**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Факультет инфокоммуникационных Сетей и систем (иксс)**

**кафедра программной инженерии и вычислительной техники (пи и вт)**

Дисциплина: «Объектно-ориентированное программирование»

**Курсовая Работа**

**«Разработка приложения для управления базой данных»**

**вариант №16**

Выполнил:

Студент группы ИКПИ-95

Принял:

Доцент кафедры ПИиВТ

Новиков С.С.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

Петрова О. Б.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_ 2021

**Оглавление**

**[1.](#_Toc59964163)****[Постановка задачи](#_Toc59964163)** [3](#_Toc59964163)

**[2.](#_Toc59964164)****[Выбор и обоснование типа разрабатываемого контейнера](#_Toc59964164)** [4](#_Toc59964164)

**[3.](#_Toc59964165)****[Разработка структуры программы](#_Toc59964165)** [4](#_Toc59964165)

**[4.](#_Toc59964166)****[Разработка собственных классов](#_Toc59964166)** [4](#_Toc59964166)

**[5.](#_Toc59964167)****[Разработка интерфейса](#_Toc59964167)** [5](#_Toc59964167)

**[6.](#_Toc59964168)****[Инструкция пользователю](#_Toc59964168)** [6](#_Toc59964168)

**[7.](#_Toc59964169)****[Заключение](#_Toc59964169)** [10](#_Toc59964169)

**[8.](#_Toc59964170)****[Код программы](#_Toc59964170)** [11](#_Toc59964170)

**[9.](#_Toc59964171)****[Список литературы](#_Toc59964171)** [25](#_Toc59964171)

# **Постановка задачи**

Создать базу данных телевизоров, работающую на основе контейнера, который следует разработать в процессе выполнения курсовой работы. В качестве прототипа разрабатываемого контейнера можно использовать контейнеры vector или list из библиотеки STL. Для работы с контейнером следует предусмотреть итератор. Остальные требования к контейнеру определяются студентом самостоятельно.

Разрабатываемая база данных должна быть создана использованием C++ и библиотеки Qt5. Среда программировании Qt Creater.

База данных должна обеспечивать выполнение следующих операций:

* создание базы данных,
* объединение баз данных,
* добавления и удаления записей,
* редактирования записей,
* просмотра базы данных,
* поиск данных (найти данные, удовлетворяющие заданному критерию),
* запись на диск,
* чтения с диска.

Интерфейсная часть программы должна содержать следующие компоненты:

* окно «ABOUT»,
* главное меню,
* всплывающее меню,
* строку состояния,
* панель инструментов,
* подсказки по командам меню и панели инструментов.

Пояснительная записка должна содержать:

* постановку задачи,
* обоснование типа разрабатываемого контейнера,
* разработку структуры программы.
* разработку отдельных модулей.
* выбор и обоснование используемых классов,
* разработку собственных классов,
* разработку интерфейса пользователя,
* инструкцию пользователя,
* заключение,
* список литературы.

# **Выбор и обоснование типа разрабатываемого контейнера**

В курсовой работе необходимо использовать контейнер для хранения объектов. В качестве прототипа разрабатываемого контейнера выбран vector. Выбор данного контейнера обусловлен относительной простотой разработки и организации, а так же возможностью произвольного доступа к своим элемента [1].

Разрабатываемый контейнер должен обладать следующими функциями:

* Создание пустого контейнера;
* Получение элемента по индексу;
* Вставка элемента в начало и конец;
* Очистка контейнера;
* Удаление элемента по индексу;

# **Разработка структуры программы**

Программа состоит из интерфейса и кода [2].

Интерфейс:

* Основное окно программы.
* Главное меню: QMenuBar.
* Таблица: QTableView.
* Строка поиска по таблице: QLineEdit.

Код:

* Основной класс формы QT — окно главной программы.
* Класс базы данных.
* Класс контейнера типа vector

# **Разработка собственных классов**

Course work— класс главного окна.

* Поля класса:
* Таблица соотношений TableView c БД: vector<int> ratio\_table;
* Ссылка на экземпляр базы данных: DataBase bd;
* Переменная для хранения выбранного поля: int sellect\_row
* Текст для поиска: QString find\_str;
* Методы класса:
* Очистка таблицы: void clearTable();
* Подсоединяет слот, обрабатывающий нажатие в таблице:

void tableSlotInit();

* Обновление таблицы: void updateTable();
* Переносит изменения из таблицы в БД: void synchronize();
* Обрабатывает выбор строки:

void slot\_sellect(QModelIndex, QModelIndex);

* Загрузка БД из файла: void slot\_load\_bd\_in\_file();
* Записывает БД в файл: void slot\_write\_bd\_in\_file();
* Добавляет строчку в таблицу: void slot\_add\_bd\_line();
* Удаляет строчку из таблицы: void slot\_delete\_bd\_line();
* Объединяет БД: void slot\_merge\_bd();
* Считывает параметр поиска: void slot\_bd\_find();

Vector — класс реализующий контейнер.

* Поля класса:
* Указатель на массив: T\* data;
* Количество элементов в массиве: size\_t size;
* Емкость массива: size\_t capasity;
* Методы класса
* void pushBack(const T& el);
* void emplaceBack(Args&&... args);
* void reserve(size\_t new\_capasity);
* void erase(size\_t begin, size\_t end);
* void clear();

DataBase—расширение класса vector методом наследования:

* Добавленные методы класса:
* void loadFromFile(const QString& path);
* void saveToFile(const QString& path);
* void margeFromFile(const QString& path);

Таким образом программу составляют модули:

main.cpp

coursework.h+cpp+ui  
 DataBase.h+cpp  
 vector.hpp+inl

# **Разработка интерфейса**

Основное окно программы (размер окна не меняется) (Рисунок 1).

