql

as

$$ 16

DECLARE

return\_value BOOLEAN := TRUE;

new\_cashbox\_id INT = -1;

new\_cashier\_id INT = -1;

new\_product\_id INT = -1;

cashbox\_cursor CURSOR (firstCursor text) FOR SELECT id from cashbox WHERE product\_name = firstCursor;

cashier\_cursor CURSOR (secondCursor text) FOR SELECT id from cashier WHERE lastname = secondCursor;

product\_cursor CURSOR (lastCursor text) FOR SELECT id from product WHERE product\_name = lastCursor;

BEGIN

OPEN cashbox\_cursor (firstCursor := \_new\_cashbox);

FETCH cashbox\_cursor INTO new\_cashbox\_id;

OPEN cashier\_cursor (secondCursor := \_new\_cashier);

FETCH cashier\_cursor INTO new\_cashier\_id;

OPEN product\_cursor (lastCursor := \_new\_product);

FETCH product\_cursor INTO new\_product\_id;

if (new\_cashbox\_id = -1)

THEN

return\_value := false;

end if;

if (new\_cashier\_id = -1)

THEN

return\_value := false;

end if;

if (new\_product\_id = -1)

THEN

return\_value := false; 17

end if;

if return\_value = true

THEN

INSERT INTO receipt(shop\_name, address, inn, id\_cashbox, quantity, id\_cashier, id\_product)

VALUES(\_shop\_name, \_address, \_inn, new\_cashbox\_id, \_quantity, new\_cashier\_id, new\_product\_id);

end if;

RETURN return\_value;

end;

$$;

/\* Функция изменения продукта \*/

create function *change\_product*(\_new\_corp\_name character varying, \_old\_product\_name character varying, \_new\_product\_name text, \_new\_shelf\_life integer, \_new\_factory\_price money) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

product\_check INT;

BEGIN

begin

SELECT count(\*) INTO product\_check

FROM product WHERE product\_name = \_new\_product\_name;

end;

if (\_old\_product\_name = \_new\_product\_name)

THEN

product\_check = product\_check - 1;

end if; 18

if (product\_check > 0)

THEN

raise EXCEPTION 'Продукт с таким названием уже существует';

return\_value := false;

end if;

if (return\_value = true)

THEN

UPDATE product set

corporation\_name = \_new\_corp\_name,

product\_name = \_new\_product\_name,

shelf\_life = \_new\_shelf\_life,

factory\_price = \_new\_factory\_price

WHERE product\_name = \_old\_product\_name;

end if;

return return\_value;

end;

$$;

/\* Изменение кассы \*/

create function *change\_cashbox*(\_old\_product\_name character varying, \_new\_product\_name character varying, \_new\_shop\_price money, \_new\_vat integer, \_new\_bank\_name character varying) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

product\_check INT;

BEGIN 19

begin

SELECT count(\*) INTO product\_check

FROM cashbox WHERE product\_name = \_new\_product\_name;

end;

if (\_old\_product\_name = \_new\_product\_name)

THEN

product\_check = product\_check - 1;

end if;

if (product\_check > 0)

THEN

raise EXCEPTION 'Продукт с таким названием уже существует';

return\_value := false;

end if;

if (return\_value = true)

THEN

UPDATE cashbox set

product\_name = \_new\_product\_name,

shop\_price = \_new\_shop\_price,

vat = \_new\_vat,

bank\_name = \_new\_bank\_name

WHERE product\_name = \_old\_product\_name;

end if;

return return\_value;

end;

$$;

/\* Функция изменения кассира \*/

create function *change\_cashier*(\_new\_firstname character varying, \_old\_lastname character varying,

20

\_new\_lastname character varying, \_new\_place\_of\_birth character varying, \_new\_gender character varying, \_new\_place\_of\_study character varying, \_new\_army boolean, \_new\_age integer) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

cashier\_check INT;

BEGIN

begin

SELECT count(\*) INTO cashier\_check

FROM cashier WHERE lastname = \_new\_lastname;

end;

if (\_old\_lastname = \_new\_lastname)

THEN

cashier\_check = cashier\_check - 1;

end if;

if (cashier\_check > 0)

THEN

raise EXCEPTION 'Продукт с таким названием уже существует';

return\_value := false;

end if;

if (return\_value = true)

THEN

UPDATE cashier set

firstname = \_new\_firstname,

lastname = \_new\_lastname,

place\_of\_birth = \_new\_place\_of\_birth, 21

gender = \_new\_gender,

place\_of\_study = \_new\_place\_of\_study,

army = \_new\_army,

age = \_new\_age

WHERE lastname = \_old\_lastname;

end if

return return\_value;

end;

$$;

/\* Удаление продукта \*/

create function *delete\_product*(\_product\_name character varying) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

product\_check INT;

BEGIN

begin

SELECT count(\*) INTO product\_check

FROM product WHERE product\_name = \_product\_name;

end;

if (product\_check = 0)

THEN

raise EXCEPTION 'Нет продуктов с таким названием!';

return\_value := false;

end if;

if (product\_check = 1) 22

THEN

DELETE FROM product

WHERE product\_name = \_product\_name;

end if;

RETURN return\_value;

end;

$$;

/\* Удаление кассы \*/

create function *delete\_cashbox*(\_name text) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

product\_check INT;

BEGIN

begin

SELECT count(\*) INTO product\_check

FROM cashbox WHERE product\_name = \_name;

end;

if (product\_check = 0)

THEN

raise EXCEPTION 'Нет продуктов с таким названием!';

return\_value := false;

end if;

if (product\_check = 1)

THEN

DELETE FROM cashbox 23

WHERE product\_name = \_name;

end if;

RETURN return\_value;

end;

$$;

/\* Удаление чека \*/

create function *delete\_receipt*(\_id integer) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

receipt\_check INT;

BEGIN

begin

SELECT count(\*) INTO receipt\_check

FROM receipt WHERE id = \_id;

end;

if (receipt\_check = 0)

