Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Воронежский государственный лесотехнический университет

имени Г.Ф. Морозова»

Кафедра Вычислительной техники и информационных систем

(название кафедры)

**Пояснительная записка**

КУРСОВОЙ РАБОТЫ

(вид работы)

Проектирование реляционных баз данных на платформе QT Creator

(тема)

09.03.02 Информационные системы и технологии

(код и наименование направления подготовки)

По дисциплине: Технологии программирования

Студент группы ИС2–191–ОБ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухарский А.В.

(номер группы) (подпись) (инициалы и фамилия)

Руководитель к.т.н, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ Юров А.Н.

(ученая степень, ученое звание) (подпись) (инициалы и фамилия)

Воронеж 2021 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Воронежский государственный лесотехнический университет

имени Г.Ф. Морозова»

Кафедра вычислительной техники и информационных систем

**ЗАДАНИЕ**

на курсовую работу по дисциплине

«Технология программирования »

Студенту 2 курса группы ИС2-191-ОБ Сухарскому А.В.

(Фамилия И.О.)

09.03.02 Информационные системы и технологии

(код и наименование направления подготовки)

Срок представления к защите «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**Тема работы:** Проектирование реляционных баз данных на платформе QT Creator

**Исходные данные для проектирования:** для своей предметной области выполнить проектирование программного продукта для работы с базами данных.

**Перечень вопросов, подлежащих разработке:**

Введение.

Теоретический материал по заданной теме.

Проектирование интерфейса пользователя.

Реализация проектирования программного продукта.

Заключение.

Список литературы.

Руководитель канд. техн. наук \_\_\_\_\_\_\_\_\_\_ А.Н.Юров

Задание принял студент \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухарский А.В.

(подпись) (число, месяц, год) (инициалы и фамилия)

ОГЛАВЛЕНИЕ

[**Введение** 4](#_Toc73905922)

[**Цель работы** 5](#_Toc73905923)

[**1.Теоретическая часть** 6](#_Toc73905924)

[**1.1Уровень драйверов** 7](#_Toc73905925)

[**2.Идея реализации программного продукта** 9](#_Toc73905926)

[**3 Описание кода программы** 11](#_Toc73905927)

[**3.1 Mainwindow.h** 11](#_Toc73905928)

[**3.2 Mainwindow.cpp** 11](#_Toc73905929)

[**4 Описание работы** 15](#_Toc73905930)

[**программного продукта** 15](#_Toc73905931)

[**Заключение** 34](#_Toc73905932)

[**Список литературы** 35](#_Toc73905933)

[**Приложение 1 листинг программы .pro** 36](#_Toc73905934)

[**Приложение 2 листинг программы .h** 37](#_Toc73905935)

[**Приложение 3 листинг программы main.cpp** 38](#_Toc73905936)

[**Приложение 4 листинг программы maimwindow.cpp** 39](#_Toc73905937)

[**Приложение 5 Создание в дизайнере рабочего пространства** 41](#_Toc73905938)

[**Схема разработки проектного решения** 42](#_Toc73905939)

[**Алгоритмы, структуры для описания и хранения данных** 43](#_Toc73905940)

# **Введение**

База данных представляет собой определенный набор данных, которые, как правило, связаны объединяющим признаком либо свойством (или несколькими). Обилие различных данных, которые могут быть помещены в единую базу, ведет к множеству вариаций того, что может быть записано: личные данные пользователей, записи, даты, заказы и так далее.

В первую очередь это удобно тем, что информацию можно быстро заносить в базу данных и так же быстро ее извлекать при необходимости. Если на заре развития программного обеспечения все необходимые данные нужно было прописывать в коде, то теперь такая необходимость отсутствует – нужная информация может быть запрошена из базы данных при помощи скриптов. Специальные алгоритмы хранения и поиска информации, которые используются в базах данных, позволяют находить нужные сведения буквально за доли секунд.

Немаловажной является и взаимосвязь информации в базе данных: изменение одной строчки может привести к значительным изменениям других строк. Работать с данными таким образом гораздо проще и быстрее, чем если бы изменения касались только одного места в базе данных.

Для управления базами данных применяется особый язык программирования – SQL. Сокращение расшифровывается как “Structured query language”. Команды, которые используются в SQL, делятся на те, которые манипулируют данными, те, которые определяют данные, и те, которые управляют данными. Осуществлять все запросы и администрировать БД помогает система управления базами данных (СУБД) – MS SQL Server. Данная СУБД была выбрана для выполнения курсовой работы из-за лёгкости настройки и эксплуатации.

**Цель работы**

Первоочередной задачей ставится создание интуитивно понятного интерфейса, простотой реализацией, надежностью в плане сбоев. Так же миниатюрность программного окна, что является частой проблемой при использовании программного продукта на мониторах небольшого размера, а также ноутбуках.

Задачи решаемые программным продуктом следующие:

- создание удобного интерфейса;

- возможность вводить и редактировать данные;

- простота взаимодействия с программным продуктом;

- программный продукт должен охватывать большую часть сферы, где применяются радиокомпоненты;

Наличие возможности дальнейшего развития программного продукта, позволяющее стать более рентабельным по сравнению с другими решениями, что позволит упростить большинства задач как пользователей, так и для директоров магазинов, ремонтных мастерских и прочего.

# **1.Теоретическая часть**

Для начала работы необходимо добавить сервис использования БД который позволит работать с базами данных при помощи встроенного в QT сервиса как — QtSql. Для подключения данного сервиса в файл с разрешением (Название проекта).pro добавить следующую строку: QT += sql.

Для того, что бы использовать , модуль QtSql, а для работы с ее классами, нужно включать одноименный заголовок. Благодаря этой строке мы получим полный функционал встроенного в QT модуля QtSql который позволит спроектировать в дальнейшем базу данных. Подключаем одноименный заголовок такой строкой : #include <QS>

**Qt** представляет модуль поддержки баз данных, классы которого разделены натри уровня: уровень драйверов, программный и пользовательский. Прежде чем начать работу с базой данных, необходимо соединиться с ней, активизировав драйвер. Запросы оформляются в виде строки. Для высылки запросов используется класс **QSqlQuery**.

Классы уровней драйверов принято разделять на три уровня:

* Уровень драйверов.
* Программный уровень.
* Уровень пользовательского интерфейса.

К первому уровню относятся классы для получения данных на физическом уровне. Это такие классы, как: **QSqlDriver**, **QSqlDriverCreator**<T\*>, **QSqlDriverCreatorBase**, **QSqlDriverPlugin** и **QSqlResult**.

Классы второго уровня предоставляют программный интерфейс для обращения к базе данных. К классам этого уровня относятся следующие классы: **QSqlDatabase**, **QSqlQuery**, **QSqlError**, **QSqlField**, **QSqlIndex** и **QSqlRecord**.

Третий уровень предоставляет модели для отображения результатов запросов в представлениях интервью. К этим классам относятся: **QSqlQueryModel**, **QSqlTableModel** и **QSqlRelationalTableModel**.

Классы первого уровня вам не придется использовать, если вы не собираетесь писать свой собственный драйвер для менеджера базы данных. В большинстве случаев все ограничивается использованием конкретной СУБД (система управления базами данных), поддерживаемой Qt.

Разберем за что отвечает каждый из классов на различных уровнях драйверов для работы со встроенной базой данный в виде SQLlite благодаря которой производится проектирование программного продукта.

**1.1Уровень драйверов**

К уровню драйверов относятся классы для получения данных на физическом уровне, такие, как:

QSqlDriver

QSqlDriverCreator<T\*>

QSqlDriverCreatorBase,

QSqlDriverPlugin

QSqlResult

QSqlDriver является абстрактным базовым классом, предназначенный для доступа к специфичным БД. Важно, что класс не должен быть использован «прямо», взамен нужно/можно воспользоваться QSqlDatabase. Хотя, если вы хотите создать свой собственный драйвер SQL, то можете наследовать от QSqlDriver и реализовать чисто виртуальные, и нужные вам виртуальные функции.

QSqlDriverCreator — шаблонный класс, предоставляющий фабрику SQL драйвера для специфичного типа драйвера. Шаблонный параметр должен быть подклассом QSqlDriver.

QSqlCreatorBase — базовый класс для фабрик SQL драйверов, чтобы возвращать экземпляр специфичного поскласса класса QSqlDriver, который вы хотите предоставить, нужно «перефразировать» метод createObject().

QSqlDatabase несет ответственность за загрузку и управление плагинов драйверов баз данных. Когда база данных добавлена (это делается функцией QSqlDatabase::addDatabase()), необходимый плагин драйвера загружается (используя QSqlDriverPlugin). QSqlDriverPlugin предоставляет собой абстрактный базовый класс дляпользовательских QSqlDriver плагинов.  
QSqlResult сам говорит о себе (как и все Qt-шные классы), этот класс предоставляет абстрактный интерфейс для доступа к данным специфичных БД. С практической точки зрения мы будем использовать QSqlQuery вместо QSqlResult, поскольку QSqlQuery предоставляет обертку («обобщенную») для БД-специфичных реализации QSqlResult.

# **2.Идея реализации программного продукта**

Для реализации поставленной задачи было необходимо разработать приложение – базу данных, с возможностью ее редактирования и изменением параметров под каждого пользователя.

В данный момент существует множество подобных программных решений, которые позволяют работать с базой данных радиокомпонентов, которые можно применять в магазинах и крупных торговых сетях. Зачастую такие программы перегружены различными функциями, которые в данный момент времени не актуальны пользователю, и их необходимость отпадает.

Зачастую такие базы данных подходят лишь для магазинов, в частности, предпринимателей, но необходимо не забывать о различных мастерских, различных мастеров и простых любителей что – то конструировать своими руками.

Для примера была взята программа «БД = радиокомпоненты=», которая наиболее подходящая для реализации базы данных по компонентам. Данная программа была найдена на одноименном форуме[[1].](#_Список_литературы)