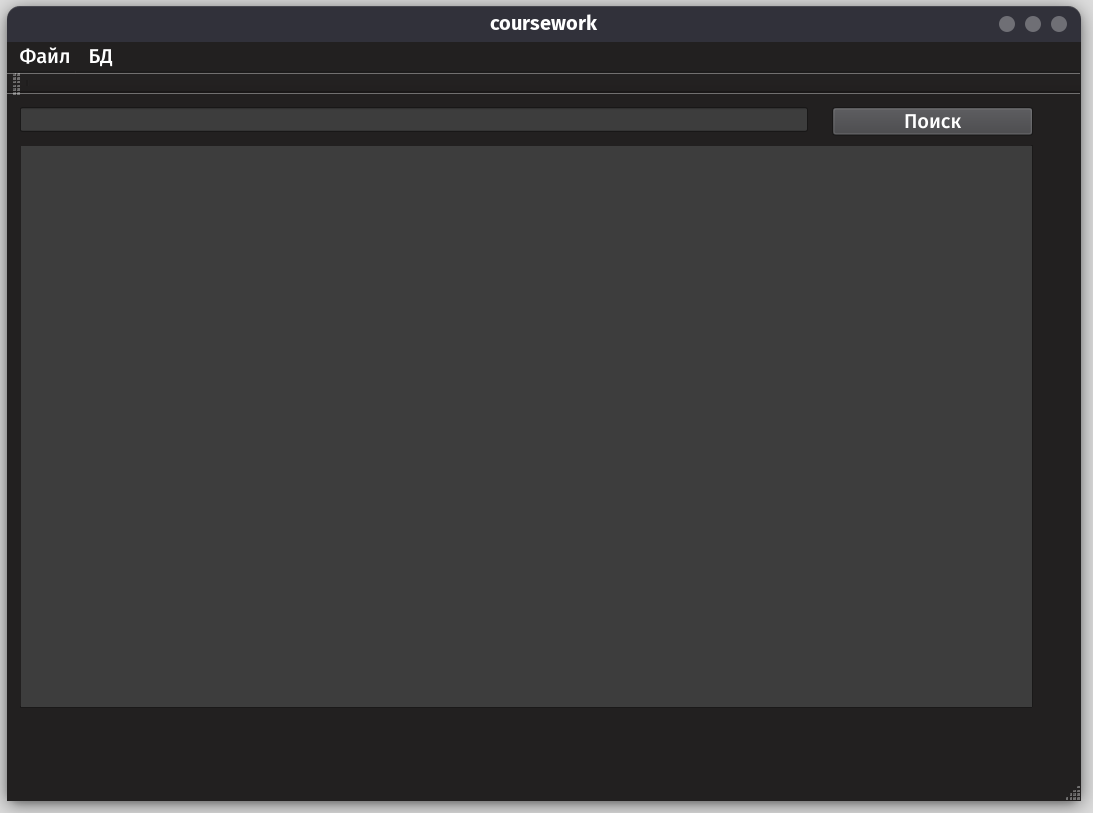


Рисунок 1: Главное меню, таблица, строка поиска

Для создания главного окна использовался родительский класс mainwindow, от которого наследвались уже все остальные элементы.  
Для реализации строки поиска были использованы QLineEdit (само поле поиска) и QPushButton(Кнопка «поиск»). Таблица реализована классом TableView. Функции меню реализованы через QMenuBar, в частности функции: открытие БД, сохранение изменений, слияние баз данных, а так же редактирование и удаление записей.

В главном меню содержатся разделы «Файл», «БД» .

Раздел «Файл» содержит операции открытия и сохранения изменений, базы и слияние БД.

«БД» содержит операции над записями в базе: добавление новой записи, удаление .

Большую часть основного окна программы занимает таблица. Она отображает элементы базы данных и результаты поиска по базе. Позволяет просматривать базу, а также редактировать двойным щелчком по ячейке.

И, наконец, строка поиска. Она расположена сразу над таблицей. Позволяет искать вхождения набора символов в строки таблицы. Поиск осуществляется только при вводе набора символов в строку поиска и нажатие клавиши «поиск». Программа мгновенно реагирует на ввод пользователя и отображает результаты поиска. Если строка пуста, отображается вся текущая база.  
  
Поля базы данных:

Поля БД представлены двумя типами переменных: QString- строковая переменная Qt и int-целочисленная переменная, хранящая в себе числа в диапазоне от -2 147 483 648 до 2 147 483 647.  
Название телевизора: QString name;  
Производитель телевизора: QString firm;

Цена телевизора: int price;  
Разрешения телевизора: QString resolution;  
Диагональ телевизора: QString diagonal;  
Умный телевизор или нет: bool smart;

# **Инструкция пользователю**

Открытие базы данных:

Чтобы открыть сохраненный ранее файл с базой данных можно выбрав в главном меню пункт [Файл] – [Загрузить файл] (рисунок 2).

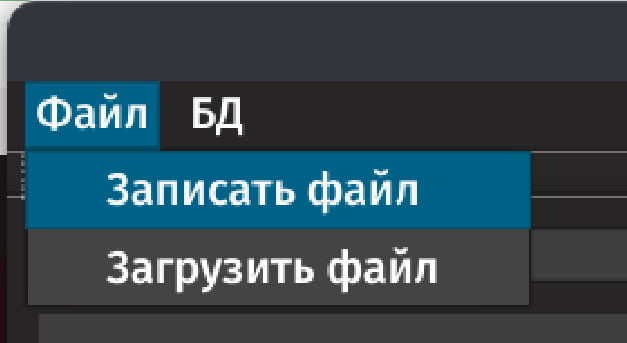


Рисунок 2.

База данных появится в таблице (Рисунок 3).