THEN

raise EXCEPTION 'Нет такого чека!';

return\_value := false;

end if;

if (receipt\_check = 1)

THEN

DELETE FROM receipt

WHERE id = \_id;

end if; 24

RETURN return\_value;

end;

$$;

/\* Удаление кассира \*/

create function *delete\_cashier*(\_lastname character varying) returns boolean

language plpgsql

as

$$

DECLARE

return\_value BOOLEAN := TRUE;

cashier\_check INT;

BEGIN

begin

SELECT count(\*) INTO cashier\_check

FROM cashier WHERE lastname = \_lastname;

end;

if (cashier\_check = 0)

THEN

raise EXCEPTION 'Нет такого кассира!';

return\_value := false;

end if;

if (cashier\_check = 1)

THEN

DELETE FROM cashier

WHERE lastname = \_lastname;

end if;

RETURN return\_value;

end; 25

$$;

6. Реализовать отдельную хранимую процедуру, состоящую из нескольких отдельных операций в виде единой транзакции, которая при определенных условиях может быть зафиксирована или откатана.

**CREATE PROCEDURE** public.*proc\_product*(\_corp\_name text, \_product\_name text, \_shelf\_life int, \_factory\_price money)

**LANGUAGE** plpgsql

**AS**

$$**DECLARE**

repeat\_product **INT**;

**BEGIN**

**SELECT** count(\*) into repeat\_product

from product

where product\_name = \_product\_name;

if repeat\_product > 1

**THEN ROLLBACK**;

**ELSE**

**INSERT INTO** product(corporation\_name, product\_name, shelf\_life, factory\_price)

**VALUES**(\_corp\_name, \_product\_name, \_shelf\_life, \_factory\_price);

**COMMIT** ;

end if;

end;

$$

7. В триггере или хранимой процедуре реализовать курсор на обновление отдельных данных.

**CREATE OR REPLACE FUNCTION** public.*change\_receipt*(\_shop\_name text, \_address text, \_inn bigint, \_quantity int, \_new\_cashbox text, \_new\_cashier text, \_new\_product text) 26

**RETURNS** boolean

**LANGUAGE** plpgsql

**AS** $function$**DECLARE**

return\_value **BOOLEAN** := **TRUE**;

new\_cashbox\_id **INT** = -1;

new\_cashier\_id **INT** = -1;

new\_product\_id **INT** = -1;

cashbox\_cursor **CURSOR** (firstCursor text) **FOR SELECT** id **FROM** cashbox **WHERE** bank\_name = firstCursor **FOR UPDATE**;

cashier\_cursor **CURSOR** (secondCursor text) **FOR SELECT** id **FROM** cashier **WHERE** firstname = secondCursor **FOR UPDATE**;

product\_cursor **CURSOR** (lastCursor text) **FOR SELECT** id **FROM** product **WHERE** product\_name = lastCursor **FOR UPDATE**;

**BEGIN**

**OPEN** cashbox\_cursor (firstCursor := \_new\_cashbox);

**FETCH** cashbox\_cursor **INTO** new\_cashbox\_id;

**OPEN** cashier\_cursor (secondCursor := \_new\_cashier);

**FETCH** cashier\_cursor **INTO** new\_cashier\_id;

**OPEN** product\_cursor (lastCursor := \_new\_product);

**FETCH** product\_cursor **INTO** new\_product\_id;

if (new\_cashbox\_id = -1)

**THEN**

return\_value := false;

end if;

if (new\_cashier\_id = -1)

**THEN**

return\_value := false;

end if;

if (new\_product\_id = -1)

**THEN** 27

return\_value := false;

end if;

if return\_value = true

**THEN**

**UPDATE** receipt

**SET** shop\_name = \_new\_shop\_name,

address = \_new\_address,

inn = \_new\_inn,

id\_cashbox = new\_cashbox\_id,

quantity = \_new\_quantity,

id\_cashier = new\_cashier\_id,

id\_product = new\_product\_id

**WHERE** id = \_id;

end if;

**RETURN** return\_value;

end;

$function$

8. В запросе (из пункта 2 или в дополнительном к тому перечню) использовать собственную скалярную функцию, а в хранимой процедуре – векторную (или табличную) функцию. Функции сохранить в базе данных.

/\* Скалярная функция \*/

**CREATE FUNCTION** public.*get\_product*(\_id int, \_quantity int)

**RETURNS** int

**LANGUAGE** plpgsql

**AS**

$$ **DECLARE**

\_full\_price int;

**BEGIN** 28

**SELECT** \_quantity \* shop\_price **INTO** \_full\_price

**FROM** cashbox **WHERE** id = \_id;

**RETURN** \_full\_price;

end;

$$

**SELECT** id, shop\_name, address, product\_name ,*get\_product*(id\_cashbox, quantity) **FROM** receipt, product;

/\* Табличная функция \*/

**CREATE FUNCTION** *get\_milk*(\_id integer, \_quantity integer)

**RETURNS** table(\_id bigint, \_corp\_name text, \_product\_name text, \_shelf\_life int, \_factory\_price money, \_date timestamp)

**LANGUAGE** sql

**AS**

$function

**SELECT** id, corporation\_name, product\_name, shelf\_life, factory\_price, date\_of\_manufacture

**FROM** product **WHERE** product\_name = 'milk';

$function$

9. Распределение прав пользователей: предусмотреть не менее двух пользователей с разным набором привилегий. Каждый набор привилегий оформить в виде роли.

cash\_information\_system=# create role db\_user;

CREATE ROLE

cash\_information\_system=# create role db\_admin;

CREATE ROLE

cash\_information\_system=# grant select, update, insert, delete on product, cashbox, log, receipt, cashier to db\_admin;

GRANT

cash\_information\_system=# grant select on product, cashbox, log, receipt, cashier to db\_user;

GRANT

cash\_information\_system=# grant update, insert on  receipt to db\_user;

GRANT

cash\_information\_system=# create user JohnWick password 'qwerty';

CREATE ROLE

cash\_information\_system=# create user Machete password 'user\_qwerty';

CREATE ROLE

cash\_information\_system=# grant db\_user to Machete; 29

GRANT ROLE

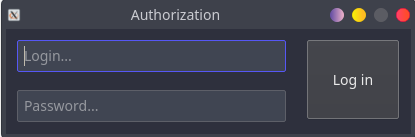
cash\_information\_system=# grant db\_admin to JohnWick;

GRANT ROLE

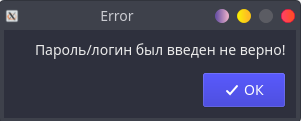
10. Интерфейс приложения

Для написания курсовой работы я использовал Qt на языке программирования C++. Также в разработке использовался Data Grip от JetBrains.