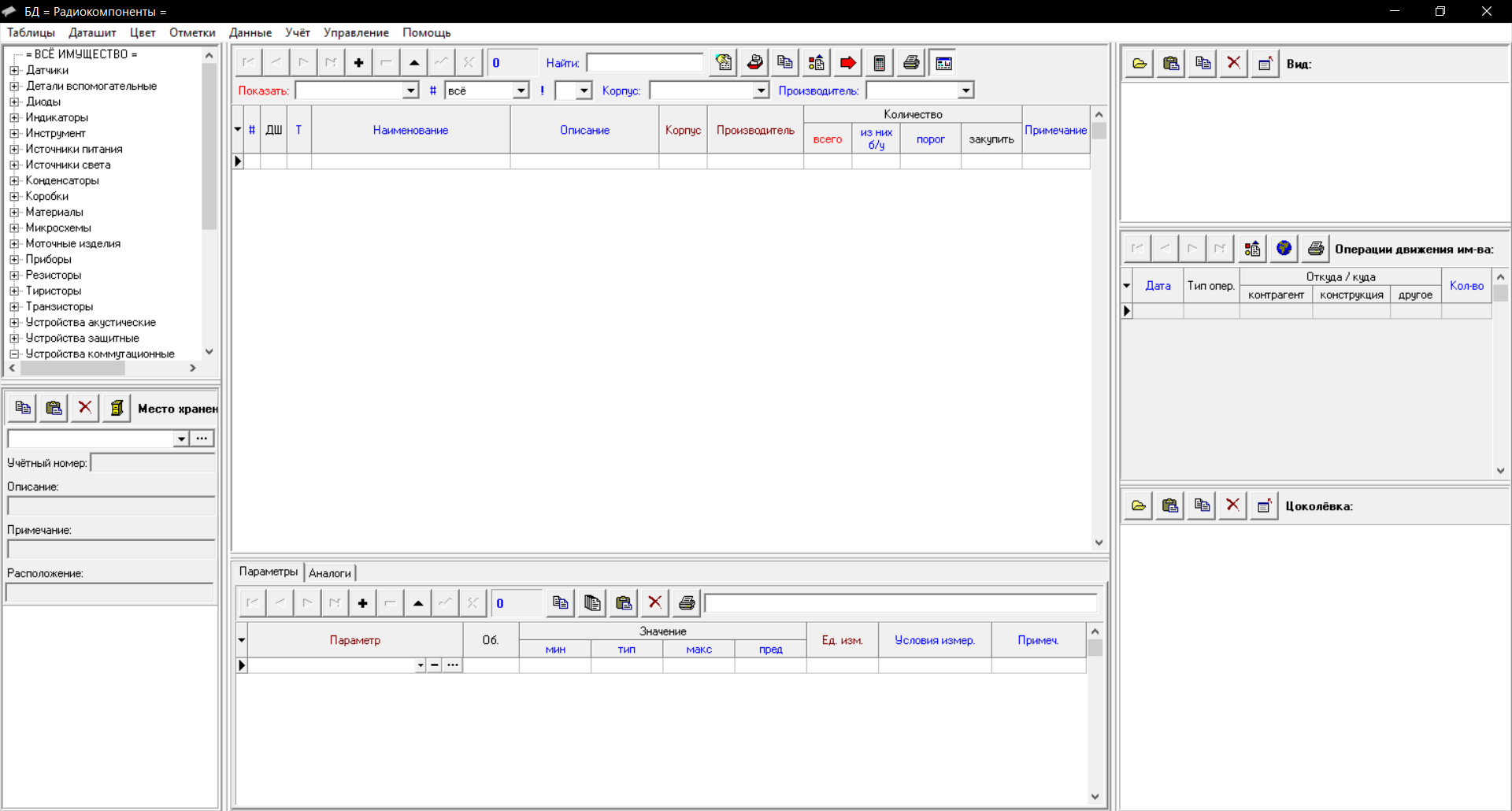


Рисунок 1 – программа «БД = радиокомпоненты =», аналог разработки

Недостаток данной является большая загруженность интерфейса, различные функции, которые не будут применятся на практике. Необходимо помнить, что пользователи бывают разные, и по этой причине пользователь не сможет реализовать потенциал программы.

Так же к недостаткам можно отнести громоздкость таких программ в плане большого размера окон, что не всегда удобно пользователю на мониторах не большого размера, в частности на ноутбуках.

Анализируя рынок представленных программных продуктах, можно сделать вывод что простых, но в то же время удобных программных решений нет. Таким образом можно сделать вывод, что реализация собственной разработки внесет вклад в эту сферу, поскольку, с такой программы можно начать обучать персонал как с работой с компьютером, так и взаимодействия с базой данных.

По этой причине было принято решение разработать свой программный продукт, реализующий простую базу данных, в которой реализована самый необходимый функционал. Добавление, поиск и редактирование записей, ток же добавлены функции демонстрации компонента и вывод всевозможных аналогов. Такой программный продукт имеет не большую информационную нагрузку и удобный интерфейс.

# **3 Описание кода программы**

**3.1 Mainwindow.h**

В файле mainwindow.h имеем следующие структуры кода :

Перовое это приватные слоты при создании на форме различных объектов

В моем случае такими объектами являются пять кнопок и Nable View cliced как представлено ниже

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_tableView\_clicked**(const QModelIndex &index);

void **on\_pushButton\_3\_clicked**();

void **on\_pushButton\_4\_clicked**();

void **on\_pushButton\_5\_clicked**();

В инкапсуляции класса были объявлены экземпляры класса для работы со встроенным в QT SQLlite для создания баз данных и работы с ними добавлена переменная row для взаимодействия со строками при удаденни строк из базы данных. Ниже приведены записи класса private:

private:

Ui::MainWindow \*ui;

QSqlDatabase db;

QSqlQuery \*query;

QSqlTableModel \* model;

int row;

QString fileName;

**3.2 Mainwindow.cpp**

Для реализации проекта подключим все необходимы библиотеки для работы с базами данных для этого в головном файле проекта с расширением .pro пропишем строку : QT += core gui sql, которая позволяет работать с базами данных.

Подключаем библиотеку классов для их работы, остановимся на каждом подробнее:

#include "mainwindow.h" – заголовочный файл основного окна приложения;

#include "ui\_mainwindow.h" – формочка основного окна приложения;

#include <QFileDialog> - позволяет пользователю перемещаться по иерархическим уровням файловой системы для выбора одного или нескольких файлов;

#include <QMessageBox> - предоставляет модельное диалоговое окно для информирования пользователя или для того, чтобы задать пользователю вопрос и получить ответ;

#include <QPixmap>- это один из виджетов, использующихся для работы с изображениями.

Далее настроим настроим подключение к базе данных и в метод add database передаем драйвер SQLlite

А в setDatabaseName мы прописываем путь к файлам, в нашем случае файлы базы данных будут распологаться в файле проекта

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

db.setDatabaseName("./testQT.db");

qDebug нам необходим для просмотра в нижней консоли как работает наша программа и где мы можем отследить те или иные ошибки, что помогает при проектировании различных программных продуктов.

if (db.open()){

qDebug ("open");

}

else

{

qDebug("Close");

}

При помощи команды query создадим нашу базу данных а так же таблицу со столбцами для внесения в них данных.

query = new QSqlQuery(db);

в созданной базе данных необходимо создать таблицу в которой будет осуществлятся работа пользователя, таким образом был прописан конструктор базы данных.

query->exec("CREATE TABLE Tranzistors (NAME TEXT, TYPE TEXT, PRICE INT, QUANTITY INT);");

model = new QSqlTableModel(this,db);

model->*setTable*("Tranzistors");

model->*select*();

Пропишем в кнопку добавление новой строки в базу данных при помощи функции insertRow

void MainWindow::**on\_pushButton\_clicked**()

{

model->insertRow(model->*rowCount*());

}

Пропишем во второй кнопке функцию удаления строки при помощи процедуры removeRow

void MainWindow::**on\_pushButton\_2\_clicked**()

{

model->removeRow(row);

}

Пропишем процедуру выделения строки в сконструированной базе данных. Благодаря ей и происходит последующие удаление выбранной строки.

void MainWindow::**on\_tableView\_clicked**(const QModelIndex &index)

{

row = index.row();

}

Следующим этапом было внедение открытия аналогов транзисторов корторые находятся в папке пгде и остальной проект, это позволяет пользлвателю просматривать различные аналоги траннзисторов.

void MainWindow::**on\_pushButton\_3\_clicked**()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Open file"), "", tr("Text Files(\*.txt);"));

if (fileName != "") {

QFile file(fileName);

if (!file.*open*(QIODevice::ReadOnly)) {

QMessageBox::critical(this, tr("Error"), tr("Could not open file"));

return;

}

QTextStream in(*&file*);

ui->textBrowser->setPlainText(in.readAll());

}

}

Ту же функцию реализуем для открытия картинок из одноимённой папки picture для просмотра внешнего вида радиодетали.

void MainWindow::**on\_pushButton\_4\_clicked**()

{

{

//QString fileName = QCoreApplication::applicationDirPath()+"/5020004616\_0.jpg";

QString fileName = QFileDialog::getOpenFileName(this, tr("Open file"), "", tr("Picture file (\*.jpg);"));

QFile file(fileName);

// QPixmap in(&file);

// ui->label->setPixmap(QPixmap(in.readAll()));

QPixmap pix(fileName);

ui->label->setPixmap(pix);

ui->label->setPixmap(pix);

}}

Последней кнопкой реализуется поиск по имени компонета где мы обращаемся к столбцу с названием NAME где происходит поиск интересующего нас компонента.

void MainWindow::**on\_pushButton\_5\_clicked**()

{

QSqlQueryModel \*model1=new QSqlQueryModel(this);

model1->setQuery(QString("select \* from Tranzistors where NAME = '%1'").arg(ui->lineEdit->text()));

ui->tableView->*setModel*(*model1*);

}

Создание формы в конструкторе приведено в приложении 5.

# **4 Описание работы**

# **программного продукта**

Запускаем программу и в открывшимся окне видим базу данных.

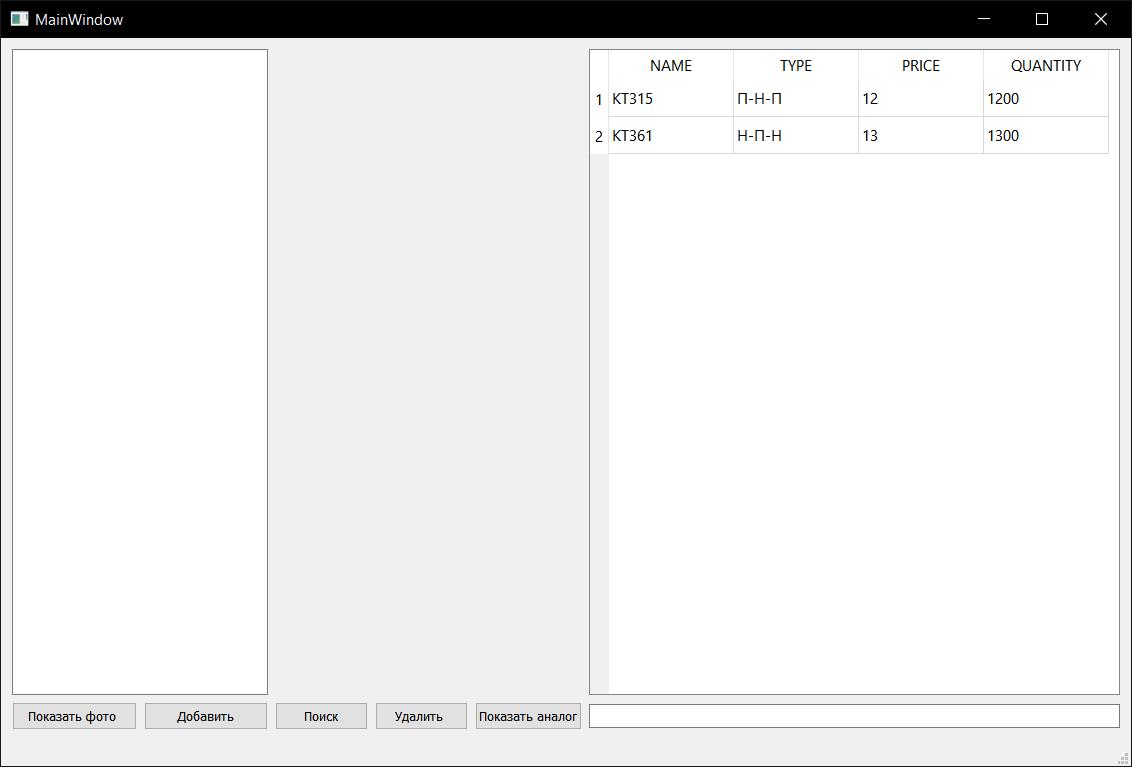


Рисунок 1 – открытая база данных с внесёнными в нее записями

Программа состоит из нескольких окон представляющие себя области для работы с базой данных. Остановимся на каждом окне по подробнее и разбкркм для чего оно необходим в данном программном продукте.