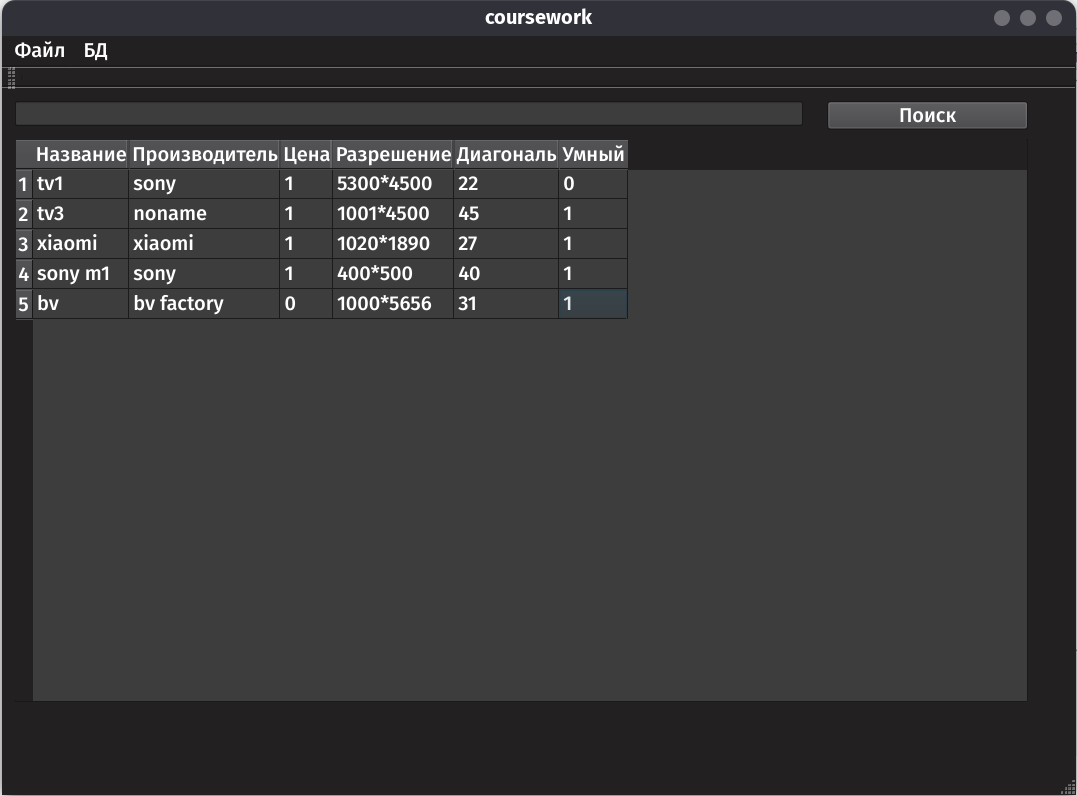


Рисунок 3: Демонстрация открытого файла

Сохранение базы данных:

Чтобы сохранить открытую базу данных можно выбрав в главном меню пункт [Файл] – [Записать файл] (Рисунок 4).

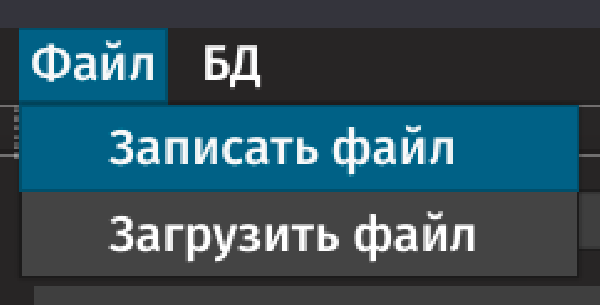


Рисунок 4.

БД будет сохранена в файл.

Объединение базы данных:

Чтобы объединить открытую базу данных с базой данных из файла можно выбрав в главном меню пункт [БД] – [Слияние] (Рисунок 5).

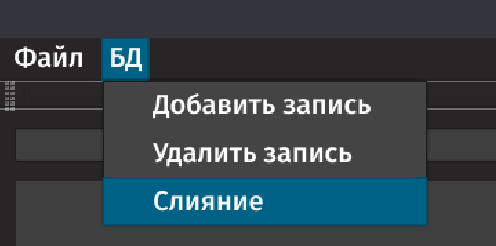


Рисунок 5.

Откроется диалоговое окно, в котором нужно выбрать файл с базой данных с расширением (\*.json). После выбора файла в диалоговом окне нажмите кнопку «Открыть» .После выбора файла база данных из файла будет добавлена в конец текущей базы (Рисунок 6).

