МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

ТЕМА КУРСОВОЙ РАБОТЫ

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-181(2)
Брякина Сергея Владимировича

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель _____ Чабанов В.В
ассистент кафедры компьютерной
инженерии и моделирования _____

Симферополь, 2018

РЕФЕРАТ

Курсовая работа: 30 с.,2 ил.,10 ист.

Объект исследования – Базы данных в приложениях на языке C++.

Цель работы – проектирование и построение приложения на основе языка C++ и SQL.

Метод исследования – использование механизмов C++ и языка SQL для построения запросов и архитектуры проекта.

C++,БАЗЫДАННЫХ,ЯЗЫКЗАПРОСОВ,SQL,СОВРЕМЕННОЕПРОЕКТИРОВАНИЕ.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Язык программирования C++

Язык программирования C++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день C++ является одним из самых популярных и распространенных языков.

Своими корнями он уходит в язык Си, который был разработан в 1969—1973 годах в компании Bell Labs программистом Деннисом Ритчи (Dennis Ritchie). В начале 1980-х годов датский программист Бьерн Страуструп (Bjarne Stroustrup), который в то время работал в компании Bell Labs, разработал C++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

Впоследствии новый язык стал набирать популярность. В него были добавлены новые возможности, которые делали его не просто дополнением к Си, а совершенно новым языком программирования. В итоге "Си с классами" был переименован в C++. И с тех по оба языка стали развиваться независимо друг от друга.

C++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко C++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на C++. Но только системным программированием применение данного языка не ограничивается. C++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ. Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход

мобильное направление, где C++ тоже нашел свое применение. И даже в веб-разработке также можно использовать C++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем C++ - язык широкого пользования, на котором можно создавать практически любые виды программ.

C++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на C++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на C++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на C++ в приложения под эти платформы.

В отличие от Си язык C++ позволяет писать приложения в объектно-ориентированном стиле, представляя программу как совокупность взаимодействующих между собой классов и объектов. Что упрощает создание крупных приложений.

Основные этапы развития

В 1979-80 годах Бьерн Страуструп разработал расширение к языку Си - "Си с классами". В 1983 язык был переименован в C++.

В 1985 году была выпущена первая коммерческая версия языка C++, а также первое издание книги "Языка программирования C++", которая представляла первое описание этого языка при отсутствии официального стандарта.

В 1989 была выпущена новая версия языка C++ 2.0, которая включала ряд новых возможностей. После этого язык развивался относительно медленно вплоть до 2011 года. Но при этом в 1998 году была предпринята первая попытка

по стандартизации языка организацией ISO (International Organiztion for Standartization). Первый стандарт получил название ISO/IEC 14882:1998 или сокращенно C++98. В дальнейшем в 2003 была издана новая версия стандарта C++03.

В 2011 году был издан новый стандарт C++11, который содержал множество добавлений и обогащал язык C++ большим числом новых функциональных возможностей. После этого в 2014 году было выпущено небольшое добавление к стандарту, известное также как C++14. И еще один ключевой релиз языка намечен на 2017.

Базы Данных

Одним из важнейших условий обеспечения эффективного функционирования любой организации является наличие развитой автоматизированной информационной системы (АИС). АИС - все системы, реализующие автоматизированный сбор, обработку и манипулирование данными и включающие технические средства обработки данных, программное обеспечение и обслуживающий персонал. Современной формой АИС являются автоматизированные банки данных (АБД), которые включают в свой состав вычислительную систему, одну или несколько БД, систему управления базами данных (СУБД) и набор прикладных программ (ПП).

В настоящее время практически во всех сферах человеческой деятельности используются базы данных. В том числе решение перечисленных задач позволит достигнуть цели, поставленной в курсовой работе, а именно, реализовать базу данных «Справочная кассы Ж/д вокзала», что позволит следить за наличием билетов на различные маршруты, с учетом цены и вида мест. Данная база данных может применяться в различных организациях, занимающихся продажей услуг по перевозке людей по определенным маршрутам. Для обеспечения надежности системы управления данными необходимо выполнить следующие основные требования:

1. целостность и непротиворечивость данных,

2. достоверность данных,
3. простота управления данными,
4. безопасность доступа к данным.

Этим требованиям удовлетворяют реляционные базы данных, реализованные в современных профессиональных СУБД.