Первое что пользователь видит это окно с четырьмя столбцами которые поименованы следующим образом:

- NAME — это название компонента, в данном случае это название транзистора, рисунок 2;

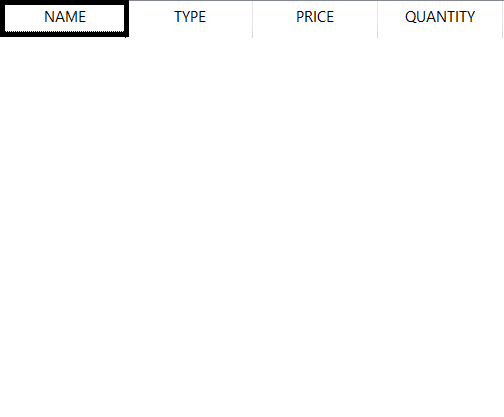


Рисунок 2 – имя столбца NAME

- TYPE – тип транзистора, в зависимости от его внутренней структуры, рисунок 3;



Рисунок 3 – имя столбца TYPE

- PRICE – цена транзистора за 1 штуку, рисунок 4;

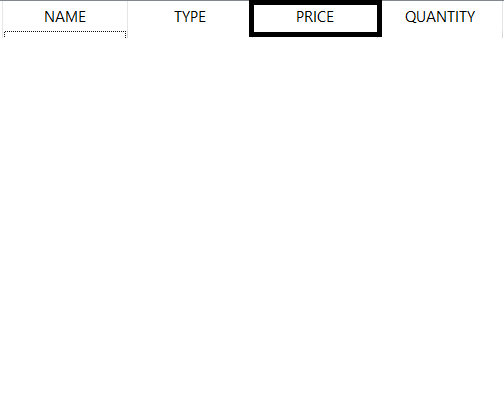


Рисунок 4 – имя столбца PRICE

- QUANTITY – количество элементов, находящихся в наличии, рисунок 5;

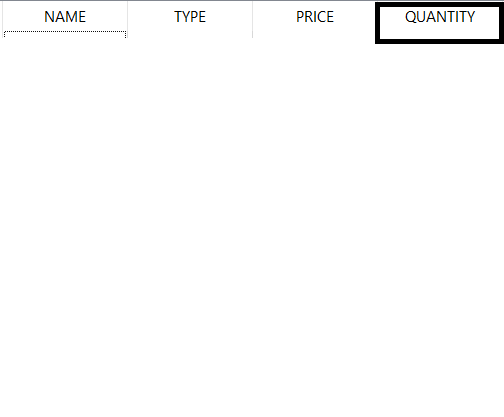


Рисунок 5 – имя столбца QUANTITY

Следующее окно представляет собой Text Browser, в котором будет ототбражатся открываемый пользователем текстовый документ. В этом текстовом документе, для каждого элемента, в нашем случает транзистора, будет отображатся список возможных аналогов для замены такогвого. Поскольку не всегда есть возможность применить тот или иной элемент по причине его отсутствия. Представлен Text Browser на рисунке 6.

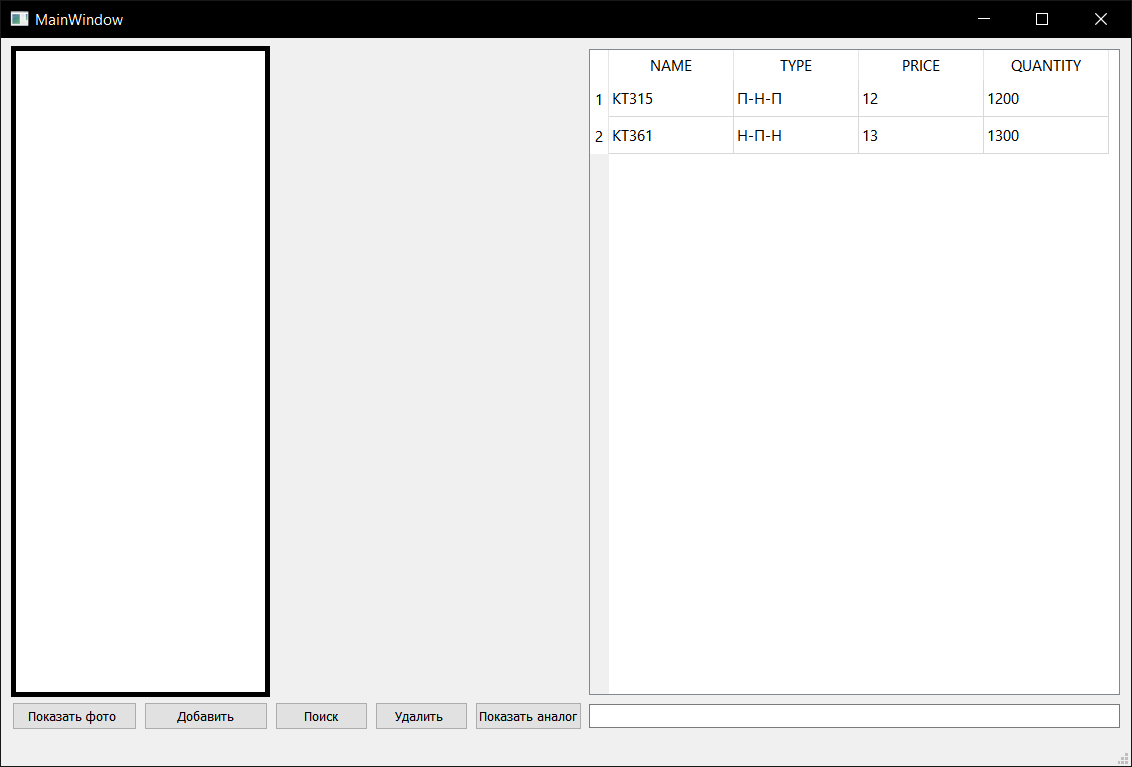


Рисунок 6 – элемент Text Browser

Следующий элемент это – text Label, в данном программном продукте он необходим для вывода картинки на форму, поскольку не всегда пользователь может найти компонент по названию, и для более быстрого поиска был применен вывод его картинки. На рисунке 7 показана область нахождения text Label.

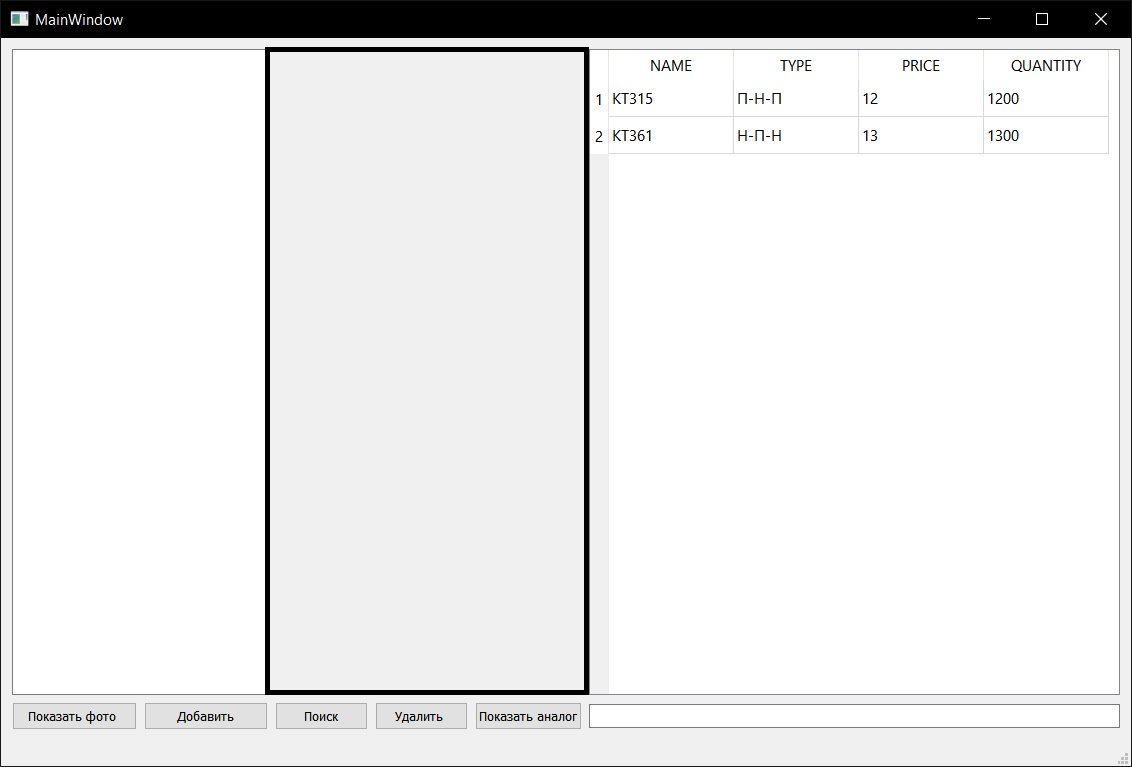


Рисунок 1 – Область нахождения text Label

В нижней части программы находится четыре кнопки, которыми пользователь может управлять программой. Каждая кнопка подписана, таким образом что бы пользователь понимал какая кнопка за что отвечает.

Кнопка «Добавить запись», благодаря этой кнопке пользователь может добавить новую запись в существующую базу данных, тем самым вносить те данный какие ему необходимо.

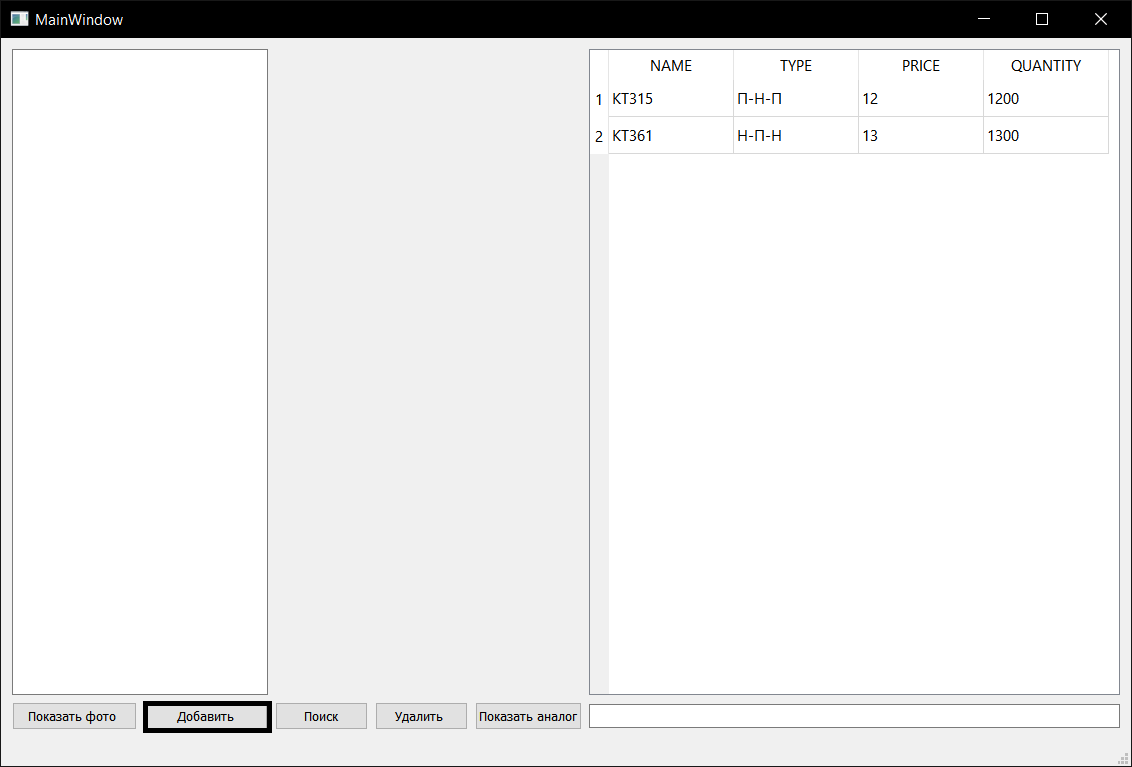


Рисунок 8 – кнопка «Добавить»

Кнопка «Удалить» позволяет пользователю программного продукта удалить не нужную ему запись из базы данных, что является необходимой функцией в подобных программах.