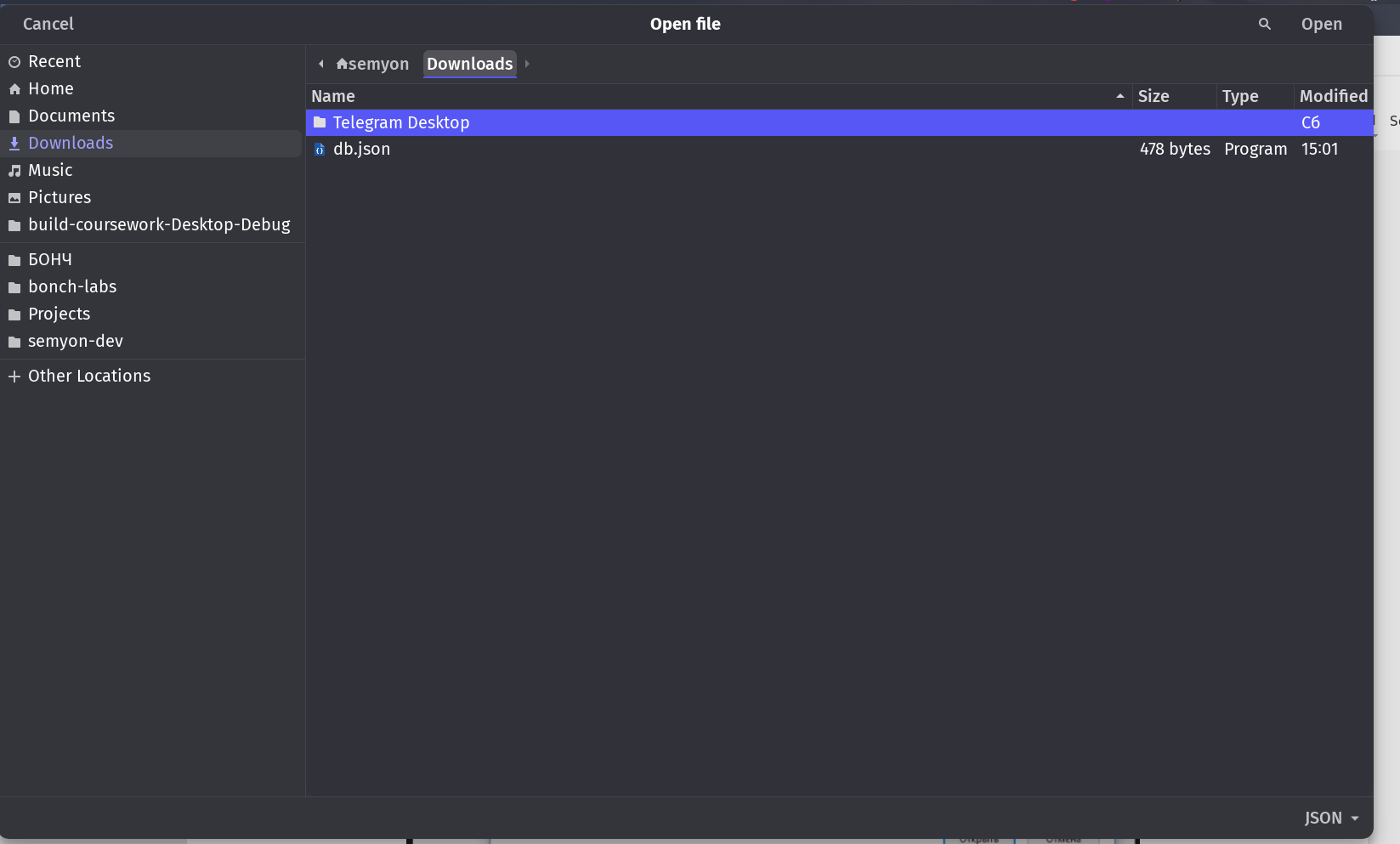


Рисунок 6.Окно выбора БД для слияния

Добавить запись в базу:

Чтобы добавить запись в базу данных можно выбрав в главном меню пункт [БД] – [Добавить запись].В конец базы будет добавлена пустая строчка (Рисунок 7). Добавить информацию в поле можно выполнив двойной клик по желаемому полю.Для сохранения изменений нужно нажать [Файл]-[Записать файл].

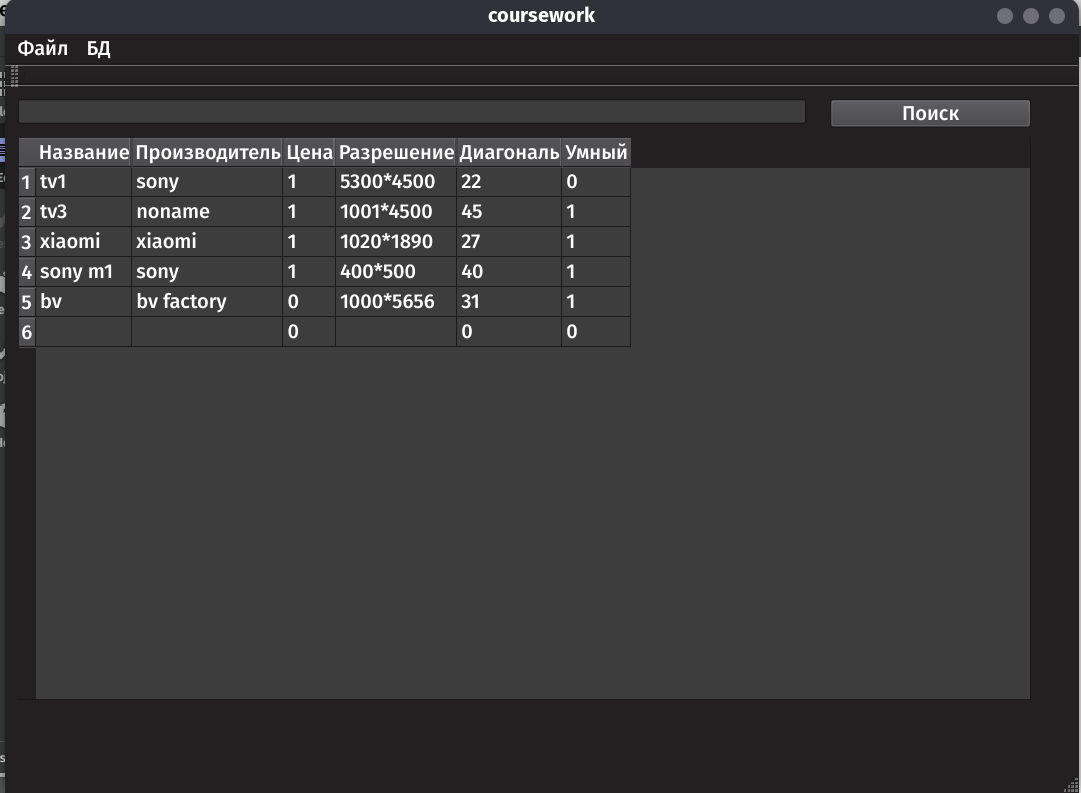


Рисунок 7: Демонстрация добавления новой строки в БД

Удаление записи из базы данных:

Чтобы удалить любую запись из базы данных, нужно нажать на нумерацию нужной записи и потом выбрать в главном меню пункт [БД] – [Удалить запись] (рисунок 8).