При запуске программы появляется окно авторизации, где пользователь должен ввести свои логин и пароль от базы данных.

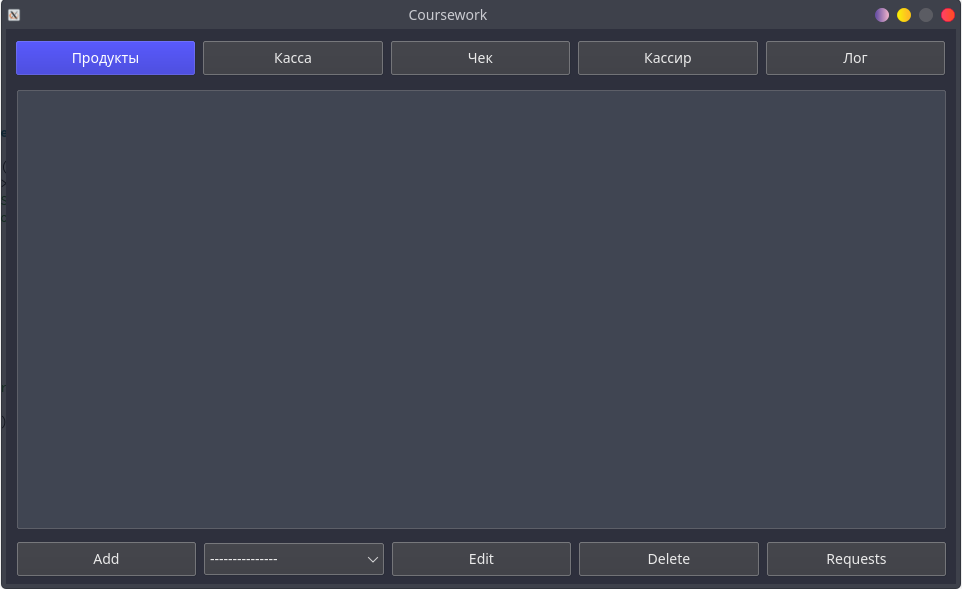


Если ввести неправильные логин/пароль, то появится ошибка.

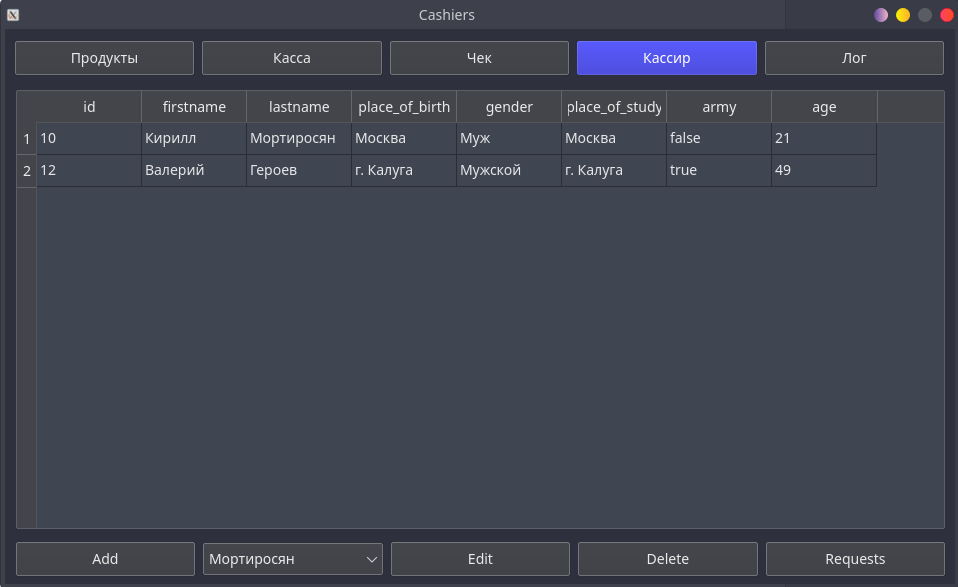


После успешного ввода данных появляется главное окно, в котором можно взаимодействовать с таблицами.

30

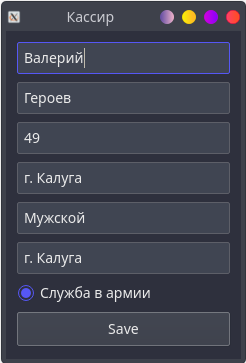


При нажатии на кнопки «Продукты/Касса/Чек/Кассир/Лог» в QTableView появится выбранная таблица.



31

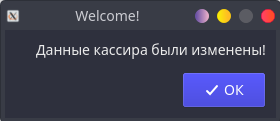
При нажатии на кнопки Add/Edit появляется поле в котором можно добавить/изменить данные в таблице.



Чтобы изменить данные, нужно выбрать поле в QComboBox и нажать Edit.