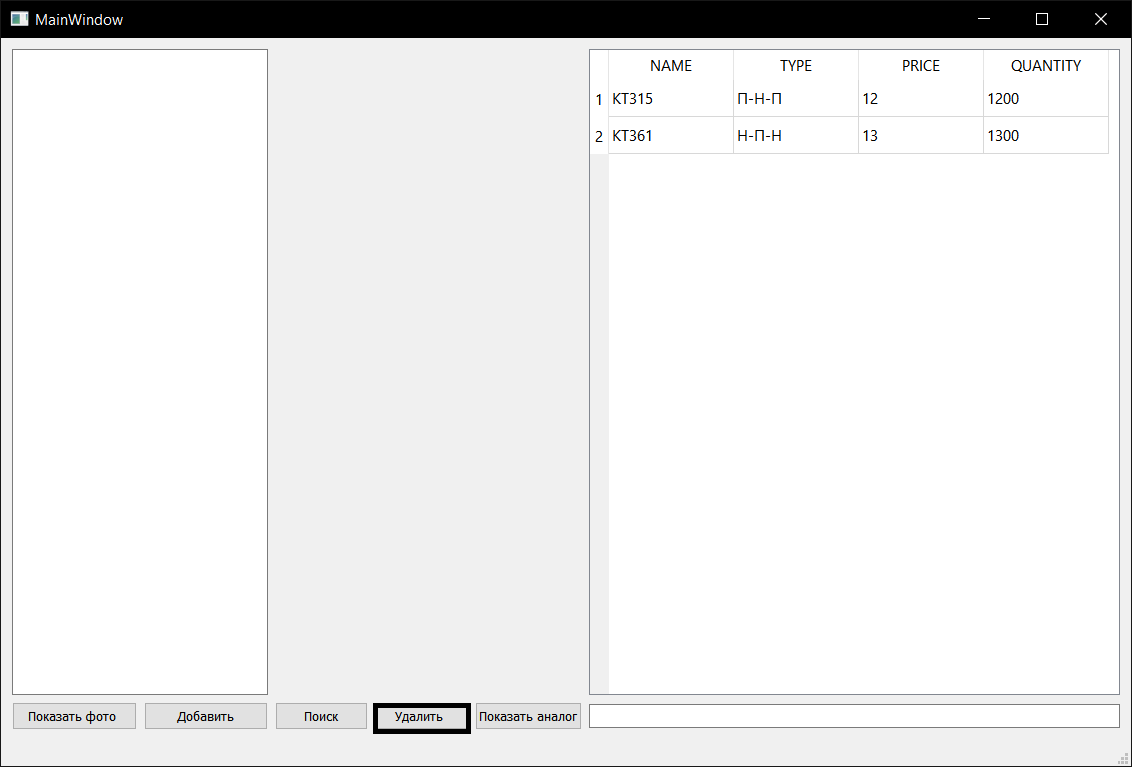


Рисунок 9 – кнопка «Удалить»

Следующая кнопка на форме «Показать фото». Этак кнопка отвечает за функции открытия фотографии элемента, который выбрал пользователь. Такое решение позволит вывести изображение транзистора, который необходим пользователю программным продуктом.

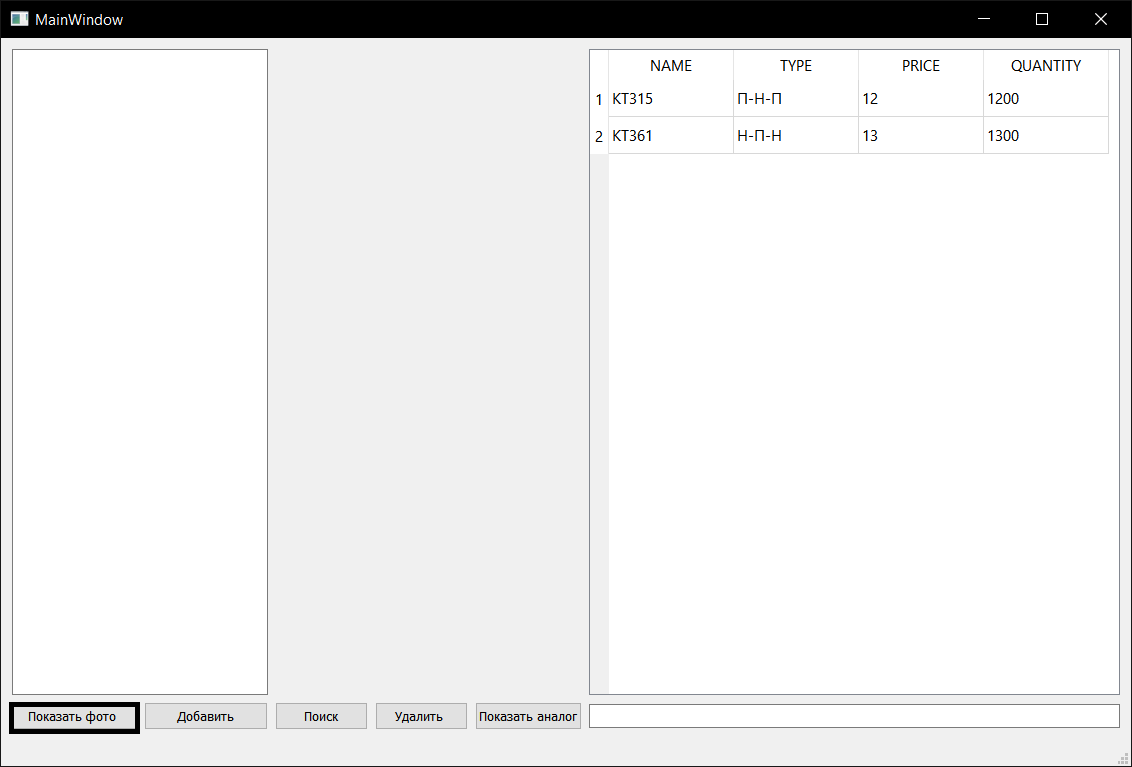


Рисунок 10 – кнопка «Показать фото»

Последняя кнопка на форме «Показать аналог». Эту кнопку можно считать такой же важной, как и кнопки «Добавить» и «удалить». Применение этой кнопки в большинстве случаев упростит работу пользователя. Чаще всего такой важнейшей функцией чаще всего не хватает в большинстве радиомагазинов, да и иметь такой программный продукт может подойти как в ремонтном центре, так и радиолюбителю.

Демонстрация работы программы состоит в следующем:

- запуск программы;

- возможности программного продукта;

- способы применения;

В открывшемся окне мы видим интерфейс пользователя программным продуктом. На котором расположены кнопки для работы с программой и виджеты для вывода информации.

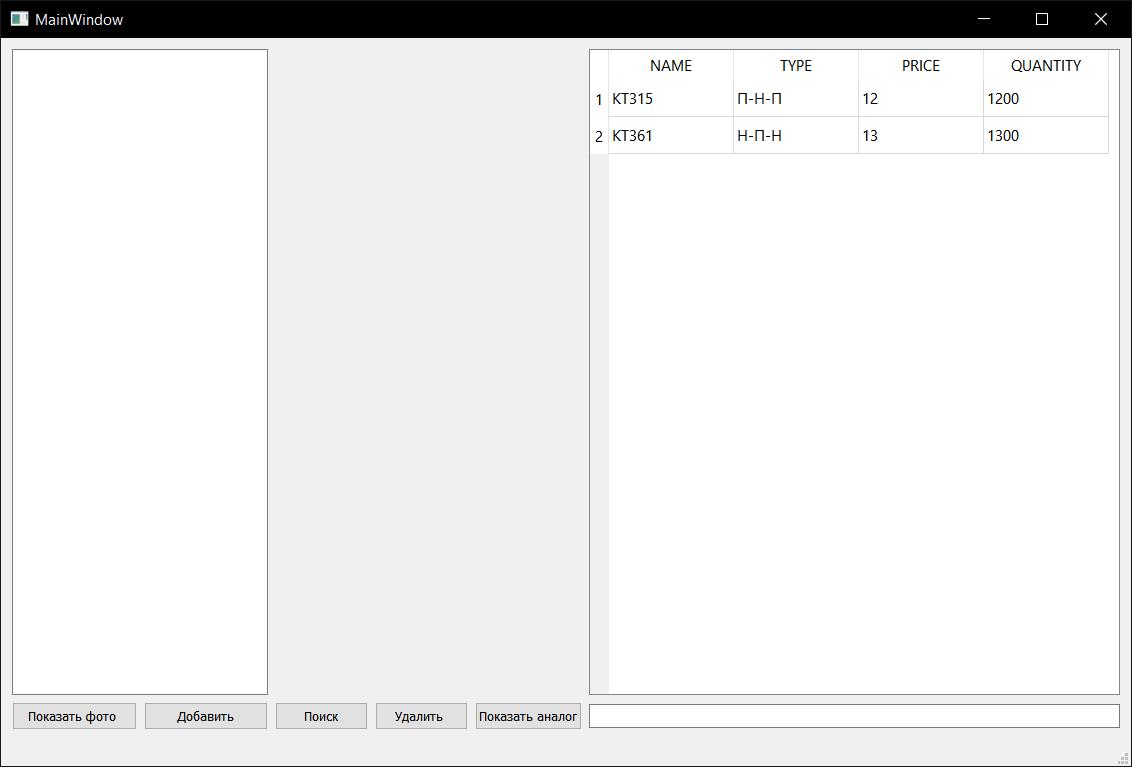


Рисунок 11 – рабочие окно программы

Для того что бы добавить новую запись, пользователю необходимо нажать на кнопку «добавить» в нижней части программы. После нажатия кнопки в окне с записями появится новая строка посеченная «\*». И выделяться соответствующим образом. Как показано на рисунке 12.