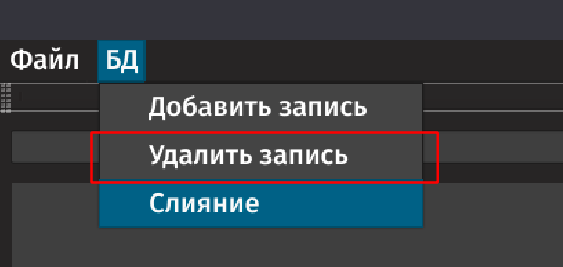


Рисунок 8: Удаление записи

Редактирование записи в базе данных:

Чтобы редактировать любую запись из базы данных, нужно выполнить двойной клик по любому полю. Текст внутри(если он имеется) выделится синим цветом (Рисунок 9). Внеся нужные изменения кликните по любому не занятому месту в БД. Для сохранения изменений нужно нажать [Файл]-[Записать файл]

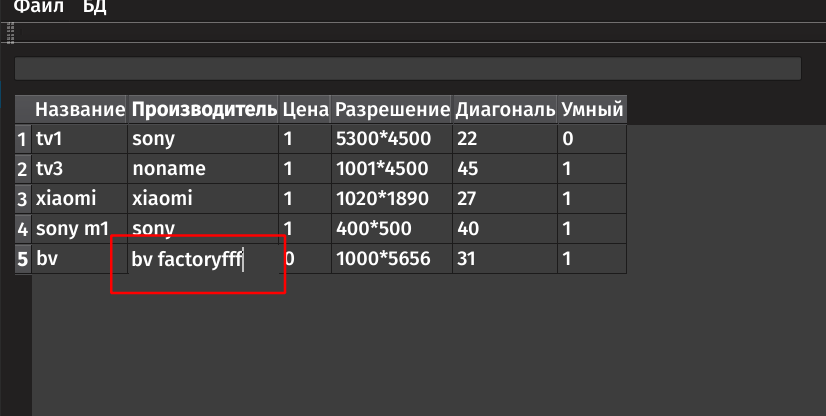


Рисунок 9: Редактирование

# **Заключение**

Программа выполняет все поставленные задачи: работа с базами данных (открытие, сохранение, добавление записи, удаление записей, редактирование записи и т.д., интуитивнопонятное отображение всех возможностей программы.

В ходе работы была выявлена следующая проблема: Qt поддерживает только один тип int, что ограничивает диапазон выводимых данных. Как решение можно предложить следующее: получить данные в текстовом формате QT(QString), конвертировать текст в стандартный текстовый формат C++(std::string) и с помощью функции std (\_strtoi64) конвертировать в число.  
Так же нельзя получить беззнаковое число, решить проблему можно использовать функцию std (\_strtoui64).

В ходе выполнения курсовой работы была освоена работа с основными средствами языка C++, библиотеки QT и редактором QT Creator.

# **Список литературы**

1. Справочные материалы Qt Creator. (Использовались при построении графического интерфейса и на протяжении всей разработки программы)
2. Коробов С. А. Лекции по дисциплине Объектно–ориентированное программирование. (Использовались при разработке класс базы данных)

# **Код программы**

**Main.cpp**

#include "coursework.h"

#include <QtWidgets/QApplication>

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

{

QApplication a(argc, argv);

coursework w;

w.show();

return a.exec();

}

**DataBase.h**

#pragma once

#include "vector.hpp"

#include <qstring.h>

struct TV

{

QString name; // название

QString firm; // фирма-производитель

int price; // цена

QString resolution; // разрешения

int diagonal; // диагональ

bool smart; // умный телевизор

};

class DataBase: public Novikov::Vector<TV>

{

public:

void loadFromFile(const QString& path);

void saveToFile(const QString& path);

void margeFromFile(const QString& path);

private:

};

**DataBase.cpp**  
#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

#include <QFile>

#include "DataBase.h"

void DataBase::**loadFromFile**(const QString& path)

{

clear();

QString val;

QFile file;

file.setFileName(path);

file.*open*(QIODevice::ReadOnly | QIODevice::Text);

val = file.readAll();

file.*close*();

QJsonDocument d = QJsonDocument::fromJson(val.toUtf8());

QJsonObject sett2 = d.object();

QJsonArray arr = sett2.value(QString("TV")).toArray();

for (size\_t i = 0; i < arr.size(); i++)

{

auto obj = arr[i].toObject();

TV tv;

tv.name = obj.value(u8"name").toString();

tv.firm = obj.value(u8"firm").toString();

tv.price = obj.value(u8"price").toInt();

tv.resolution = obj.value(u8"resolution").toString();

tv.diagonal = obj.value(u8"diagonal").toInt();

tv.smart = obj.value(u8"smart").toBool();

pushBack(tv);

}

}

void DataBase::**saveToFile**(const QString& path)

{

QJsonDocument json;

QJsonArray arr;

for (size\_t i = 0; i < getSize(); i++)

{

QJsonObject obj;

obj.insert("name", operator[](i).name);

obj.insert("firm", operator[](i).firm);

obj.insert("price", operator[](i).price);

obj.insert("resolution", operator[](i).resolution);

obj.insert("diagonal", operator[](i).diagonal);

obj.insert("smart", operator[](i).smart);

arr.push\_back(obj);

}

QJsonObject obj;

obj.insert("TV", arr);

json.setObject(obj);

QFile file;

file.setFileName(path);

file.*open*(QIODevice::WriteOnly | QIODevice::Text);

file.write(json.toJson(QJsonDocument::Compact));

file.*close*();

}

void DataBase::**margeFromFile**(const QString& path)

{

QString val;

QFile file;

file.setFileName(path);

file.*open*(QIODevice::ReadOnly | QIODevice::Text);

val = file.readAll();

file.*close*();

QJsonDocument d = QJsonDocument::fromJson(val.toUtf8());

QJsonObject sett2 = d.object();

QJsonArray arr = sett2.value(QString("TV")).toArray();

for (size\_t i = 0; i < arr.size(); i++)

{

auto obj = arr[i].toObject();