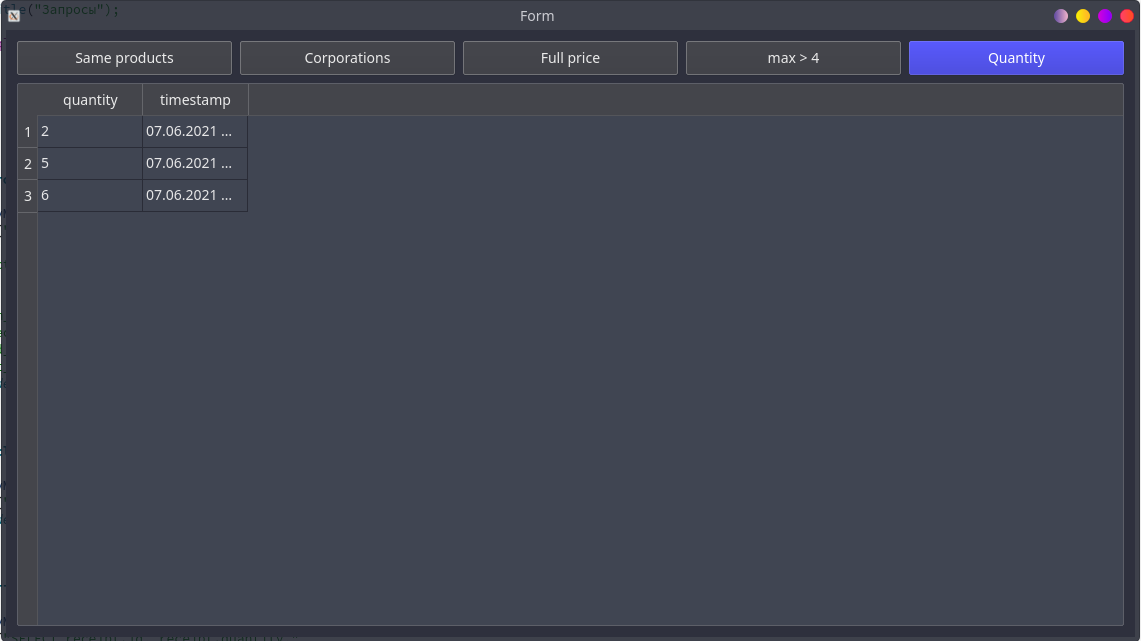
Так же, при выборе поля в QComboBox, можно удалить строку с помощью нажатия кнопки Delete.

После успешного добавления/удаления/изменения записи, приходит уведомление.



32

Так же есть кнопка Requests. Тогда открывается новое окно, и в нем выводятся табилицы с запросами из 2 пункта.



11. Листинг кода

Окно MainWindow:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

MainWindow::setWindowTitle("Authorization");

MainWindow::move(QApplication::desktop()->width()/2, QApplication::desktop()->height()/2);

ui->password\_lineEdit->setEchoMode(QLineEdit::Password); 33

}

MainWindow::~MainWindow()

{

delete ui;

login = "";

password = "";

}

void MainWindow::on\_login\_button\_clicked()

{

login = ui -> login\_lineEdit->text();

password = ui ->password\_lineEdit->text();

db = QSqlDatabase::addDatabase("QPSQL");

db.setDatabaseName("cash\_information\_system");

db.setHostName("localhost");

db.setUserName(login);

db.setPassword(password);

bool checker = db.open();

if (checker){

mainMenu = new main\_menu();

mainMenu -> show();

this->close();

} else {

QMessageBox::about(this, "Error", "Пароль/логин был введен не верно!"); 34

}

ui ->password\_lineEdit->setText("");

}

Окно main\_menu:

#include "main\_menu.h"

#include "ui\_main\_menu.h"

main\_menu::main\_menu(QWidget \*parent) :

QWidget(parent),

ui(new Ui::main\_menu)

{

ui->setupUi(this);

main\_menu::setWindowTitle("Coursework");

QSqlDatabase db = QSqlDatabase::database("cash\_information\_system");

chosen\_table = "none";

cashboxForm = new cashbox("", false);

cashierForm = new cashier("", false);

receiptForm = new receipt(0, false);

productForm = new product("", false);

requestForm = new requests();

}

main\_menu::~main\_menu()

{

delete ui;

} 35

void main\_menu::on\_cashbox\_clicked()

{

main\_menu::setWindowTitle("Cashboxes");

cashboxModel = new QSqlTableModel(this, db);

cashboxModel -> setTable("cashbox");

cashboxModel -> select();

ui -> tableView -> setModel(cashboxModel);

chosen\_table = "cashbox";

combo\_box\_model = new QSqlQueryModel;

QSqlQuery get\_cashbox = QSqlQuery(db);

get\_cashbox.prepare("SELECT product\_name FROM cashbox");

get\_cashbox.exec();

combo\_box\_model->setQuery(get\_cashbox);

ui -> comboBox->setModel(combo\_box\_model);

}

void main\_menu::on\_receipt\_clicked()

{

ui->comboBox->setCurrentIndex(0);

main\_menu::setWindowTitle("Receipt");

receiptModel = new QSqlTableModel(this, db);

receiptModel -> setTable("receipt");

receiptModel ->select(); 36

ui -> tableView -> setModel(receiptModel);

chosen\_table = "receipt";

combo\_box\_model = new QSqlQueryModel;

QSqlQuery get\_receipt = QSqlQuery(db);

get\_receipt.prepare("SELECT id FROM receipt");

get\_receipt.exec();

combo\_box\_model->setQuery(get\_receipt);

ui -> comboBox->setModel(combo\_box\_model);

}

void main\_menu::on\_cashier\_clicked()

{

main\_menu::setWindowTitle("Cashiers");

cashierModel = new QSqlTableModel(this, db);

cashierModel -> setTable("cashier");

cashierModel -> select();

ui -> tableView -> setModel(cashierModel);

chosen\_table = "cashier";

combo\_box\_model = new QSqlQueryModel;

QSqlQuery get\_cashier = QSqlQuery(db);

get\_cashier.prepare("SELECT lastname FROM cashier");

get\_cashier.exec();

combo\_box\_model->setQuery(get\_cashier); 37

ui -> comboBox->setModel(combo\_box\_model);

}

void main\_menu::on\_log\_clicked()

{

main\_menu::setWindowTitle("Logs");

logModel = new QSqlTableModel(this, db);

logModel -> setTable("log");

logModel -> select();

ui -> tableView -> setModel(logModel);

combo\_box\_model = new QSqlQueryModel;

ui->comboBox->setModel(combo\_box\_model);