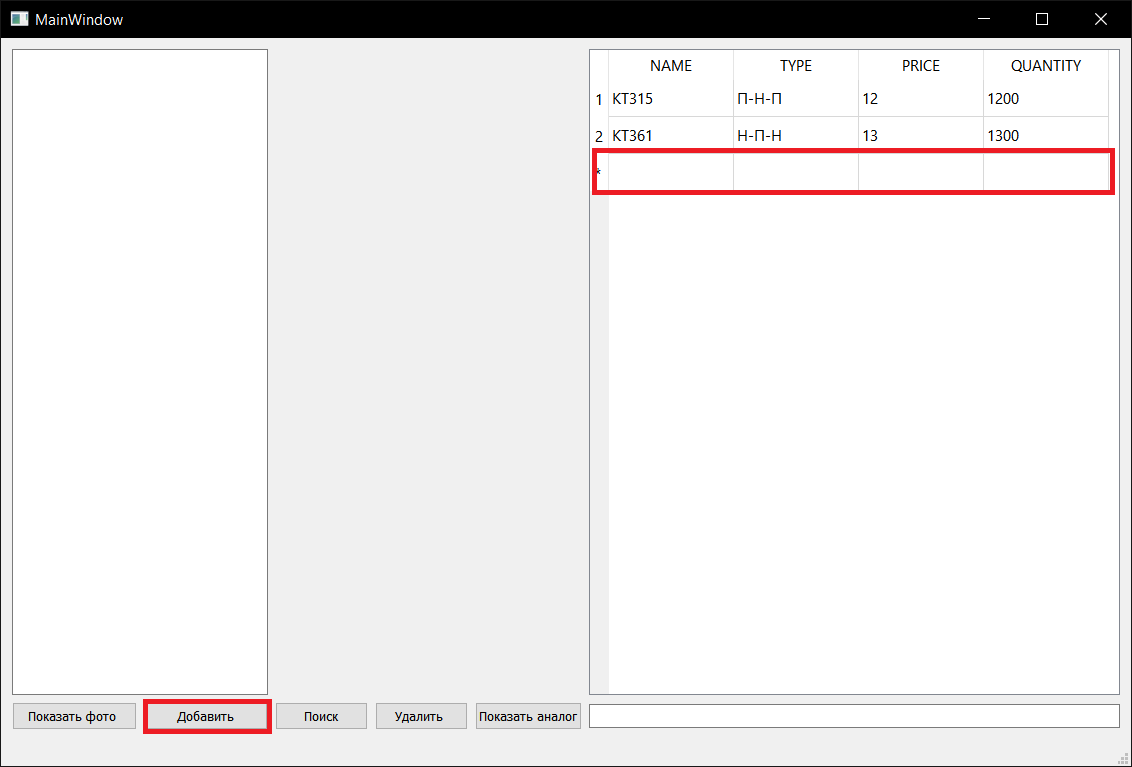


Рисунок 12 – нажатие кнопки «добавить»

Пользователь нажатием на первый столбец новой кнопки может вносить необходимый ему текст. В данном программном продукте это название транзистора. Как видно из рисунка 12, уже добавлены два транзистора, а именно: КТ315 КТ361. В добавленную строку внесем транзистор под названием КТ920. Внесенные данные будут показаны на рисунке 13. Стоит заметить, что столбцы PRICE, QUANTITY имеют численный параметр по этой причине мы можем вводить данные как с клавиатуры, так и при помощи стрелок увеличения или уменьшения.

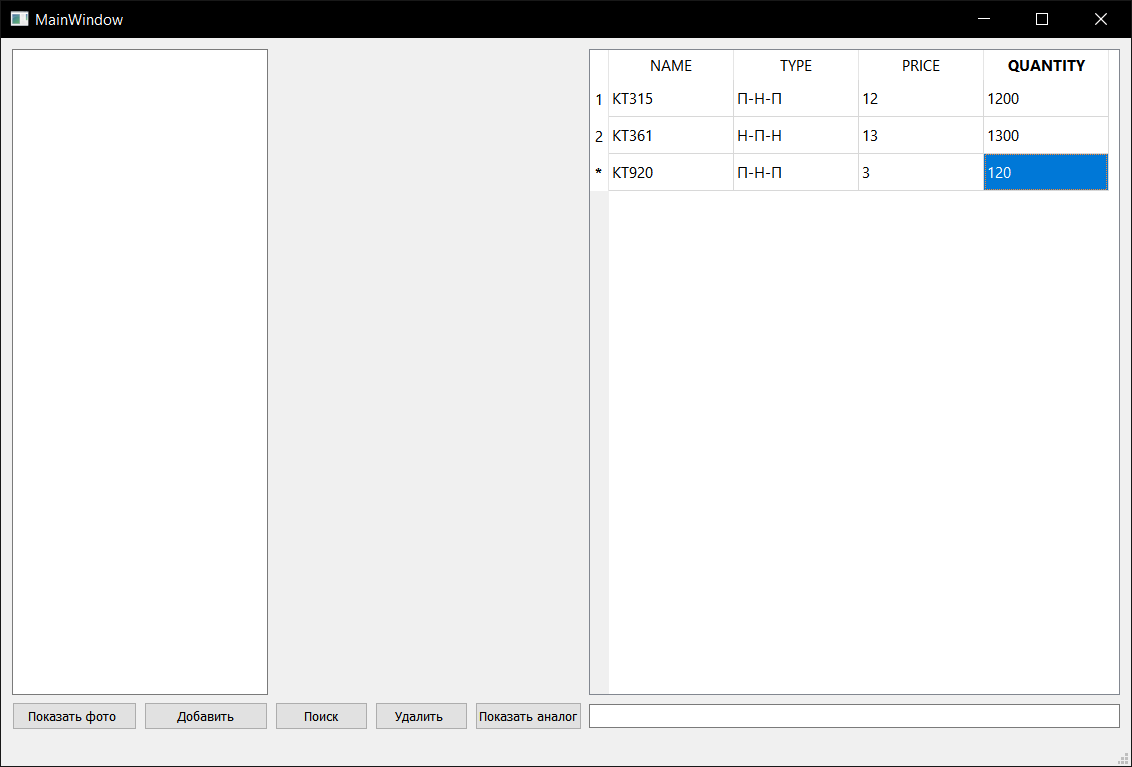


Рисунок 13 – добавленные данные в новую строку

Для того что бы окончательно добавить запись нам необходимо подтвердить внесенные данные. Это модно сделать при помощи нажатия кнопки Enter на клавиатуре, либо нажатием левого щелчка мыши на верхней записи. В результате чего «\*» изменится на номер строки в записях.

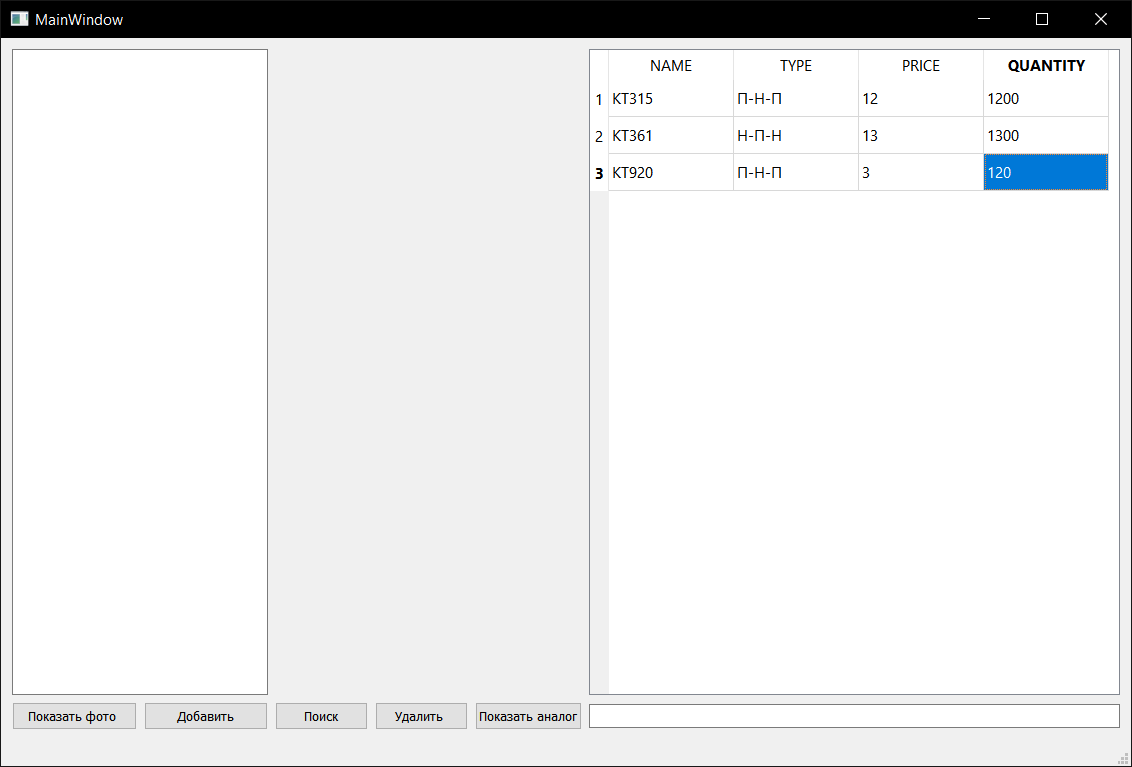


Рисунок 14 – новая подтвержденная запись

Соседняя конопка с добавлением записи имеет название «удалить». При помощи нее в данном программном продукте можно удалить не нужную строку : запись. Для того что бы удалить не нужную пользователю запись, ее необходимо выбрать путем нажатия на нее левой кнопки мыши. Выбранная строка будет отмечена путем подсвечивания номера строки (записи),а так же жирным выделением номера строки(записи)

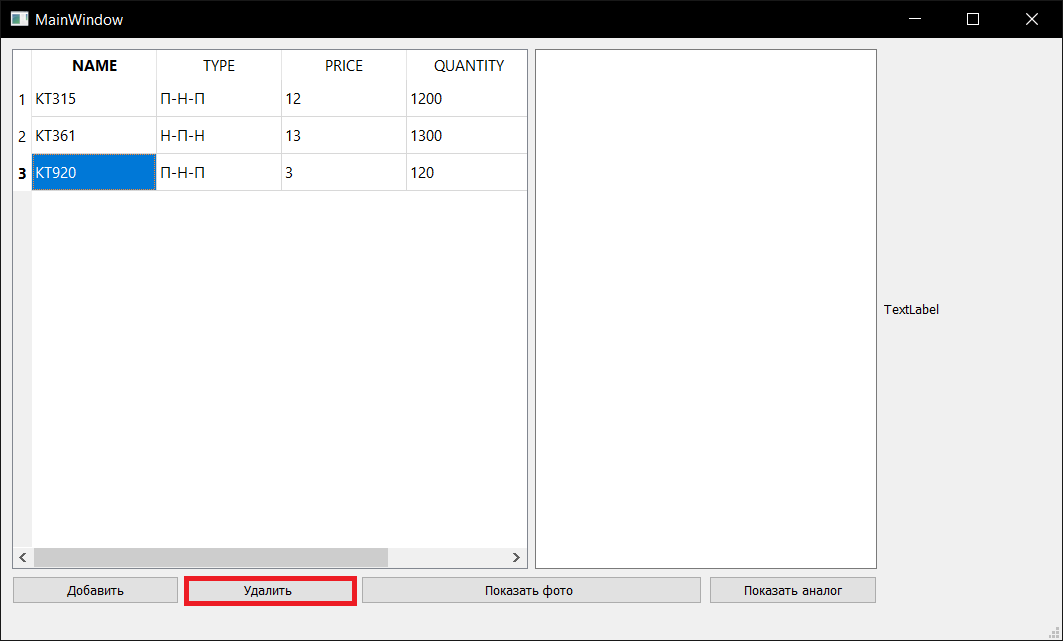


Рисунок 15 – выделенная запись и кнопка «удалить»

После нажатия кнопки «удалить» появится пустая запись, а в место номера строки будет высвечиваться восклицательный знак, после перезапуска программы он исчезнет и все строки подвинуться на верх и изменят свой номер, на более меньший. На рисунках 15 и 16 приведено окно программы после удаления и сортировки строк таблицы после перезапуска программы.

