МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра «Компьютерных интеллектуальных технологий проектирования»

КУРСОВОЙ ПРОЕКТ

по дисциплине «Базы данных»

Тема: «Разработка CRUD-приложения»

Расчетно-пояснительная записка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Разработал студент  гр. бИВТ - 212 | | Н.Е. Климов | | |
| Подпись, дата Инициалы, фамилия | | |
| Руководитель | | А.А. Филимонова | | |
| Подпись, дата Инициалы, фамилия | | |
| Нормоконтролер | | А.А. Филимонова | | |
| Подпись, дата Инициалы, фамилия | | |
| Защищен |  | | Оценка |  |
| дата | |  |

Воронеж 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра «Компьютерных интеллектуальных технологий проектирования»

ЗАДАНИЕ

# на курсовой проект

по дисциплине «Базы данных»

Тема проекта: «Разработка CRUD-приложения»

Вариант работы «Модуль складского учета»

Студент группы бИВТ- 212, Климов Николай Евгеньевич

Технические условия: Office Word 2016, Qt Creator, SQLiteStudio

Содержание и объем работы (графические работы, расчеты и прочее)

25 страниц, 9 рисунков.

Сроки выполнения этапов: Рассмотрение теоретических сведений (март 2023), выполнение заданий (апрель 2023), оформление пояснительной записки (май 2023).

Срок защиты курсового проекта май 2023

Руководитель А.А. Филимонова

Подпись, дата, инициалы, фамилия

Задание принял студент Н.Е. Климов

Подпись, дата, инициалы, фамилия

# Замечания руководителя

СОДЕРЖАНИЕ

Задание [на курсовой проект 2](#_Toc135867594)

[Замечания руководителя 3](#_Toc135867595)

[ВВЕДЕНИЕ 5](#_Toc135867596)

[1 Постановка задачи 7](#_Toc135867597)

[2 Проектирование базы данных 8](#_Toc135867598)

[2.1 Концептуальное проектирование 8](#_Toc135867599)

[2.2 Логическое проектирование 8](#_Toc135867600)

[2.3 Физическое проектирование 9](#_Toc135867601)

[2.3.1 Выбор СУБД 9](#_Toc135867602)

[2.3.2 Физическая модель базы данных 10](#_Toc135867603)

[3 Реализация модуля складского учета 12](#_Toc135867604)

[3.1 Реализация базы данных 12](#_Toc135867605)

[3.2 Реализация приложения 17](#_Toc135867606)

[4 Тестирование приложения 20](#_Toc135867607)

[ЗАКЛЮЧЕНИЕ 24](#_Toc135867608)

[Список литературы 25](#_Toc135867609)

# ВВЕДЕНИЕ

Базы данных (БД) составляют основу компьютерной поддержки обработки информации практически во всех областях человеческой деятельности.

База данных – это программа, обладающая достаточными возможностями для хранения и организации информации, ввода новых записей, изменения существующих записей, а также извлечения информации и управления ею. Система управления базами данных (СУБД) необходима для выполнения этих функций.

CRUD – приложения являются одной из самых фундаментальных концепций в веб-разработке, и понимание операций CRUD необходимо для работы с данными CRUD расшифровывается как "Создание", "Чтение", "Обновление" и "Удаление" и относится к четырем основным операциям, выполняемым при управлении и манипулировании данными в разработке программного обеспечения. Он относится к четырем основным операциям, выполняемым при управлении и манипулировании данными при разработке программного обеспечения. Эти операции составляют основу большинства приложений, взаимодействующих с базами данных и системами хранения.

Операция CREATE является первой из четырех операций CRUD и добавляет новые записи в базу данных. После подготовки нового экземпляра операция CREATE вставляет его в систему хранения и присваивает ему уникальный идентификатор, чтобы в последствии его можно было искать и изменять.

Второй элемент парадигмы CRUD, операция READ, необходима для получения и отображения данных из системы хранения, такой как база данных. При разработке программного обеспечения операция READ позволяет пользователям получать доступ и просматривать информацию, хранящуюся в приложении.