ГЛАВА 1
АНАЛИЗ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Выбор СУДБ:

В настоящее время в мире используется достаточно большое количество универсальных промышленных СУБД. Среди них можно выделить несколько несомненных лидеров, как по уровню развития технологий, так и по объему рынка — они вместе занимают более 90% мирового рынка СУБД. Это СУБД первого эшелона — Oracle, Microsoft SQL Server, MySQL, Microsoft Access и IBM DB2, в последнее время быстро становится популярна система с открытым кодом PostgreSQL. Список СУБД второго эшелона довольно велик, сюда относят такие СУБД, как Sybase, Informix, Ingress, Adabas, Interbase, Progress, Cache, Linter, Firebird, Teradata и т д.

Рассмотрим более подробно наиболее распространенные СУБД.

1. СУБД Oracle одна из наиболее мощных современных СУБД, предназначенных для реализации баз данных уровня корпорации, что предъявляет серьезные требования к серверу. Oracle может работать в большинстве операционных систем: Windows-NT, -2000, Linux, UNIX, AIX, Nowell Netware.

Использование Oracle в качестве СУБД дает возможность выбора языка программирования. Традиционно для этого используется язык PL/SQL, но можно использовать и гораздо более мощный язык программирования Java.

Oracle полностью располагает мощными и удобными средствами администрирования не только одного сервера, но и группы серверов, расположенных в разных частях планеты.

Основными преимуществами Oracle можно считать поддержку баз данных очень большого объема (до 64 Гбайт), мощные средства разработки и администрирования, поддержку многопроцессорности и двух языковых сред, а

также интеграцию с Web. Вместе с этим программа предъявляет серьезные аппаратные требования и высокую цену.

2. СУБД MS SQL Server-2000 предлагает широкий спектр услуг администрирования и легко масштабируется. Это позволяет использовать ее в информационных системах для среднего бизнеса и больших компьютерных информационных системах (КИС).

В основе платформы MS SQL Server используется среда Windows. Главное преимущество программы тесная интеграция с программными продуктами от Microsoft и возможность экспорта/импорта данных в большинство распространенных форматов данных, что позволяет использовать MS SQL Server как центральное хранилище данных.

3. СУБД Borland Interbase содержит все, что требуется от СУБД, предназначенной для нужд малого и среднего бизнеса. К тому же начиная с версии 6.0 программа стала бесплатной, что тоже существенно. Программа нетребовательна к аппаратной части. Borland Interbase поддерживается платформами Windows и Linux, а также UNIX, NetBSD, FreeBSD.

Популярные языки программирования от Borland, как Delphi, Kylix и C++ Builder, поставляются с компонентами, позволяющими работать с данной СУБД. Именно это позволяет достичь очень высокого быстродействия программы.

4.СУБД MySQL получила широкое распространение в качестве средства работы с базами данных в Интернете. Программа совершенно нетребовательна к ресурсам сервера, на котором работает, очень быстрая и к тому же совершенно бесплатная: исходные коды и дистрибутивы для различных платформ доступны на сайте в Интернете. Изначально программа была ориентирована на операционную систему Linux, но сейчас уже существуют версии программы для операционных систем Windows, UNIX, NetBSD, FreeBSD, AIX. В последнее время программа завоевывает популярность у пользователей Macintosh с использованием операционной системой Mac OSX.

5. СУБД MS Access используется для решения локальных офисных задач с ограниченным объемом данных и формирование отчетов по результатам работы, при этом отчеты могут быть представлены в стандартном для офисных приложений виде.

6 СУБД SQLite. SQLite — это встраиваемая кроссплатформенная БД, которая поддерживает достаточно полный набор команд SQL и доступна в исходных кодах (на языке C).

Выбор был сделан в пользу СУБД SQLite, по следующим причинам:

1. Доступность исходных кодов принципиально на языке C(СИ)
2. Хорошо продуманная логика и простота применения при написании запросов
3. Наличие встроенных драйверов в среде Qt.

Выбор среды разработки:

Qt — кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby, Java — Qt Jambi; PHP — PHP-Qt и другие. Выбор пал на данную среду разработки из-за легкости создания интерфейсов и доступности документации в самой среде разработки.