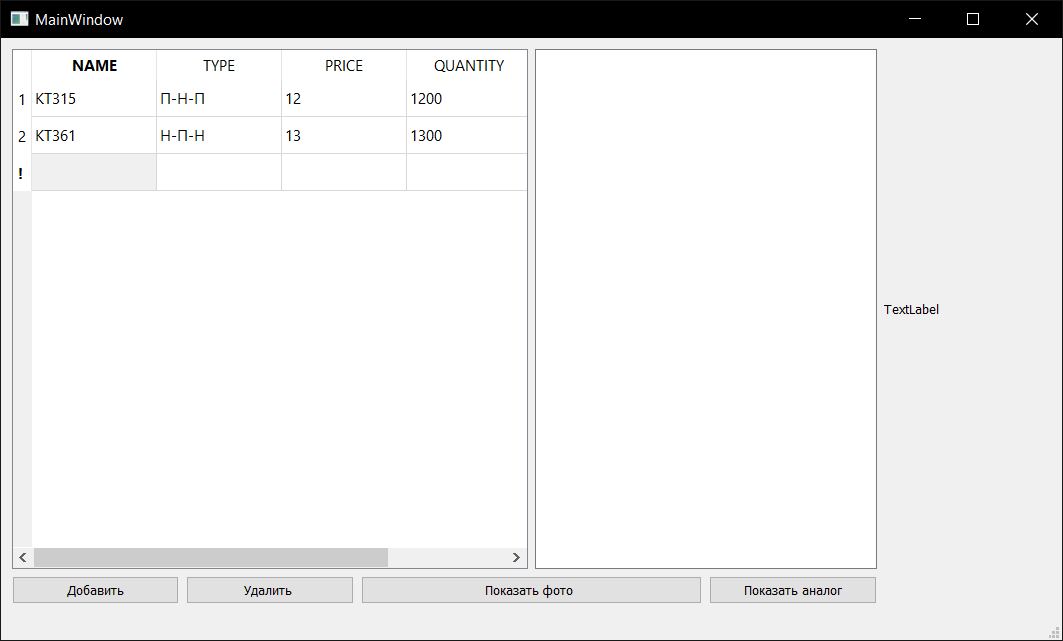


Рисунок 15 – выполнено удаление записи в таблице

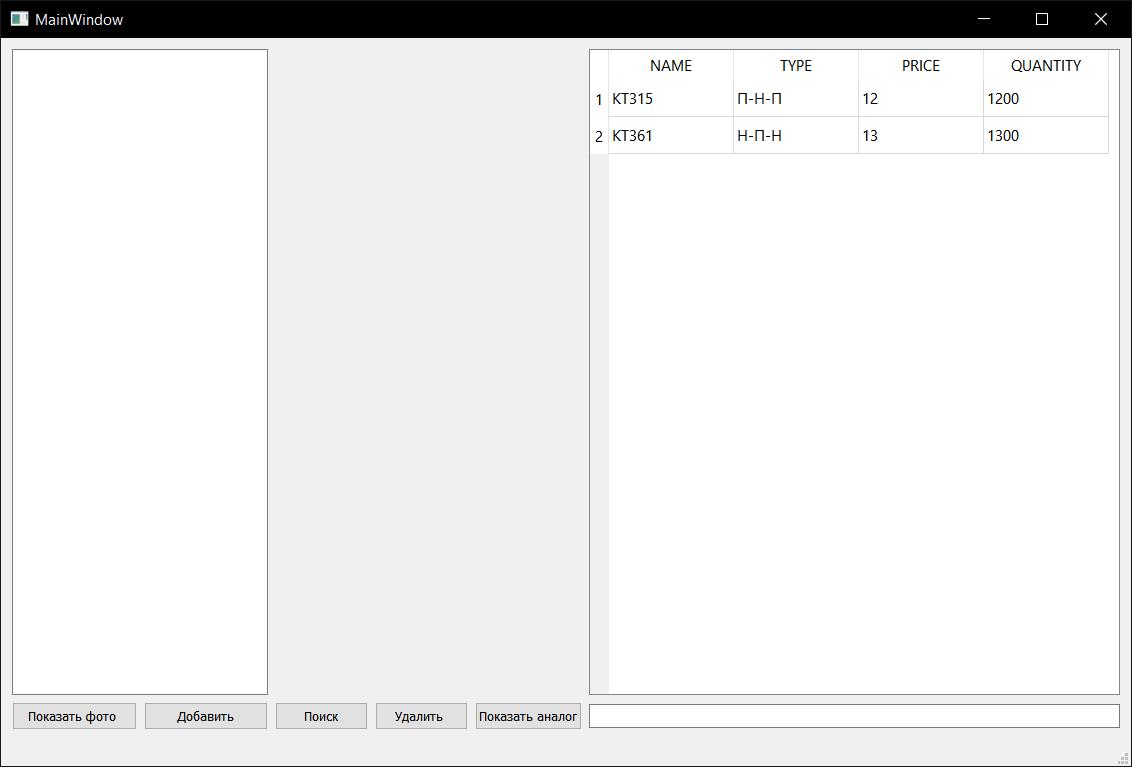


Рисунок 16 – выполнен перезапуск программы

Изменение записи в данной разработке весьма удобна, поскольку двойным нажатием на ячейку таблицы выделяется ранее введённое значение и появляется курсор, как показано на рисунке 17.

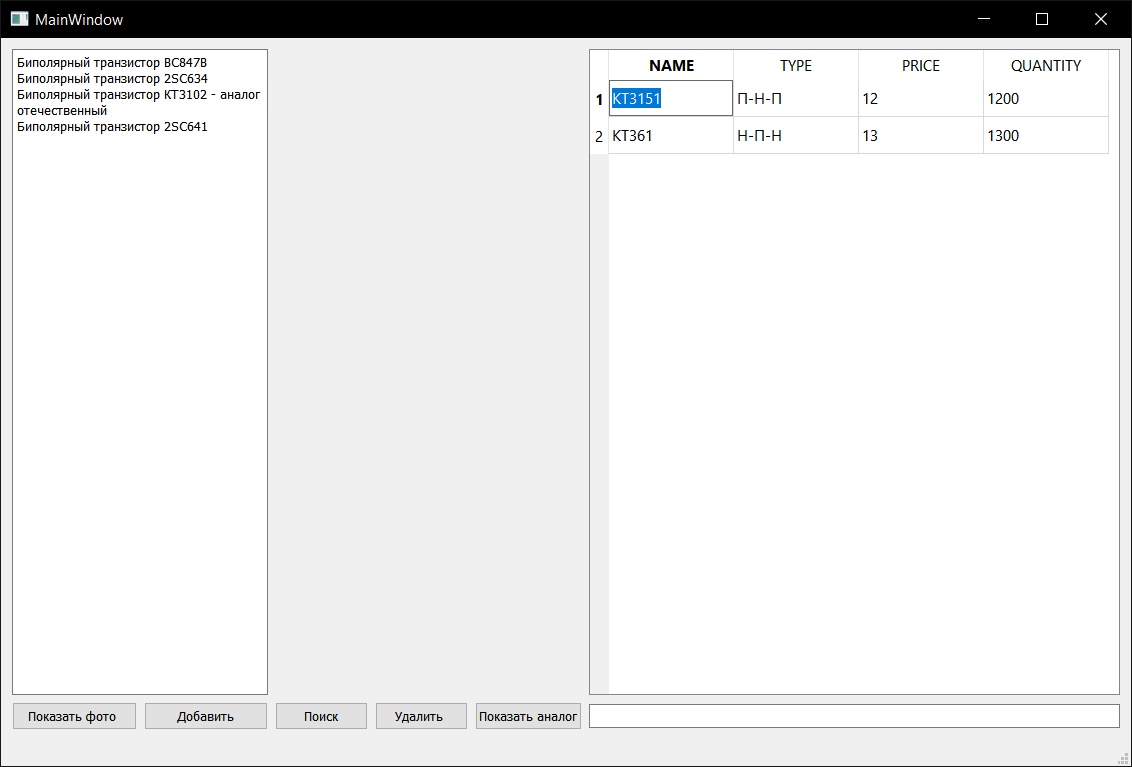


Рисунок 17 – выделенная ячейка в таблице

Следующим нажатием левой кнопки мыши снимается выделение и мы можем изменять данную запись, и щелчком в свободной области мы подтверждаем внесенное изменение в запись. Измененная запись показана на рисунке 18.

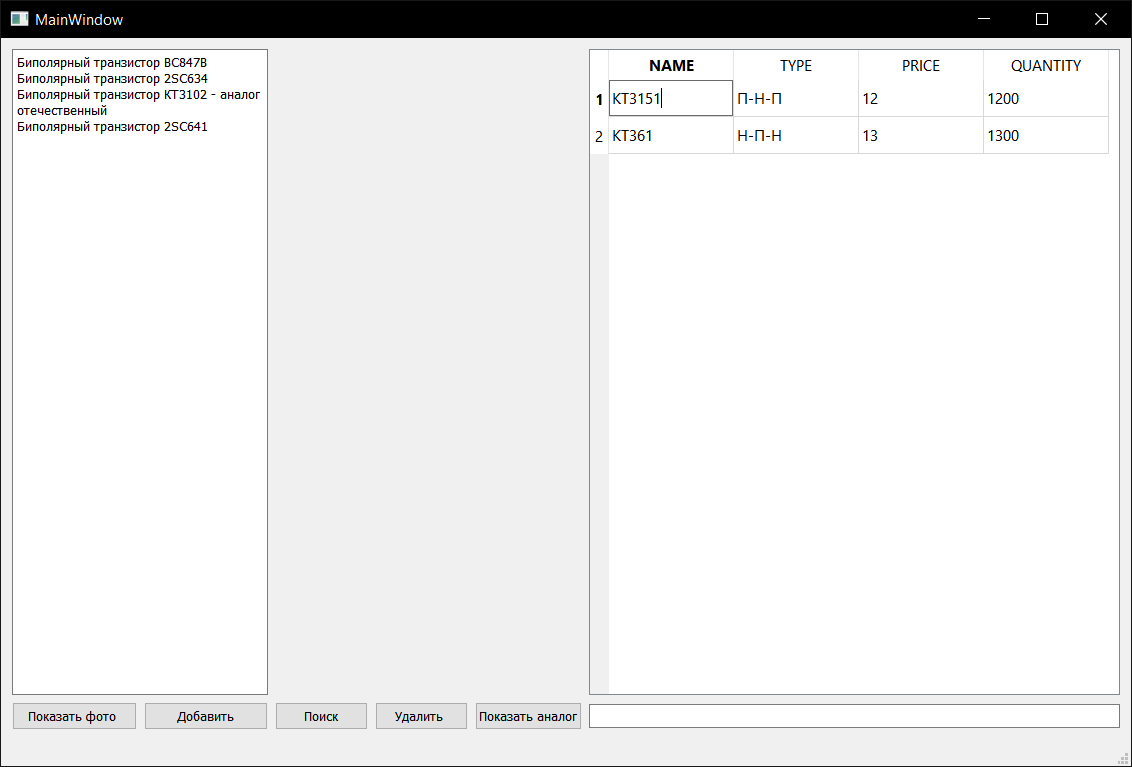


Рисунок 18 – внесенные изменения в таблицу

Кнопка показать фото работает следующим образом. При нажатии на данную кнопку открывается окно к расположению файлов к проекту как показано на рисунке 19.