Операция UPDATE является третьим компонентом структуры CRUD и отвечает за изменение существующих данных в системе хранения. При разработке программного обеспечения операция UPDATE позволяет пользователям изменять информацию, хранящуюся в приложении.

Операция DELETE является последним аспектом структуры CRUD и отвечает за удаление данных из системы хранения. При разработке программного обеспечения операция DELETE позволяет пользователям навсегда удалять определенные записи или ресурсы в приложении.

Цель моей базы данных - это ведение учета товаров склада, а также их стоимости при покупке и продаже. Применение на практике моего проекта подойдет для большинства предприятий, которые занимаются купли-продажей различных товаров.

# 1 Постановка задачи

Требуется разработать базу данных для предприятия, которое занимается купли-продажей различных товаров, используя склады.

Директор предприятия должен иметь приложение с возможностями добавления, редактирования и удаления всех имеющихся данных, а именно товара, который покупает и продает предприятие, склада, которым пользуется предприятие для хранения товара, а также проданных и закупленных предприятием товаров. Также приложение должно быть с графическим интерфейсом с возможностью масштабирования, и с возможностью поиска по любым сущностям таблицы

Для наглядности работы базы данных смоделирована UML диаграмма вариантов использования, представленная на Рисунке 1.

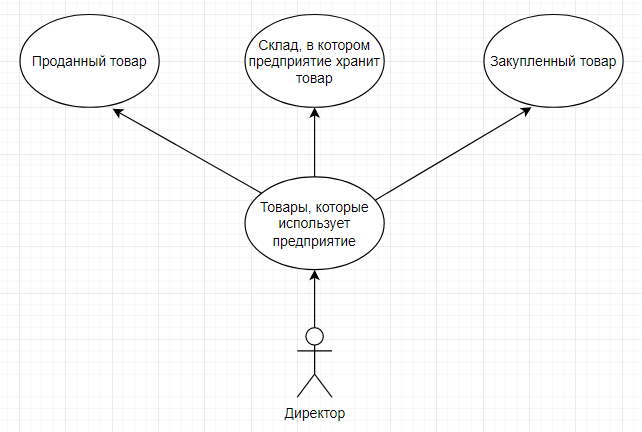


Рисунок 1 - UML диаграмма вариантов использования

# 2 Проектирование базы данных

# 2.1 Концептуальное проектирование

Разрабатываемая база данных подойдет для работы предприятия, которое занимается купле-продажей товаров. Для комфортного использования необходимо реализовать такие сущности, как учет товаров, склада, закупленных и проданных товаров, т.е. соответствующие им таблицы в БД. Последние три таблицы будут связанны с первой по названию товара.

Для дополнения словесного описания каждой связи была создана концептуальная модель базы данных. Результат ее построения представлен на Рисунке 2.

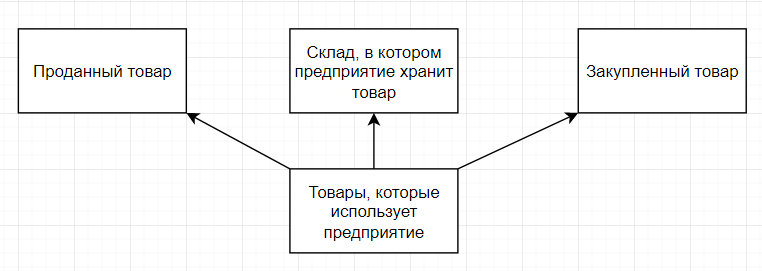


Рисунок 2 – Концептуальная модель базы данных

Таким образом, можно выделить связь: товар, которым пользуется предприятие, может храниться на складе, может быть куплен или может быть продан.

# 2.2 Логическое проектирование

Логическое проектирование - это процесс построения информационной модели на основе существующей модели данных, независимо от условий физической реализации, используемой СУБД или другой системы [2]. На этапе логического проектирования были определены атрибуты разрабатываемых сущностей базы данных.