}

void main\_menu::on\_product\_clicked()

{

main\_menu::setWindowTitle("Products");

productModel = new QSqlTableModel(this, db);

productModel -> setTable("product");

productModel -> select();

ui -> tableView -> setModel(productModel);

chosen\_table = "product";

combo\_box\_model = new QSqlQueryModel; 38

QSqlQuery get\_product = QSqlQuery(db);

get\_product.prepare("SELECT product\_name FROM product");

get\_product.exec();

combo\_box\_model->setQuery(get\_product);

ui -> comboBox->setModel(combo\_box\_model);

}

void main\_menu::on\_add\_clicked()

{

if (chosen\_table == "none"){

QMessageBox::about(this, "Error", "Выберите таблицу которую хотите изменить!");

}

if (chosen\_table == "cashbox"){

cashboxForm = new cashbox("", false);

cashboxForm->show();

}

if (chosen\_table == "product"){

productForm = new product("", false);

productForm->show();

}

if (chosen\_table == "receipt"){

receiptForm->show();

}

if (chosen\_table == "cashier"){

cashierForm = new cashier("", false); 39

cashierForm->show();

}

}

void main\_menu::on\_edit\_clicked()

{

if (chosen\_table == "none"){

QMessageBox::about(this, "Error", "Выберите таблицу которую хотите изменить!");

}

if (chosen\_table == "cashbox"){

cashboxForm = new cashbox(ui->comboBox->currentText(), true);

cashboxForm->show();

}

if (chosen\_table == "product"){

productForm = new product(ui->comboBox->currentText(), true);

productForm->show();

}

if (chosen\_table == "receipt"){

receiptForm = new receipt(ui->comboBox->currentText().toInt(), true);

receiptForm->show();

}

if (chosen\_table == "cashier"){

cashierForm = new cashier(ui->comboBox->currentText(), true);

cashierForm->show();

} 40

}

void main\_menu::on\_requests\_clicked()

{

requestForm->show();

}

void main\_menu::on\_delete\_2\_clicked()

{

QSqlQuery deleteQuery = QSqlQuery(db);

QString item = ui->comboBox->currentText();

bool checker;

if (chosen\_table == "cashbox"){

deleteQuery.prepare("SELECT \* FROM public.delete\_cashbox(:\_name)");

deleteQuery.bindValue(":\_name", item);

checker = deleteQuery.exec();

if (!checker){

QMessageBox::about(this,"Error", "Ошибка! Касса " + item + " не была удалена!");

}

}

if (chosen\_table == "product"){

deleteQuery.prepare("SELECT \* FROM public.delete\_product(:\_product\_name)");

deleteQuery.bindValue(":\_product\_name", item); 41

checker = deleteQuery.exec();

if (!checker){

QMessageBox::about(this,"Error", "Ошибка! Продукт " + item + " не был удален!");

}

}

if (chosen\_table == "cashier"){

deleteQuery.prepare("SELECT \* FROM public.delete\_cashier(:\_lastname)");

deleteQuery.bindValue(":\_lastname", item);

checker = deleteQuery.exec();

if (!checker){

QMessageBox::about(this,"Error", "Ошибка! Кассир " + item + " не был удален!");

}

}

if (chosen\_table == "receipt"){

deleteQuery.prepare("SELECT \* FROM public.delete\_receipt(:\_id)");

deleteQuery.bindValue(":\_id", item.toInt());

checker = deleteQuery.exec();

if (!checker){

QMessageBox::about(this,"Error", "Ошибка! Чек " + item + " не был удален!");

}

}

}

42

Окно добавления/изменения product:

#include "product.h"

#include "ui\_product.h"

product::product(QString name, bool isEditing) :

ui(new Ui::product)

{

ui->setupUi(this);

product::setWindowTitle("Продукты");

QSqlDatabase db = QSqlDatabase::database("cash\_information\_system");

old\_name = name;

ui->product\_name->setText(name);

editing = isEditing;

if (editing){

QSqlQuery editQuery = QSqlQuery(db);

editQuery.prepare("SELECT corporation\_name from product WHERE product\_name = :\_current\_product\_name");

editQuery.bindValue(":\_current\_product\_name", old\_name);

editQuery.exec();

editQuery.first();

ui->corp\_name->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT shelf\_life from product WHERE product\_name = :\_current\_product\_name");

editQuery.bindValue(":\_current\_product\_name", old\_name); 43

editQuery.exec();

editQuery.first();

ui->shelf\_life->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT factory\_price from product WHERE product\_name = :\_current\_product\_name");

editQuery.bindValue(":\_current\_product\_name", old\_name);

editQuery.exec();

editQuery.first();

ui->factory\_price->setText(editQuery.value(0).toString());

}

}

product::~product()

{

delete ui;

}

void product::on\_save\_clicked()

{

QSqlQuery queryModel = QSqlQuery(db);

bool isOk = true;

if (ui->corp\_name->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите название фирмы!");

} 44

if (ui->product\_name->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите название продукта!");

}

if (ui->shelf\_life->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите срок годности!");

}

if (ui->factory\_price->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите заводские цены!");

}

if (isOk){

if (!editing){

queryModel.prepare("select \* from add\_product(:\_corp\_name, :\_product\_name, :\_shelf\_life, :\_factory\_price)");

queryModel.bindValue(":\_corp\_name", ui->corp\_name->text());

queryModel.bindValue(":\_product\_name", ui->product\_name->text());

queryModel.bindValue(":\_shelf\_life", ui->shelf\_life->text());

queryModel.bindValue(":\_factory\_price", ui->factory\_price->text());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Новый продукт!");

ui->corp\_name->setText("");

ui->product\_name->setText("");

ui->shelf\_life->setText("");

ui->factory\_price->setText(""); 45

this->close();

}

else {

QMessageBox::about(this,"Error!", "Продукт не был добавлен!");