TV tv;

tv.name = obj.value(u8"name").toString();

tv.firm = obj.value(u8"firm").toString();

tv.price = obj.value(u8"price").toInt();

tv.resolution = obj.value(u8"resolution").toString();

tv.diagonal = obj.value(u8"diagonal").toInt();

tv.smart = obj.value(u8"smart").toBool();

pushBack(tv);

}

}

**Vector.hpp**

#pragma once

#include <utility>

#include <iostream>

namespace **Novikov**

{

template <typename T>

class **Vector**

{

public:

class **Iterator**

{

public:

**Iterator**();

**Iterator**(Vector& r);

**Iterator**(const Iterator& r) = default;

Iterator& operator=(const Iterator& r) = default;

Iterator& operator++(); //prefix

Iterator operator++(int);//postfix

Iterator operator+(size\_t i)const;

Iterator operator-(size\_t i)const;

bool operator==(const Iterator& r);

bool operator!=(const Iterator& r);

T& operator\*()const;

private:

Vector\* v;

size\_t i;

};

class **ConstIterator**

{

public:

**ConstIterator**();

**ConstIterator**(const Vector& r);

**ConstIterator**(const ConstIterator& r) = default;

ConstIterator& operator=(const ConstIterator& r) = default;

ConstIterator& operator++(); //prefix

ConstIterator operator++(int);//postfix

ConstIterator operator+(size\_t i)const;

ConstIterator operator-(size\_t i)const;

bool operator==(const ConstIterator& r);

bool operator!=(const ConstIterator& r);

const T& operator\*()const;

friend Vector;

private:

const Vector\* v;

size\_t i;

};

**Vector**();

**Vector**(const Vector& r);

**Vector**(Vector&& r) noexcept;

~**Vector**();

void **pushBack**(const T& el);

template <typename ...Args>

void **emplaceBack**(Args&&... args);

void **reserve**(size\_t new\_capasity);

void **erase**(size\_t begin, size\_t end);

void **clear**();

size\_t **getSize**() const;

Vector& operator=(const Vector& r);

Iterator **begin**();

Iterator **end**();

ConstIterator **begin**()const;

ConstIterator **end**()const;

ConstIterator **cbegin**()const;

ConstIterator **cend**()const;

T& operator[](size\_t i);

const T& operator[](size\_t i) const;

private:

T\* data;

size\_t size;

size\_t capasity;

};

}

#include "vector.inl"

**Vector.inl**  
#include "vector.hpp"

//#include "vector.hpp"

#pragma once

namespace **Novikov**

{

//---Iterator---//

template<typename T>

inline Vector<T>::Iterator::**Iterator**(): v(nullptr), i(0) {}

template<typename T>

inline Vector<T>::Iterator::**Iterator**(Vector& r) : v(&r), i(0) {}

template<typename T>

inline typename Vector<T>::Iterator& Vector<T>::Iterator::operator++() //prefix

{

++i;

return \*this;

}

template<typename T>

typename Vector<T>::Iterator inline Vector<T>::Iterator::operator++(int)//postfix

{

auto t = \*this;

++i;

return t;

}

template<typename T>

typename Vector<T>::Iterator Vector<T>::Iterator::operator+(size\_t i)const

{

auto t = \*this;

t.i += i;

}

template<typename T>

typename Vector<T>::Iterator Vector<T>::Iterator::operator-(size\_t i)const

{

auto t = \*this;

t.i -= i;

}

template<typename T>

bool Vector<T>::Iterator::operator==(const Iterator& r)

{

return v == r.v && i == r.i;

}

template<typename T>

bool Vector<T>::Iterator::operator!=(const Iterator& r)

{

return v != r.v || i != r.i;

}

template<typename T>

inline T& Vector<T>::Iterator::operator\*() const

{

return v->operator[](i);

}

//---iterator---//

//---ConstIterator---//

template<typename T>

inline Vector<T>::ConstIterator::**ConstIterator**() :v(nullptr), i(0) {}

template<typename T>

Vector<T>::ConstIterator::**ConstIterator**(const Vector& r) : v(&r), i(0) {}

template<typename T>

inline typename Vector<T>::ConstIterator& Vector<T>::ConstIterator::operator++() //prefix

{

++i;

return \*this;

}

template<typename T>

typename Vector<T>::ConstIterator inline Vector<T>::ConstIterator::operator++(int)//postfix

{

auto t = \*this;

++i;

return t;

}

template<typename T>

inline typename Vector<T>::ConstIterator Vector<T>::ConstIterator::operator+(size\_t i)const

{

auto t = \*this;

t.i += i;

}

template<typename T>

inline typename Vector<T>::ConstIterator Vector<T>::ConstIterator::operator-(size\_t i)const

{

auto t = \*this;

t.i -= i;

}

template<typename T>

bool Vector<T>::ConstIterator::operator==(const ConstIterator& r)

{

return v == r.v && i == r.i;

}

template<typename T>

bool Vector<T>::ConstIterator::operator!=(const ConstIterator& r)

{

return v != r.v || i != r.i;

}

template<typename T>

inline const T& Vector<T>::ConstIterator::operator\*() const

{

return v->operator[](i);

}

//---iterator---//

//---vector---//

template<typename T>

inline Vector<T>::**Vector**():

data(static\_cast<T\*>(operator new[](10 \* sizeof(T)))), size(0), capasity(10)

{}

template<typename T>

inline Vector<T>::**Vector**(const Vector& r):

data(static\_cast<T\*>(operator new[](r.size \* sizeof(T)))), size(r.size), capasity(r.size)

{

for (size\_t i = 0; i < size; i++)

new (data + i) T(r.data[i]);

}

template<typename T>

inline Vector<T>::**Vector**(Vector&& r) noexcept:

data(r.data), size(r.size), capasity(r.capasity)

{

r.data = nullptr;

}

template<typename T>

inline Vector<T>::~**Vector**()

{

if (data == nullptr)

return;

for (size\_t i = 0; i < size; i++)

data[i].~T();

operator delete[](data);

}

template<typename T>

void inline Vector<T>::**pushBack**(const T& el)