При логическом проектировании была выбрана реляционная модель данных из-за простоты просмотра и работы с табличным представлением данных. Исходя из концептуального дизайна, каждая сущность стала таблицей. Связи между таблицами были установлены с помощью механизмов первичных и внешних ключей (Рисунок 3). Обозначение: # - уникальный идентификатор.

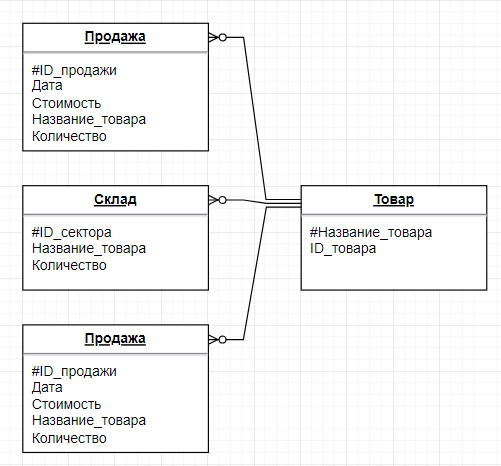


Рисунок 3 - Логическая модель базы данных

# 2.3 Физическое проектирование

# 2.3.1 Выбор СУБД

Для реализации базы данных была выбрана встраиваемая СУБД SQLiteStudio. Такая СУБД интегрируется в приложение в виде отдельного модуля. Она используется для управления данными только внутри него.

Встроенная СУБД предназначена для хранения данных локально в данном приложении не предназначена для коллективного использования в сети. Физически встроенные СУБД в основном реализованы в виде подключаемых библиотек. Они также быстрее традиционных клиент-серверных СУБД и не требуют установки серверов, поэтому востребованы для локального программного обеспечения, обрабатывающего большие объемы данных.

Можно отметить следующие преимущества:

- Значительно легче по весу, чем другие базы данных;

- Перед использованием СУБД не требуется сложная конфигурация или длительная установка;

- СУБД подходит для UNIX-подобных систем, MacOS и Windows;

- Базы данных состоят из записей таблиц, связей между таблицами, индексов и других компонентов; в SQLiteStudio они хранятся в одном файле (database file), который находится на том же устройстве, что и программа.

# 2.3.2 Физическая модель базы данных

Физическая модель базы данных – это модель данных, которая определяет, как представлены данные, и содержит все детали, необходимые СУБД для создания базы данных. Она состоит из структур таблиц, имен и значений столбцов, внешних и первичных ключей и связей между таблицами.

Модель базы данных является реляционной. Поскольку для реализации базы данных была выбрана SQLiteStudio, физическая модель будет использовать типы данных, связанные с этой СУБД (Рисунок 4).

В таблице «Товар» первичным ключом для связи является колонка «Название\_товара». Таблица «Склад» имеет первичный ключ «ID\_продажи» и один внешний «Название\_товара». Таблица «Склад» имеет один первичный ключ «ID\_сектора» и один внешний «Название\_товара». Таблица «Закупка» имеет один первичный ключ «ID\_закупки» и один внешний «Название\_товара».

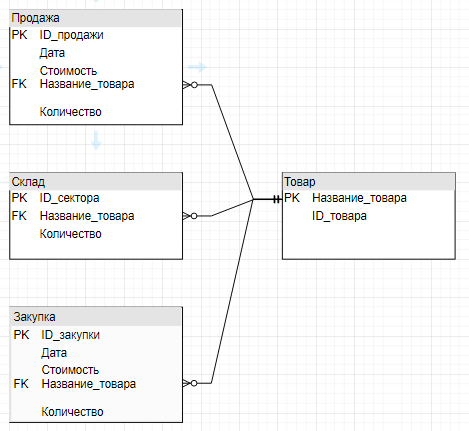


Рисунок 4 – Физическая модель базы данных

Визуализация необходимой базы данных таким образом облегчает создание необходимых таблиц и связей между таблицами в будущем.

Физическая модель базы данных была построена на основе логической модели, показанной на Рисунке 3. Это обеспечило правильное построение модели и ее готовность к следующему этапу создания базы данных, без потери каких-либо предполагаемых сущностей.