}

}

else{

queryModel.prepare("select \* from change\_product(:\_new\_corp\_name, :\_old\_product\_name, :\_new\_product\_name, :\_new\_shelf\_life, :\_new\_factory\_price)");

queryModel.bindValue(":\_new\_corp\_name", ui->corp\_name->text());

queryModel.bindValue(":\_old\_product\_name", old\_name);

queryModel.bindValue(":\_new\_product\_name", ui->product\_name->text());

queryModel.bindValue(":\_new\_shelf\_life", ui->shelf\_life->text());

queryModel.bindValue(":\_new\_factory\_price", ui->factory\_price->text());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Параметры продукта были изменены!");

ui->corp\_name->setText("");

ui->product\_name->setText("");

ui->shelf\_life->setText("");

ui->factory\_price->setText("");

this->close();

}

else {

QMessageBox::about(this,"Error!", "Параметры не были изменены!");

} 46

}

}

}

Окно добавления/изменения cashbox:

#include "cashbox.h"

#include "ui\_cashbox.h"

cashbox::cashbox(QString name, bool isEditing) :

ui(new Ui::cashbox)

{

ui->setupUi(this);

cashbox::setWindowTitle("Касса");

old\_name = name;

editing = isEditing;

QSqlDatabase db = QSqlDatabase::database("cash\_information\_system");

products = new QSqlQueryModel;

QSqlQuery get\_products = QSqlQuery(db);

get\_products.prepare("select product\_name from product");

get\_products.exec();

products->setQuery(get\_products);

ui->product\_name->setModel(products);

if (editing){

ui->product\_name->setCurrentText(old\_name);

QSqlQuery editQuery = QSqlQuery(db);

editQuery.prepare("SELECT shop\_price from cashbox WHERE product\_name = :\_current\_product\_name"); 47

editQuery.bindValue(":\_current\_product\_name", old\_name);

editQuery.exec();

editQuery.first();

ui->shop\_price->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT vat from cashbox WHERE product\_name = :\_current\_product\_name");

editQuery.bindValue(":\_current\_product\_name", old\_name);

editQuery.exec();

editQuery.first();

ui->vat->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT bank\_name from cashbox WHERE product\_name = :\_current\_product\_name");

editQuery.bindValue(":\_current\_product\_name", old\_name);

editQuery.exec();

editQuery.first();

ui->bank\_name->setText(editQuery.value(0).toString());

}

}

cashbox::~cashbox()

{

delete ui;

} 48

void cashbox::on\_save\_clicked()

{

QSqlQuery queryModel = QSqlQuery(db);

bool isOk = true;

if (ui->shop\_price->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите магазинную цену!");

}

if (ui->vat->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите НДС!");

}

if (ui->bank\_name->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите название банка!");

}

if (ui->product\_name->currentText() == ""){

QMessageBox::about(this,"Error!", "Сперва заполните таблицу Product!");

this->close();

}

if (isOk){

if (!editing){

queryModel.prepare("select \* from add\_cashbox(:\_product\_name, :\_shop\_price, :\_vat, :\_bank\_name)");

queryModel.bindValue(":\_product\_name", ui->product\_name->currentText());

queryModel.bindValue(":\_shop\_price", ui->shop\_price->text()); 49

queryModel.bindValue(":\_vat", ui->vat->text());

queryModel.bindValue(":\_bank\_name", ui->bank\_name->text());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Новая пометка в кассе!");

ui->shop\_price->setText("");

ui->vat->setText("");

ui->bank\_name->setText("");

this->close();

}

else {

QMessageBox::about(this,"Error!", "Пометка не была сделана!");

}

}

else{

queryModel.prepare("select \* from change\_cashbox(:\_old\_product\_name, :\_new\_product\_name, :\_new\_shop\_price, :\_new\_vat, :\_new\_bank\_name)");

queryModel.bindValue(":\_old\_product\_name", old\_name);

queryModel.bindValue(":\_new\_product\_name", ui->product\_name->currentText());

queryModel.bindValue(":\_new\_shop\_price", ui->shop\_price->text());

queryModel.bindValue(":\_new\_vat", ui->vat->text());

queryModel.bindValue(":\_new\_bank\_name", ui->bank\_name->text());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Пометка в кассе была изменена!");

ui->shop\_price->setText(""); 50

ui->vat->setText("");

ui->bank\_name->setText("");

this->close();

}

else {

QMessageBox::about(this,"Error!", "Пометка не была изменена!");

}

}

}

}

Окно добавления/изменения receipt:

#include "receipt.h"

#include "ui\_receipt.h"

receipt::receipt(int id, bool isEditing) :

ui(new Ui::receipt)

{

receipt::setWindowTitle("Чек");

ui->setupUi(this);

editing = isEditing;

receiptId = id;

QSqlDatabase db = QSqlDatabase::database("cash\_information\_system");

products = new QSqlQueryModel;

QSqlQuery get\_products = QSqlQuery(db);

get\_products.prepare("select product\_name from product"); 51

get\_products.exec();

products->setQuery(get\_products);

ui->id\_product->setModel(products);

cashboxes = new QSqlQueryModel;

QSqlQuery get\_cashbox = QSqlQuery(db);

get\_cashbox.prepare("select product\_name from cashbox");

get\_cashbox.exec();

cashboxes->setQuery(get\_cashbox);

ui->id\_cashbox->setModel(cashboxes);

cashiers = new QSqlQueryModel;

QSqlQuery get\_cashier = QSqlQuery(db);

get\_cashier.prepare("select lastname from cashier");

get\_cashier.exec();

cashiers->setQuery(get\_cashier);

ui->id\_cashier->setModel(cashiers);

if (editing){

QSqlQuery editQuery = QSqlQuery(db);

editQuery.prepare("SELECT shop\_name from receipt WHERE id = :\_current\_id");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first();

ui->shop\_name->setText(editQuery.value(0).toString()); 52

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT address from receipt WHERE id = :\_current\_id");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first();

ui->address->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT inn from receipt WHERE id = :\_current\_id");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first();

ui->inn->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT quantity from receipt WHERE id = :\_current\_id");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first();

ui->quantity->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT product\_name from cashbox WHERE id = (select id\_cashbox from receipt where id = :\_current\_id)");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first(); 53

cashboxes->setQuery(editQuery);

ui->id\_cashbox->setModel(cashboxes);