{

if (size == capasity)

reserve(size \* 2);

new (data + size) T(el);

size++;

}

template<typename **T**>

template<typename ...Args>

void inline Vector<T>::**emplaceBack**(Args&&... args)

{

if (size == capasity)

reserve(size \* 2);

new (data + size) T(std::move(args ...));

size++;

}

template<typename T>

void inline Vector<T>::**reserve**(size\_t new\_capasity)

{

if (this->capasity >= new\_capasity)

return;

T\* temp\_data = static\_cast<T\*>(operator new[](new\_capasity \* sizeof(T)));

for (size\_t i = 0; i < size; i++)

new (temp\_data + i) T(std::move(data[i]));

for (size\_t i = 0; i < size; i++)

data[i].~T();

operator delete[](data);

data = temp\_data;

capasity = new\_capasity;

}

template<typename T>

void inline Vector<T>::**erase**(size\_t begin, size\_t end)

{

for (size\_t i = begin; i != end; i++)

{

data[i].~T();

}

size -= end - begin;

for (size\_t i = begin; i < size; i++)

{

new (data + i) T(std::move(data[i + end - begin]));

data[i + end - begin].~T();

}

}

template<typename T>

void inline Vector<T>::**clear**()

{

for (size\_t i = 0; i < size; i++)

data[i].~T();

size = 0;

}

template<typename T>

inline size\_t Vector<T>::**getSize**() const

{

return size;

}

template<typename T>

Vector<T>& Vector<T>::operator=(const Vector& r)

{

for (size\_t i = 0; i < size; i++)

data[i].~T();

if (capasity < r.size)

{

operator delete[](data);

data = static\_cast<T\*>(operator new[](r.size \* sizeof(T)));

capasity = r.size;

}

size = r.size;

for (size\_t i = 0; i < size; i++)

new (data + i) T(r.data[i]);

return \*this;

}

template<typename T>

typename Vector<T>::Iterator inline Vector<T>::**begin**(){ return Iterator(\*this); }

template<typename T>

typename Vector<T>::Iterator inline Vector<T>::**end**(){Iterator i; i.i = size; i.v = this; return i;};

template<typename T>

typename Vector<T>::ConstIterator inline Vector<T>::**begin**()const { return ConstIterator(\*this); };

template<typename T>

typename Vector<T>::ConstIterator inline Vector<T>::**end**()const { ConstIterator i; i.i = size; i.v = this; return i; };

template<typename T>

typename Vector<T>::ConstIterator inline Vector<T>::**cbegin**()const { return ConstIterator(\*this); };

template<typename T>

typename Vector<T>::ConstIterator inline Vector<T>::**cend**()const { ConstIterator i; i.i = size; i.v = this; return i; };

template<typename T>

T& Vector<T>::operator[](size\_t i)

{

return data[i];

}

template<typename T>

const T& Vector<T>::operator[](size\_t i) const

{

return data[i];

}

//---vector---//

}

**Coursework.h**

#pragma once

#include <vector>

#include <QtWidgets/QMainWindow>

#include "ui\_coursework.h"

#include "DataBase.h"

class **coursework** : public QMainWindow

{

Q\_OBJECT

public:

**coursework**(QWidget \*parent = Q\_NULLPTR);

private:

void **clearTable**();

void **tableSlotInit**();

void **updateTable**();

void **synchronize**();

Ui::courseworkClass ui;

std::vector<int> ratio\_table;

DataBase bd;

int sellect\_row;

QString find\_str;

private slots:

void **slot\_sellect**(QModelIndex, QModelIndex);

void **slot\_load\_bd\_in\_file**();

void **slot\_write\_bd\_in\_file**();

void **slot\_add\_bd\_line**();

void **slot\_delete\_bd\_line**();

void **slot\_merge\_bd**();

void **slot\_bd\_find**();

};

**Coursework.cpp**#include "coursework.h"

#include <QStandardItemModel>

#include <QFile>

#include <QTableWidget>

#include <QFileDialog>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

coursework::**coursework**(QWidget\* parent)

: QMainWindow(*parent*), sellect\_row(-1)

{

ui.setupUi(this);

connect(ui.actionAdd, SIGNAL(triggered()), this, SLOT(slot\_add\_bd\_line()));

connect(ui.actionDelete, SIGNAL(triggered()), this, SLOT(slot\_delete\_bd\_line()));

connect(ui.actionMerge, SIGNAL(triggered()), this, SLOT(slot\_merge\_bd()));

connect(ui.actionLoad, SIGNAL(triggered()), this, SLOT(slot\_load\_bd\_in\_file()));

connect(ui.actionSave, SIGNAL(triggered()), this, SLOT(slot\_write\_bd\_in\_file()));

connect(ui.pushButton, SIGNAL(clicked()), this, SLOT(slot\_bd\_find()));

}

//private

void coursework::**clearTable**()

{

if (ui.tableView->model() == nullptr)

return;

delete ui.tableView->model();

}

void coursework::**tableSlotInit**()

{

QItemSelectionModel\* sm = ui.tableView->selectionModel();

connect(sm, SIGNAL(currentRowChanged(QModelIndex, QModelIndex)),

this, SLOT(slot\_sellect(QModelIndex, QModelIndex)));

}

void coursework::**updateTable**()

{

clearTable();

ratio\_table.clear();

QStandardItemModel\* model = new QStandardItemModel;

QStandardItem\* item;

QStringList horizontalHeader;

horizontalHeader.append(u8"Название");

horizontalHeader.append(u8"Производитель");

horizontalHeader.append(u8"Цена");

horizontalHeader.append(u8"Разрешение");

horizontalHeader.append(u8"Диагональ");

horizontalHeader.append(u8"Умный");

model->setHorizontalHeaderLabels(horizontalHeader);

for (size\_t i\_bd = 0, i\_vt = 0; i\_bd < bd.getSize(); i\_bd++)