# 3 Реализация модуля складского учета

# 3.1 Реализация базы данных

Первым действием при создании базы данных SQL в Qt является помещение строки “QT += core qui sql” в pro-файл проекта. Это необходимо для начала использования SQL [1].

Для подключения к будущим базам данных необходимо подключить соответствующие библиотеки: “QSqlDatabase”, “QSqlQuery”. Это позволит нам создавать таблицы и запросы в базе данных как основу для написания программ. После подключения библиотек создаем объект класса “QSqlDatabase”. Затем, передав драйвер “QSQLITE” в методе “addDatabase” и путь для создания библиотеки в методе “setDatabaseName”, в папке проекта появится файл с расширением “.db” в качестве базы данных после построения.

Реализация подключения БД:

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

db.setDatabaseName("…/TestDDB.db");

if (db.open())

qDebug("База данных открыта");

else

qDebug("Ошибка. База данных не открыта");

Для создания таблиц используем функционал SQLiteStudio.

Cоздание таблицы «Товар»:

CREATE TABLE Товар (

Название\_товара TEXT PRIMARY KEY

NOT NULL,

ID\_товара INTEGER NOT NULL

);

Создание таблицы «Склад»:

CREATE TABLE Склад (

ID\_сектора INTEGER PRIMARY KEY

NOT NULL,

Название\_товара TEXT NOT NULL

REFERENCES Товар (Название\_товара),

Количество INTEGER NOT NULL

);

Создание таблицы «Продажа»:

CREATE TABLE Продажа (

ID\_продажи INTEGER PRIMARY KEY

NOT NULL,

Дата TEXT NOT NULL,

Стоимость INTEGER NOT NULL,

Название\_товара TEXT REFERENCES Товар (Название\_товара)

NOT NULL,

Количество INTEGER NOT NULL

);

Создание таблицы «Закупка»:

CREATE TABLE Закупка (

ID\_закупки INTEGER PRIMARY KEY

NOT NULL,

Дата TEXT NOT NULL,

Стоимость INTEGER NOT NULL,

Название\_товара TEXT NOT NULL

REFERENCES Товар (Название\_товара),

Количество INTEGER NOT NULL

);

Полученные таблицы будут выводиться в форме “tableView” с помощью классов “QSqlQuery” и “QSqlQueryModel” [4].

Реализация вывода таблицы:

query = new QSqlQuery(db);

model = new QSqlTableModel(this,db);

model -> setTable("Закупка");

model ->select();

ui ->tableView->setModel(model);

Кнопки для редактирования, поиска, добавления, удаления и обновления данных таблицы обеспечивают простоту использования.

Функция добавления реализована с помощью “QString”, “QSqlQuery” и “QMessageBox”.

Реализация добавления:

void save\_tovar::on\_save\_clicked()

{

QString nazvanie,id\_tovara;

nazvanie=ui->lineEdit->text();

id\_tovara=ui->lineEdit\_2->text();

QSqlQuery qry;

qry.prepare("INSERT INTO Товар (Название\_товара,ID\_товара) "

"VALUES (:nazvanie, :id\_tovara)");

qry.bindValue(":nazvanie", nazvanie);

qry.bindValue(":id\_tovara", id\_tovara);

if(qry.exec()){

QMessageBox::information(this,"База\_данных","Сохранено");

} else{

QMessageBox::critical(this,"База\_данных",qry.lastError().text());}

}

Функция редактирования реализована с помощью “QString”, “QSqlQuery” и “QMessageBox”.

Реализация редактирования:

void redak\_tovar::on\_redak\_clicked()

{

QString nazvanie,id\_tovara;

nazvanie=ui->lineEdit->text();

id\_tovara=ui->lineEdit\_2->text();

QSqlQuery qry;

qry.prepare("UPDATE Товар set Название\_товара=:nazvanie, ID\_товара=:id\_tovara WHERE Название\_товара=:nazvanie" );

qry.bindValue(":nazvanie", nazvanie);

qry.bindValue(":id\_tovara", id\_tovara);

if(qry.exec()){

QMessageBox::information(this,"База\_данных","Изменено");

} else{

QMessageBox::critical(this,"База\_данных",qry.lastError().text());

}

}

Функция обновления реализована с помощью “QString”, “QSqlQuery”, “QSqlTableModel” и “tableView”.