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT product\_name from product WHERE id = (select id\_product from receipt where id = :\_current\_id)");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first();

products->setQuery(editQuery);

ui->id\_product->setModel(products);

editQuery = QSqlQuery(db);

editQuery.prepare("SELECT lastname from cashier WHERE id = (select id\_cashier from receipt where id = :\_current\_id)");

editQuery.bindValue(":\_current\_id", receiptId);

editQuery.exec();

editQuery.first();

cashiers->setQuery(editQuery);

ui->id\_cashier->setModel(cashiers);

}

}

receipt::~receipt()

{

delete ui;

} 54

void receipt::on\_Save\_clicked()

{

QSqlQuery queryModel = QSqlQuery(db);

bool isOk = true;

if (ui->shop\_name->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите название магазина!");

}

if (ui->address->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите адрес!");

}

if (ui->inn->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите inn!");

}

if (ui->id\_cashbox->currentText() == ""){

QMessageBox::about(this,"Error!", "Сперва заполните таблицу Cashbox!");

this->close();

}

if (ui->quantity->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите количество продуктов!");

}

if (ui->id\_cashier->currentText() == ""){

QMessageBox::about(this,"Error!", "Сперва заполните таблицу Cashier!"); 55

this->close();

}

if (ui->id\_product->currentText() == ""){

QMessageBox::about(this,"Error!", "Сперва заполните таблицу Product!");

this->close();

}

if (isOk){

if (!editing){

queryModel.prepare("select \* from add\_receipt(:\_shop\_name, :\_address, :\_inn, :\_quantity, :\_new\_cashbox, :\_new\_cashier, :\_new\_product)");

queryModel.bindValue(":\_shop\_name", ui->shop\_name->text());

queryModel.bindValue(":\_address", ui->address->text());

queryModel.bindValue(":\_inn", ui->inn->text());

queryModel.bindValue(":\_quantity", ui->quantity->text());

queryModel.bindValue(":\_new\_cashbox", ui->id\_cashbox->currentText());

queryModel.bindValue(":\_new\_cashier", ui->id\_cashier->currentText());

queryModel.bindValue(":\_new\_product", ui->id\_product->currentText());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Добавлен новый чек!");

ui->shop\_name->setText("");

ui->address->setText("");

ui->inn->setText("");

ui->quantity->setText("");

this->close();

} 56

else {

QMessageBox::about(this,"Error!", "Новый чек не был добавлен!");

this->close();

}

}

else {

queryModel.prepare("select \* from change\_receipt(:\_id, :\_new\_shop\_name, :\_new\_address, :\_new\_inn, :\_new\_quantity, :\_new\_cashbox, :\_new\_cashier, :\_new\_product);");

queryModel.bindValue(":\_id", receiptId);

queryModel.bindValue(":\_new\_shop\_name", ui->shop\_name->text());

queryModel.bindValue(":\_new\_address", ui->address->text());

queryModel.bindValue(":\_new\_inn", ui->inn->text());

queryModel.bindValue(":\_new\_quantity", ui->quantity->text());

queryModel.bindValue(":\_new\_cashbox", ui->id\_cashbox->currentText());

queryModel.bindValue(":\_new\_cashier", ui->id\_cashier->currentText());

queryModel.bindValue(":\_new\_product", ui->id\_product->currentText());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Чек был изменен!");

ui->shop\_name->setText("");

ui->address->setText("");

ui->inn->setText("");

ui->quantity->setText("");

this->close();

}

else { 57

QMessageBox::about(this,"Error!", "Чек не был изменен!");

this->close();

}

}

}

}

Окно добавления/изменения cashier:

#include "cashier.h"

#include "ui\_cashier.h"

cashier::cashier(QString lastname, bool isEditing) :

ui(new Ui::cashier)

{

ui->setupUi(this);

cashier::setWindowTitle("Кассир");

QSqlDatabase db = QSqlDatabase::database("cash\_information\_system");

old\_lastname = lastname;

editing = isEditing;

ui->lastname->setText(lastname);

if (editing){

QSqlQuery editQuery = QSqlQuery(db);

editQuery.prepare("select firstname from cashier where lastname = :\_current\_lastname");

editQuery.bindValue(":\_current\_lastname", old\_lastname);

editQuery.exec();

editQuery.first();

ui->firstname->setText(editQuery.value(0).toString()); 58

editQuery = QSqlQuery(db);

editQuery.prepare("select place\_of\_birth from cashier where lastname = :\_current\_lastname");

editQuery.bindValue(":\_current\_lastname", old\_lastname);

editQuery.exec();

editQuery.first();

ui->place\_of\_birth->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("select gender from cashier where lastname = :\_current\_lastname");

editQuery.bindValue(":\_current\_lastname", old\_lastname);

editQuery.exec();

editQuery.first();

ui->gender->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("select place\_of\_study from cashier where lastname = :\_current\_lastname");

editQuery.bindValue(":\_current\_lastname", old\_lastname);

editQuery.exec();

editQuery.first();

ui->place\_of\_study->setText(editQuery.value(0).toString());

editQuery = QSqlQuery(db);

editQuery.prepare("select army from cashier where lastname = :\_current\_lastname");

editQuery.bindValue(":\_current\_lastname", old\_lastname);

editQuery.exec(); 59

editQuery.first();

ui->army->setChecked(editQuery.value(0).toBool());

editQuery = QSqlQuery(db);

editQuery.prepare("select age from cashier where lastname = :\_current\_lastname");

editQuery.bindValue(":\_current\_lastname", old\_lastname);

editQuery.exec();

editQuery.first();

ui->age->setText(editQuery.value(0).toString());

}

}

cashier::~cashier()

{

delete ui;

}

void cashier::on\_save\_clicked()