{

if (find\_str == "" || bd[i\_bd].name.indexOf(find\_str) != -1|| bd[i\_bd].firm.indexOf(find\_str) != -1)

{

ratio\_table.push\_back(i\_bd);

item = new QStandardItem(bd[i\_bd].name);

model->setItem(i\_vt, 0, *item*);

item = new QStandardItem(bd[i\_bd].firm);

model->setItem(i\_vt, 1, *item*);

item = new QStandardItem(QString::number(bd[i\_bd].price));

model->setItem(i\_vt, 2, *item*);

item = new QStandardItem(bd[i\_bd].resolution);

model->setItem(i\_vt, 3, *item*);

item = new QStandardItem(QString::number(bd[i\_bd].diagonal));

model->setItem(i\_vt, 4, *item*);

item = new QStandardItem(QString::number(bd[i\_bd].smart));

model->setItem(i\_vt, 5, *item*);

i\_vt++;

}

}

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

ui.tableView->resizeRowsToContents();

ui.tableView->resizeColumnsToContents();

tableSlotInit();

}

void coursework::**synchronize**()

{

for (size\_t i = 0; i < ui.tableView->model()->*rowCount*(); i++)

{

auto index = ui.tableView->model()->*index*(i, 0);

bd[ratio\_table[i]].name = ui.tableView->model()->*data*(index).toString();

index = ui.tableView->model()->*index*(i, 1);

bd[ratio\_table[i]].firm = ui.tableView->model()->*data*(index).toString();

index = ui.tableView->model()->*index*(i, 2);

bd[ratio\_table[i]].price = ui.tableView->model()->*data*(index).toInt();

index = ui.tableView->model()->*index*(i, 3);

bd[ratio\_table[i]].resolution = ui.tableView->model()->*data*(index).toString();

index = ui.tableView->model()->*index*(i, 4);

bd[ratio\_table[i]].diagonal = ui.tableView->model()->*data*(index).toInt();

index = ui.tableView->model()->*index*(i, 5);

bd[ratio\_table[i]].smart = ui.tableView->model()->*data*(index).toBool();

}

}

//slots

void coursework::**slot\_sellect**(QModelIndex r, QModelIndex)

{

sellect\_row = r.row();

}

void coursework::**slot\_load\_bd\_in\_file**()

{

QString file\_name = QFileDialog::getOpenFileName(this, tr(u8"Open file"),

"",

tr("JSON (\*.json)"));

if (file\_name == "")

return;

find\_str = "";

bd.loadFromFile(file\_name);

updateTable();

}

void coursework::**slot\_write\_bd\_in\_file**()

{

if (ui.tableView->model() == nullptr)

return;

QString file\_name = QFileDialog::getSaveFileName(this, tr(u8"Save file"),

"",

tr("JSON (\*.json)"));

if (file\_name == "")

return;

synchronize();

bd.saveToFile(file\_name);

}

void coursework::**slot\_add\_bd\_line**()

{

synchronize();

bd.pushBack(TV());

updateTable();

ui.tableView->*scrollTo*(ui.tableView->model()->*index*(ui.tableView->model()->*rowCount*()-1, 0));

}

void coursework::**slot\_delete\_bd\_line**()

{

if (sellect\_row == -1)

{

return;

}

synchronize();

bd.erase(ratio\_table[sellect\_row], size\_t(ratio\_table[sellect\_row])+1);

updateTable();

}

void coursework::**slot\_merge\_bd**()

{

if (ui.tableView->model() == nullptr)

return;

QString file\_name = QFileDialog::getOpenFileName(this, tr("Open file"),

"",

tr("JSON (\*.json)"));

if (file\_name == "")

return;

synchronize();

find\_str = "";

bd.margeFromFile(file\_name);

updateTable();

}

void coursework::**slot\_bd\_find**()

{

find\_str = ui.lineEdit->text();

synchronize();

updateTable();

}

**Coursework.ui**<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>courseworkClass</class>

<widget class="QMainWindow" name="courseworkClass">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>859</width>

<height>607</height>

</rect>

</property>

<property name="windowTitle">

<string>coursework</string>

</property>

<widget class="QWidget" name="centralWidget">

<widget class="QTableView" name="tableView">

<property name="geometry">

<rect>

<x>10</x>

<y>40</y>

<width>811</width>

<height>451</height>

</rect>

</property>

</widget>

<widget class="QPushButton" name="pushButton">

<property name="geometry">

<rect>

<x>660</x>

<y>10</y>

<width>161</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Поиск</string>

</property>

</widget>

<widget class="QLineEdit" name="lineEdit">

<property name="geometry">

<rect>

<x>10</x>

<y>10</y>

<width>631</width>

<height>20</height>

</rect>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>859</width>

<height>21</height>

</rect>

</property>

<widget class="QMenu" name="menu">

<property name="title">

<string>Файл</string>

</property>

<addaction name="actionSave"/>

<addaction name="actionLoad"/>

</widget>

<widget class="QMenu" name="menu\_2">

<property name="title">

<string>БД</string>

</property>

<addaction name="actionAdd"/>

<addaction name="actionDelete"/>

<addaction name="actionMerge"/>

</widget>

<addaction name="menu"/>

<addaction name="menu\_2"/>

</widget>

<widget class="QToolBar" name="mainToolBar">

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

<addaction name="separator"/>

</widget>

<widget class="QStatusBar" name="statusBar"/>

<action name="actionSave">

<property name="text">

<string>Записать файл</string>

</property>

</action>

<action name="actionLoad">

<property name="text">

<string>Загрузить файл</string>

</property>

</action>

<action name="actionAdd">

<property name="text">

<string>Добавить запись</string>

</property>

</action>

<action name="actionDelete">

<property name="text">

<string>Удалить запись</string>

</property>

</action>

<action name="actionMerge">

<property name="text">

<string>Слияние</string>

</property>

</action>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources>

<include location="coursework.qrc"/>

</resources>

<connections/>

</ui>