Реализация обновления:

# void tovar::on\_obnov\_clicked()

# {

# QString id;

# QSqlQuery qry;

# qry.prepare("select \* from Товар");

# model = new QSqlTableModel(this,db);

# model->setTable("Товар");

# model->select();

# ui->tableView->setModel(model);

# }

Функция поиска реализована с помощью “QString”, “ QSqlQuery”, “setFilter” и “tableView”.

Реализация поиска:

void tovar::on\_poisk\_clicked()

{

char percent = '%';

QSqlQuery query;

QString querySearchString = QString("Название\_товара LIKE '%1%2' OR ID\_товара LIKE '%1%2'")

.arg(ui->poiskS->text()).arg(percent);

this->model->setFilter(querySearchString);

this->model->select();

this->ui->tableView->update();

}

Функция удаления реализована с помощью “QMessageBox” и “removeRow”.

Реализация удаления:

void tovar::on\_udalen\_clicked()

{

ui->statusbar->showMessage("Удаление позиции...");

QMessageBox::StandardButton reply = QMessageBox::question(this,"Удаление позиции.","Вы уверены, что хотите удалить позицию?");

if(reply == QMessageBox::Yes)

{

model->removeRow(row); //УДАЛЕНИЕ СТРОКИ ПО НО ЕЕ НОМЕРУ

}

else

qDebug("/q");

ui->statusbar->showMessage(" ");

}

# 3.2 Реализация приложения

Для создания программного инструмента были выбраны язык программирования C++ и фреймворк Qt creator.

Qt предоставляет набор полезных библиотек классов, а также специфическую модель разработки приложений, определенный каркас структур. Следуя принципам и соглашениям "хорошего стиля программирования C++/Qt", можно значительно снизить частоту отлавливаемых приложений ошибок, которые часто поражают программы на C++, написанные без библиотек Qt, таких как утечки памяти, необработанные исключения, неоткрытые файлы и несвободные дескрипторы объектов ресурсов [4].

Важным преимуществом Qt является хорошо продуманный, логичный и последовательный набор классов, обеспечивающий очень высокий уровень абстракции. Поэтому программистам, использующим Qt, требуется писать меньше кода. Сам код также более тонкий, простой, логичный и понятный.

C++ используется для создания всех видов программного обеспечения, от игр до операционных систем. Язык также широко используется для интенсивной обработки данных и научных вычислений. C++ предоставляет разработчикам мощный и гибкий инструмент для разработки программного обеспечения. Его способность писать эффективный и быстрый код делает его одним из самых популярных языков программирования в мире [5].

Язык позволяет создавать приложения и программы любой сложности, от простых консольных утилит до сложных игровых движков. Микроконтроллеры и системы реального времени также могут быть запрограммированы на этом языке. Язык используется для создания операционных систем, различных типов программного обеспечения и игровых движков и применяется во многих областях, включая разработку игр, научные исследования и разработку системного программного обеспечения.

Создание программных средств состояло из двух основных этапов: создание баз данных и создание программного обеспечения для их обработки. Первый этап подробно описан в предыдущем разделе.

Разработка программного обеспечения началась с написания удобной и работоспособной "оболочки". Сначала был реализован вывод выбранной пользователем таблицы, а затем написаны все функции, связанные с этой таблицей.

Каждую таблицу пользователь может масштабировать как ему захочется, но при этом пользователь не может поменять расположения кнопок и столбцов таблицы. Также приложение возвращается в изначальный вид при его обновлении, это сделано для того, чтобы предотвратить случайное изменение окна в неблагоприятный для работы вид. Специально для удобства были расширены столбцы таблицы, чтобы они были более читабельны и одинаковых размеров.