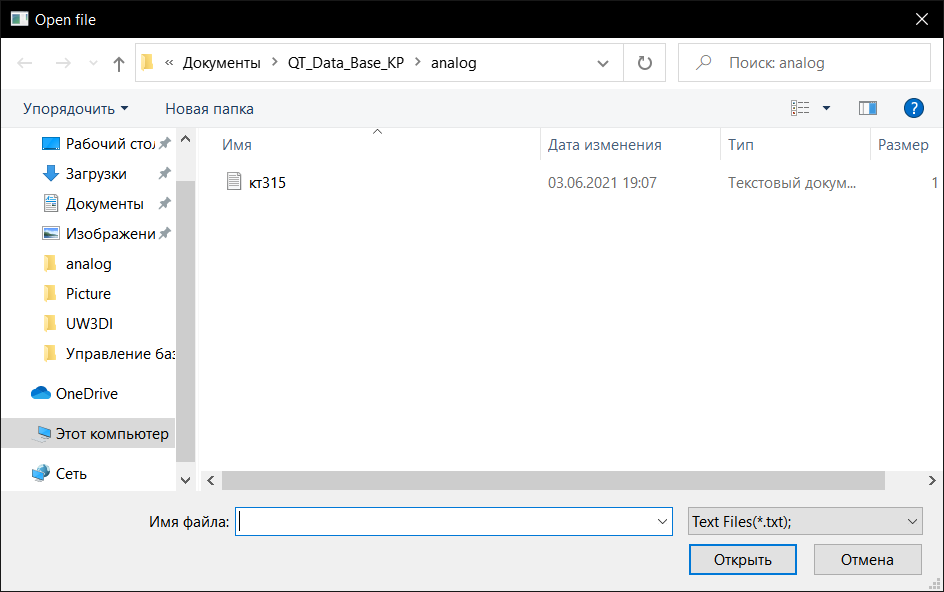


Рисунок 19 – папка с проектом с двумя папками для дополнительной информации

Открываем папку «analog», в которой хранятся файлы в формате .txt , с внесенными в них аналогами транзисторов в данной базе данных. В папке выбираем файл с названием кт315. И нажимаем кнопку открыть. Проделанная работа представлена на рисунках 20 и 21 .

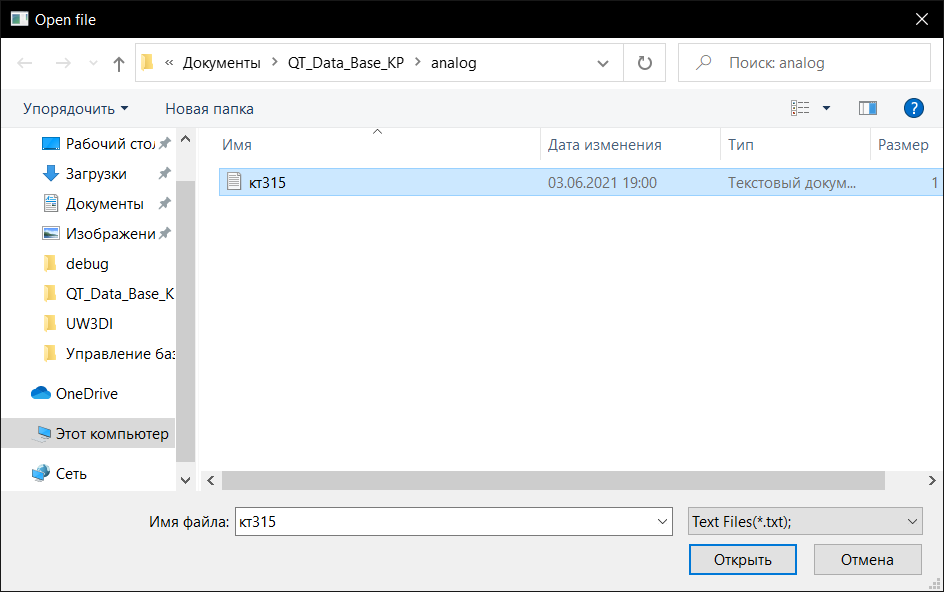


Рисунок 20 – выбранный файл с аналогами

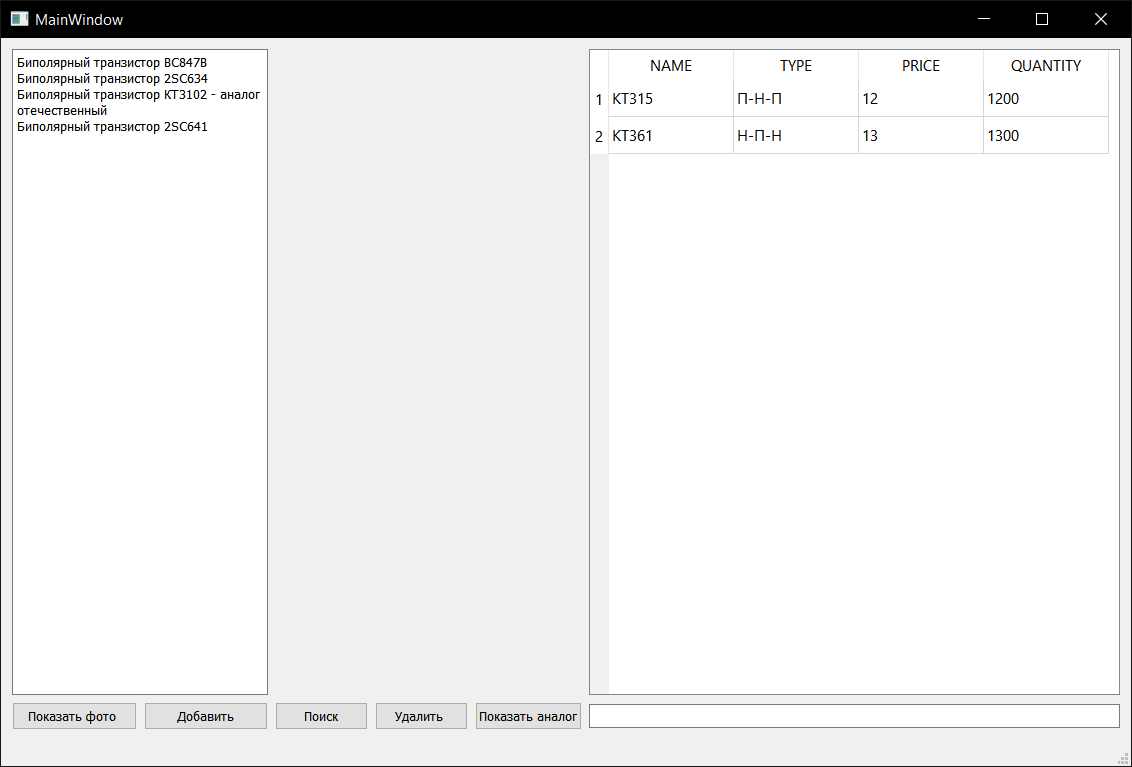


Рисунок 21 – выведены аналоги транзистора КТ315

Такую же операцию проделаем с кнопкой «показать фото». Действия повторяются такие же, только выбираем папку с названием «picture». Все последующие действия представлены на рисунках 22 – 24.

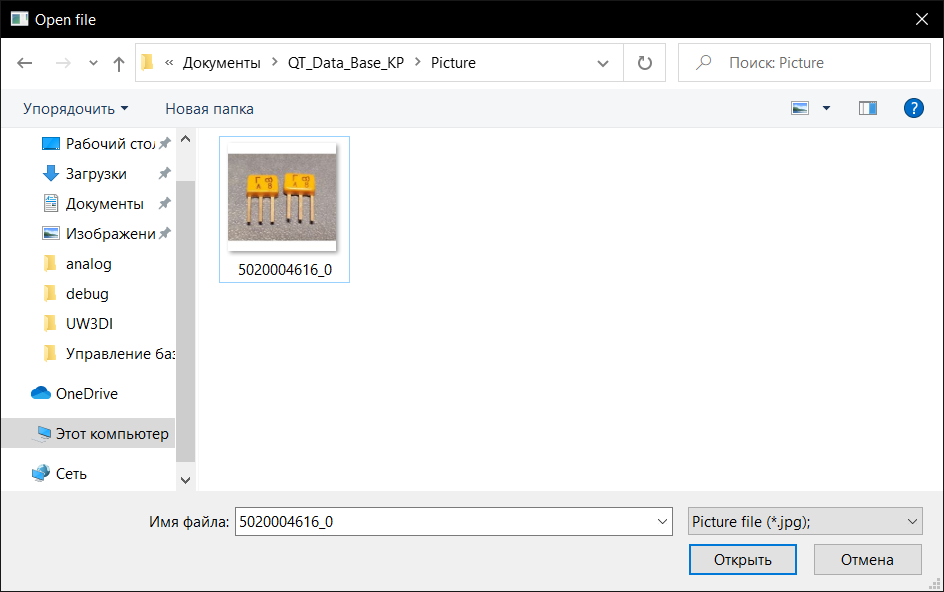


Рисунок 22 – открытая папка picture

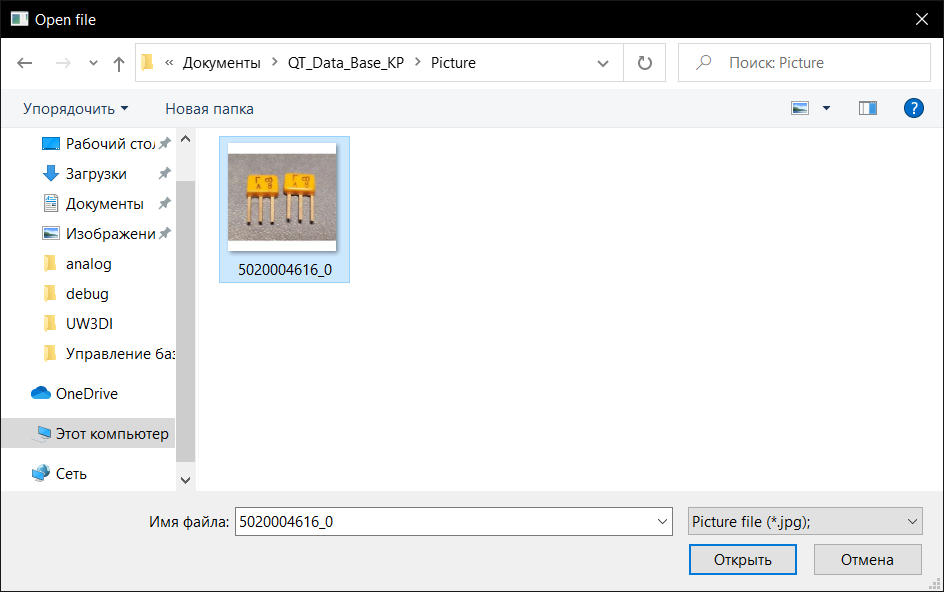


Рисунок 23 – выбранный файл 5020004616\_0 (КТ 315)

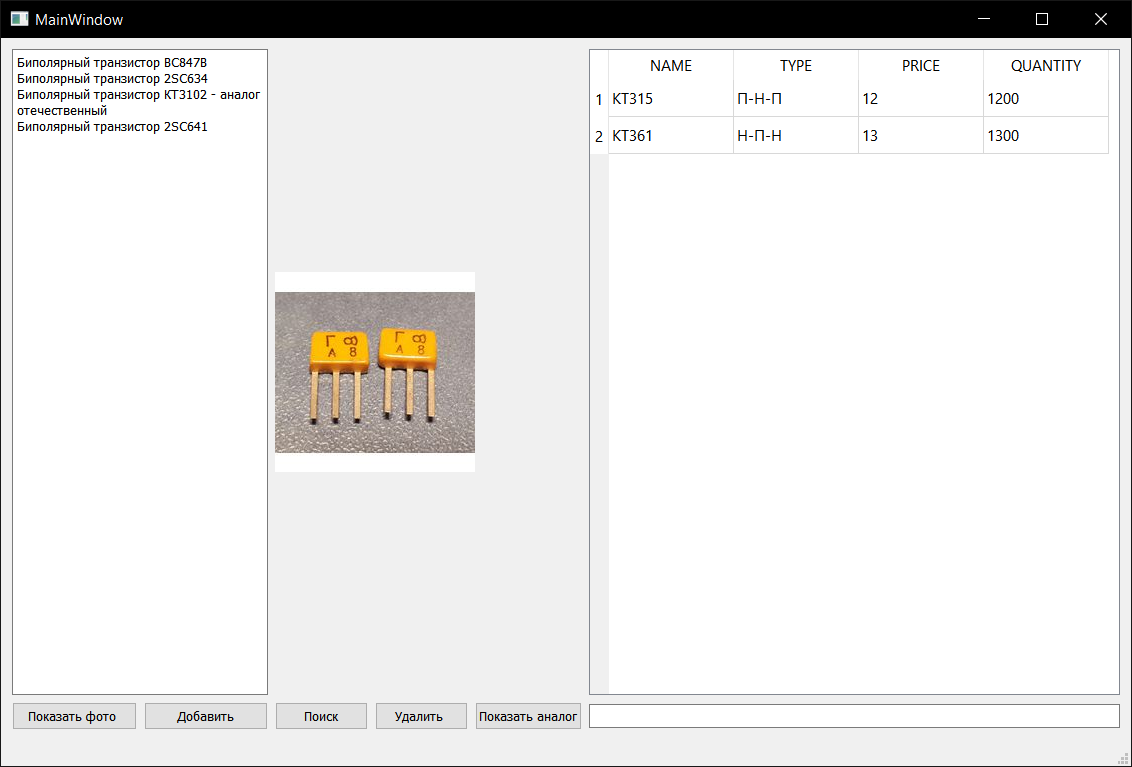


Рисунок 24 – открытое изображение на форме проекта

# **Заключение**

В результате выполнения данной курсовой работы были решены задачи проектирования и создания базы данных в которой отображается количество компонентов а так же их цена. В данной базе данных это транзисторы. Так же были введены функции отображения аналогов транзисторов, и реализован показ внешнего вида компонента, что позволяет сотруднику понять как выглядит тот или иной компонент.