{

QSqlQuery queryModel = QSqlQuery(db);

bool isOk = true;

if (ui->firstname->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите имя кассира!");

}

if (ui->lastname->text() == ""){ 60

isOk = false;

QMessageBox::about(this,"Error!", "Введите фамилию кассира!");

}

if (ui->place\_of\_birth->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите место рождения кассира!");

}

if (ui->place\_of\_study->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите место учебы кассира!");

}

if (ui->gender->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите пол кассира!");

}

if (ui->age->text() == ""){

isOk = false;

QMessageBox::about(this,"Error!", "Введите возраст кассира!");

}

if (isOk){

if (!editing){

queryModel.prepare("select \* from add\_cashier(:\_firstname, :\_lastname, :\_age, :\_place\_of\_birth, :\_gender, :\_place\_of\_study, :\_army)");

queryModel.bindValue(":\_firstname", ui->firstname->text());

queryModel.bindValue(":\_lastname", ui->lastname->text());

queryModel.bindValue(":\_age", ui->age->text()); 61

queryModel.bindValue(":\_place\_of\_birth", ui->place\_of\_birth->text());

queryModel.bindValue(":\_gender", ui->gender->text());

queryModel.bindValue(":\_place\_of\_study", ui->place\_of\_birth->text());

queryModel.bindValue(":\_army", ui->army->isChecked());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Новый кассир!");

ui->firstname->setText("");

ui->lastname->setText("");

ui->age->setText("");

ui->place\_of\_birth->setText("");

ui->gender->setText("");

ui->place\_of\_study->setText("");

this->close();

}

else {

QMessageBox::about(this,"Error!", "Кассир не был добавлен!");

}

}

else{

queryModel.prepare("select \* from change\_cashier(:\_new\_firstname, :\_old\_lastname, :\_new\_lastname, :\_new\_place\_of\_birth, :\_new\_gender, :\_new\_place\_of\_study, :\_new\_army, :\_new\_age)");

queryModel.bindValue(":\_new\_firstname", ui->firstname->text());

queryModel.bindValue(":\_old\_lastname", old\_lastname);

queryModel.bindValue(":\_new\_lastname", ui->lastname->text());

queryModel.bindValue(":\_new\_place\_of\_birth", ui->place\_of\_birth->text()); 62

queryModel.bindValue(":\_new\_gender", ui->gender->text());

queryModel.bindValue(":\_new\_place\_of\_study", ui->place\_of\_birth->text());

queryModel.bindValue(":\_new\_army", ui->army->isChecked());

queryModel.bindValue(":\_new\_age", ui->age->text());

isOk = queryModel.exec();

if (isOk){

QMessageBox::about(this,"Welcome!", "Данные кассира были изменены!");

ui->firstname->setText("");

ui->lastname->setText("");

ui->age->setText("");

ui->place\_of\_birth->setText("");

ui->gender->setText("");

ui->place\_of\_study->setText("");

this->close();

}

else {

QMessageBox::about(this,"Error!", "Данные кассира не были изменены!");

}

}

}

}

Окно Requests: 63

#include "requests.h"

#include "ui\_requests.h"

requests::requests(QWidget \*parent) :

QWidget(parent),

ui(new Ui::requests)

{

requests::setWindowTitle("Запросы");

ui->setupUi(this);

QSqlDatabase db = QSqlDatabase::database("cash\_information\_system");

}

requests::~requests()

{

delete ui;

}

void requests::on\_same\_products\_clicked()

{

QSqlQueryModel \*queryModel = new QSqlQueryModel;

queryModel->setQuery("SELECT product\_name, "

"SUM (CASE "

"WHEN id\_product = 13 THEN 1 "

"ELSE 0 "

"END "

") AS suquantitym\_of\_products " 64

"FROM product, receipt "

"WHERE receipt.id\_product = product.id AND product.id = 13 "

"GROUP BY product\_name");

ui->tableView->setModel(queryModel);

}

void requests::on\_corps\_clicked()

{

QSqlQueryModel \*queryModel = new QSqlQueryModel;

queryModel->setQuery("SELECT \* FROM num\_of\_products ORDER BY quantity");

ui->tableView->setModel(queryModel);

}

void requests::on\_full\_price\_clicked()

{

QSqlQueryModel \*queryModel = new QSqlQueryModel;

queryModel->setQuery("SELECT receipt.id, receipt.quantity,"

"(SELECT shop\_price FROM cashbox WHERE cashbox.id = receipt.id\_cashbox),"

"(SELECT product\_name FROM product WHERE product.id = receipt.id\_product),"

"cashbox.shop\_price \* receipt.quantity AS full\_price "

"FROM cashbox, receipt "

"where receipt.quantity > (select sum(receipt.quantity)/count(receipt.quantity) from receipt)");

ui->tableView->setModel(queryModel);

} 65

void requests::on\_max\_clicked()

{

QSqlQueryModel \*queryModel = new QSqlQueryModel;

queryModel->setQuery("SELECT id\_product, MAX(quantity), product\_name "

"FROM receipt, product "

"WHERE product.id = receipt.id\_product "

"GROUP BY id\_product, product\_name "

"HAVING (MAX(quantity) >= 4)");

ui->tableView->setModel(queryModel);

}

void requests::on\_quantity\_clicked()

{

QSqlQueryModel \*queryModel = new QSqlQueryModel;

queryModel->setQuery("SELECT quantity, timestamp FROM receipt "

"WHERE id\_product = any(select id from product where corporation\_name='H&M')");

ui->tableView->setModel(queryModel);

}

Вывод:

С помощью данной курсовой работы, мы закрепили на практике теорию создания баз данных, а так же получили первый опыт написания SQL-запросов. В будущем такие знания помогут создавать более серьезные проекты.

Источники:

https://doc.qt.io/

https://www.postgresql.org/docs/13/index.html

66

https://www.youtube.com/user/ProgrammingKnowledge

https://www.youtube.com/user/ROBOTOBORpro

https://www.qt.io/download

https://www.jetbrains.com/ru-ru/datagrip/

67