Реализация расширения столбцов таблицы:

ui->tableView->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Stretch);

ui->tableView->horizontalHeader()->setSectionResizeMode(1, QHeaderView::Stretch);

Таким образом, написав программу, мы смогли достичь своей цели-создать простое в использовании приложение со всеми возможностями CRUD-приложения.

# 4 Тестирование приложения

Для проверки исправности приложения пройдём полное тестирование:

1. Запустим приложение. Убедимся, что внешний вид окна соответствует показанному на скриншоте ниже.

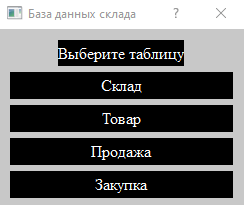


Рисунок 5 – Главное окно приложения

1. Выберем каждую из таблиц, сравним её вид с представленными ниже.

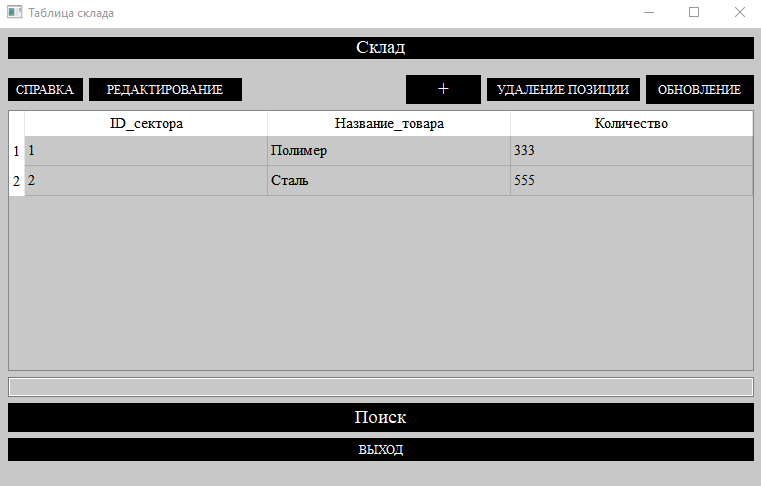


Рисунок 6 – Таблица «Склад»

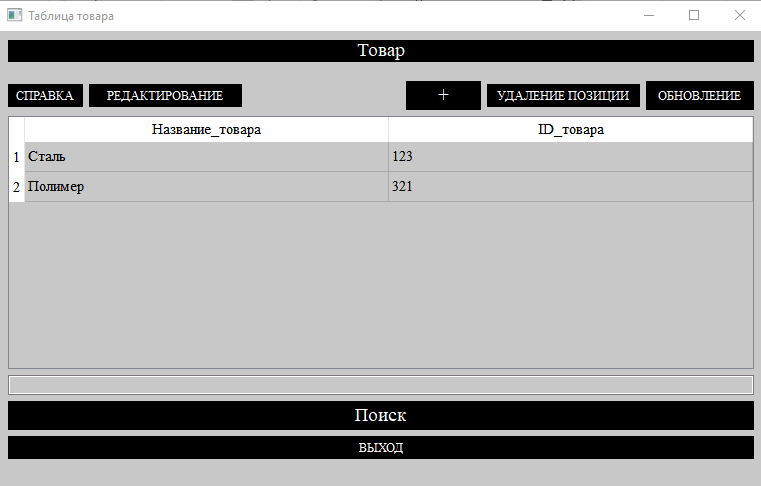


Рисунок 7 – Таблица «Товар»