Данная задача была реализована в программном продукте QT Creator 5.1.14.

# **Список литературы**

1. https://forum.cxem.net/index.php?/topic/201321-программа-для-учёта-радиодеталей/
2. Изучение основ языка SQL: методические указания к лабораторным работам по курсу «Базы данных» / Московский институт электроники и математики НИУ ВШЭ; Сост.: И.П. Карпова. – М., 2012 – 39 с.
3. Пушников, А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: Учебное пособие / Изд-е Башкирского ун-та. - Уфа, 1999. - 108 с.
4. Пушников А.Ю. Введение в системы управления базами данных. Часть 2. Нормальные формы отношений и транзакции: Учебное пособие / Изд-е Башкирского ун-та. - Уфа, 1999. - 138 с.
5. Проектирование реляционной базы данных: Метод. указания к курсовому проектированию по курсу "Базы данных" / Московский государственный институт электроники и математики; Сост.: Карпова И.П. – М., 2003. – 28 с.
6. Коннолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация, сопровождение. Теория и практика. – 2-е изд.: Пер. с англ.: Уч. пос. – М.: Издательский дом "Вильямс", 2000. – 1120 с.
7. Крёнке, Д. Теория и практика построения баз данных. 9-е изд. – СПб.: Питер, 2005 – 859 с.

**Приложение 1 листинг программы .pro**

QT += core gui sql

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

main.cpp \

mainwindow.cpp

HEADERS += \

mainwindow.h

FORMS += \

mainwindow.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

**Приложение 2 листинг программы .h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include<QSqlDatabase>

#include<QtDebug>

#include<QSqlQuery>

#include<QSqlTableModel>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_tableView\_clicked**(const QModelIndex &index);

void **on\_pushButton\_3\_clicked**();

void **on\_pushButton\_4\_clicked**();

private:

Ui::MainWindow \*ui;

QSqlDatabase db;

QSqlQuery \*query;

QSqlTableModel \* model;

int row;

QString fileName;

};

#endif // MAINWINDOW\_H

**Приложение 3 листинг программы main.cpp**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

return a.exec();

}

**Приложение 4 листинг программы maimwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QFileDialog>

#include <QMessageBox>

#include <QPixmap>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

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

db.setDatabaseName("./testQT.db");

if (db.open()){

qDebug("open");

}

else

{

qDebug("Close");

}

query = new QSqlQuery(db);

query->exec("CREATE TABLE Tranzistors (NAME TEXT, TYPE TEXT, PRICE INT, QUANTITY INT);");

model = new QSqlTableModel(this,db);

model->*setTable*("Tranzistors");

model->*select*();

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

// QString myPath = QCoreApplication::applicationDirPath()+"/testQT.db";

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_pushButton\_clicked**()

{

model->insertRow(model->*rowCount*());

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

model->removeRow(row);

}

void MainWindow::**on\_tableView\_clicked**(const QModelIndex &index)

{

row = index.row();

}

void MainWindow::**on\_pushButton\_3\_clicked**()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Open file"), "", tr("Text Files(\*.txt);"));

if (fileName != "") {

QFile file(fileName);

if (!file.*open*(QIODevice::ReadOnly)) {

QMessageBox::critical(this, tr("Error"), tr("Could not open file"));

return;

}

QTextStream in(*&file*);

ui->textBrowser->setPlainText(in.readAll());

}

}

void MainWindow::**on\_pushButton\_4\_clicked**()

{

{

//QString fileName = QCoreApplication::applicationDirPath()+"/5020004616\_0.jpg";

QString fileName = QFileDialog::getOpenFileName(this, tr("Open file"), "", tr("Picture file (\*.jpg);"));

QFile file(fileName);

// QPixmap in(&file);

// ui->label->setPixmap(QPixmap(in.readAll()));

QPixmap pix(fileName);

ui->label->setPixmap(pix);

ui->label->setPixmap(pix);

}}

void MainWindow::**on\_pushButton\_5\_clicked**()

{

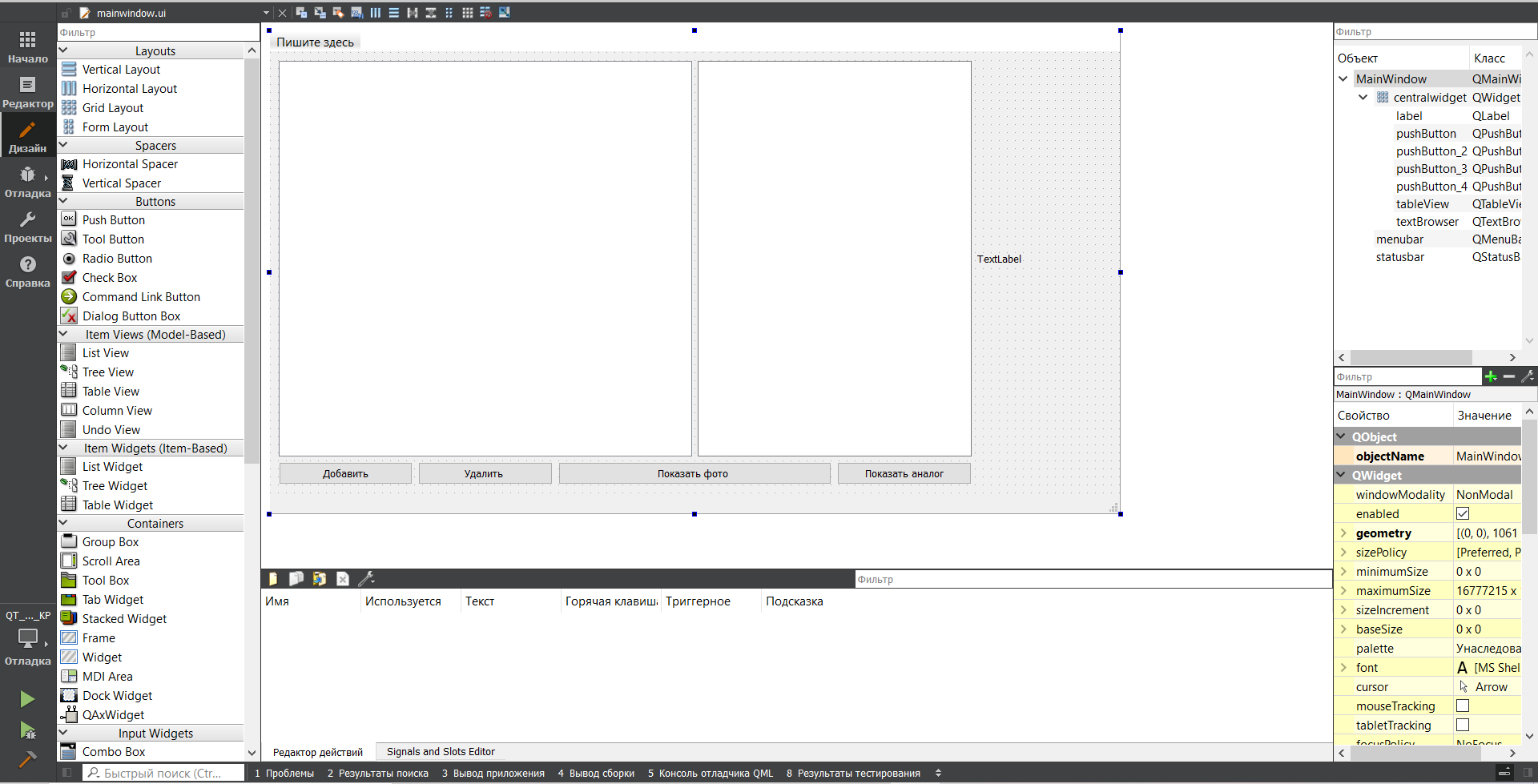
QSqlQueryModel \*model1=new QSqlQueryModel(this);

model1->setQuery(QString("select \* from Tranzistors where NAME = '%1'").arg(ui->lineEdit->text()));

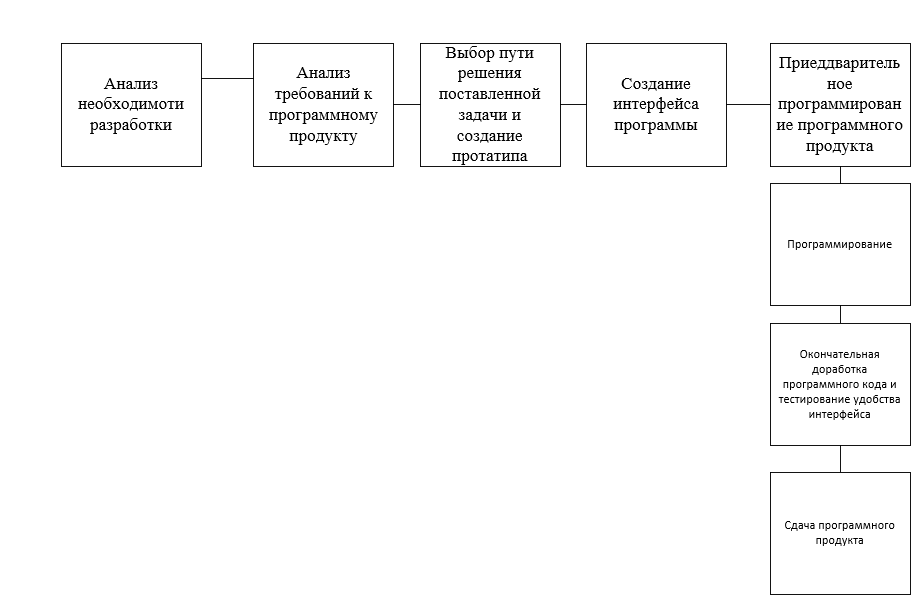
ui->tableView->*setModel*(*model1*);

}

**Приложение 5 Создание в дизайнере рабочего пространства**



**Схема разработки проектного решения**



**Алгоритмы, структуры для описания и хранения данных**