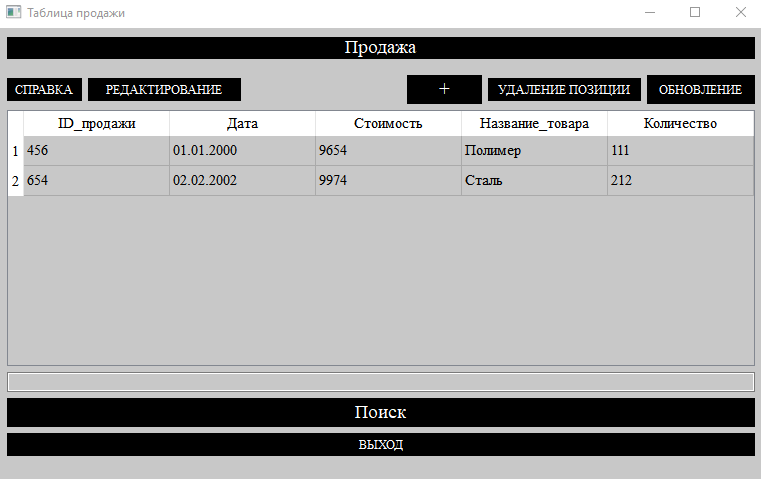


Рисунок 8 – Таблица «Продажа»

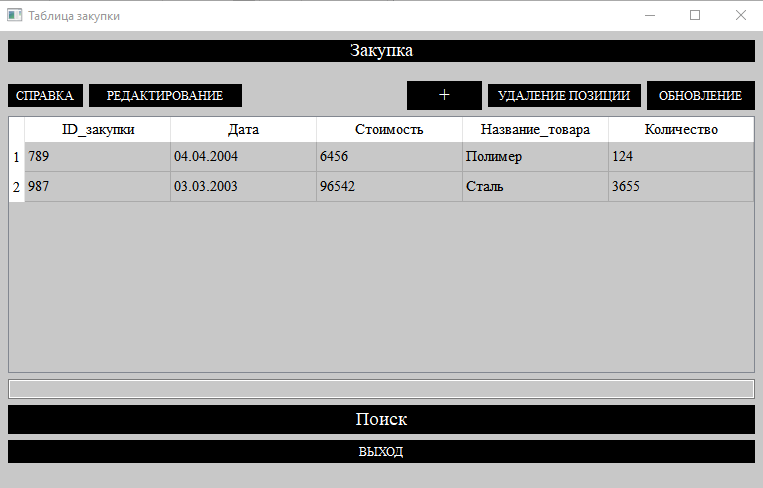


Рисунок 9 – Таблица «Закупка»

1. Проверим у каждой таблицы исправность работы всех функций: добавим строку, удалим выбранную строку (проверим вывод предупреждающего окна), редактируем данные в таблице, произведем поиск в каждой таблице. После каждой процедуры произведем обновление таблицы, сверим выведенные данные.
2. Проверим работоспособность кнопки «Справка». По нажатию должно выводиться окно, в котором написана справка для пользователя.
3. Закроем приложение с помощью кнопки «Выход». Убедимся, что приложение закрылось полностью и перед закрытием вывело предупреждающее окно.
4. Заново запустим приложение и убедимся в том, что все изменения были сохранены, никакие данные не потеряны, и база данных работает исправно.

# ЗАКЛЮЧЕНИЕ

В процессе написания данной работы были применены базовые навыки программирования, алгоритмизации. Были изучены новые библиотеки, структуры языка программирования C++.

Результатом данной работы является написание программы, помогающей директору предприятия вести модульный учет склада. Приложение позволяет хранить, удалять, редактировать информацию, а также производить поиск и сортировку по каждой таблице. Работа реализована с использованием полученных знаний в ходе изучение теоретических основ программирования и создания баз данных.

Ссылка на проект на github:

# Список литературы

1. Страуструп. Б. Язык программирования C++. Специальное издание. — М.: БиномПресс, 2007. — 1104 с.
2. Вайсфельд М. Объектно-ориентированное мышление, 4-е издание. — М.: Санкт-Петербург, 2014. — 304 с.
3. Сиддхартха Рао. Освой самостоятельно C++ за 21 день, 7-е издание — М.: «Вильямс», 2013. — 688 с.
4. Земсков Ю.В. Qt 4 на примерах. — СПб.: БХВ-Петербург, 2008. — 608 с.
5. Ж. Бланшет, М. Саммерфилд. Qt 4. Программирование GUI на С++, 2-е издание — 2008.