ГЛАВА 2
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Постановка задачи

В соответствии с целью работы требовалось создать приложение на основе СУДБ, которое имитировало работу реального приложения «Справочной кассы железно-дорожного вокзала».

Чтобы это реализовать требовалось изучить язык запросов SQL, и написать определенное количество основополагающих запросов для функционирования приложения

2.2 Описание алгоритмов

Шаблон проектирования

Для решения поставленной задачи воспользуемся шаблоном проектирования, представленным на рисунке 2.1

Данный шаблон описывает работы большинства приложений , работающих в нынешнее время. Он достаточно хорошо раскрывает основные принципы ООП такие как: Инкапсуляция, Наследование,Полиморфизм(не так явно).

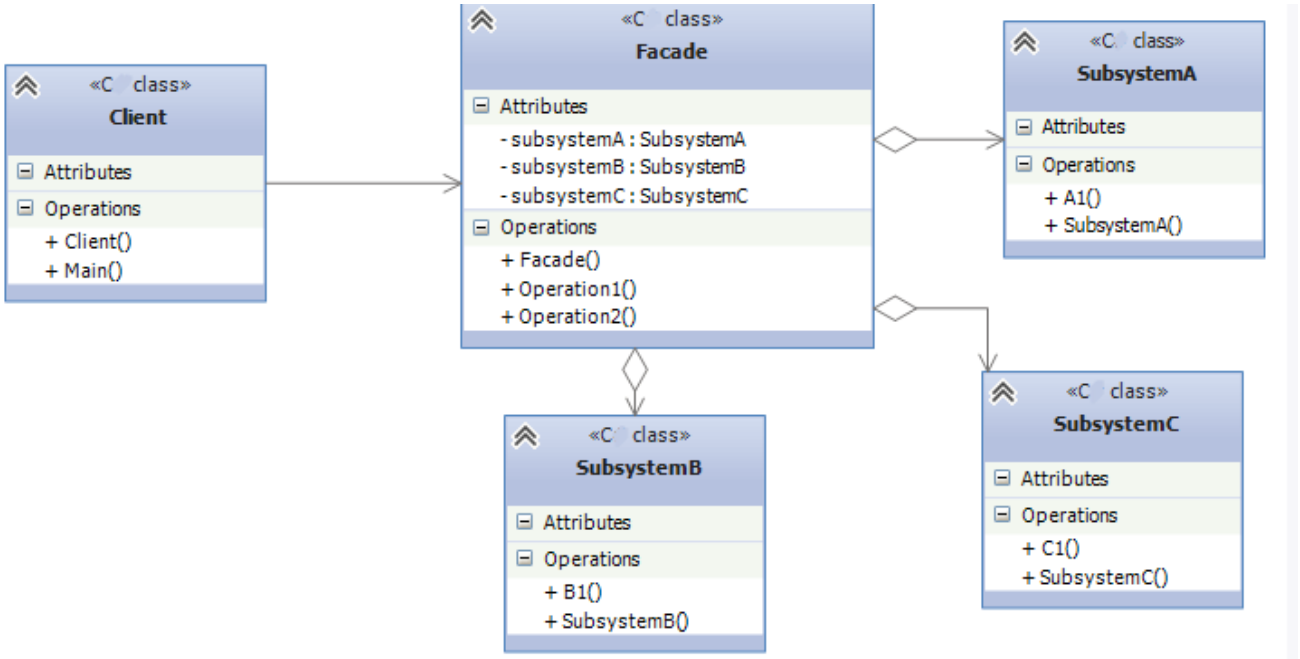

Рисунок 2.1 «Шаблон Фассад»

Работа запросов

Работа SQL запросов показана на рисунке 2.2

Рисунок 2.2 «Алгоритмы SQL»

Данный алгоритм показывает, что после запроса идет передача управления системе управления базы данных, которая в свою очередь обрабатывает сам запрос и в зависимости от имеющихся поле в базе, отклоняет или подтверждает запрос.

Для полного функционирования проекта было принято решение разбить его на два связанных между собой полноценных приложения. Одно из которых будет носить функции редактора БД, а другое функции клиента для покупки билетов.

2.3 Описание структур данных

Основной класс приложения, выполняющего функции редактора БД, определяющий работу запросов к базе, отвечающий за создание самой базы и подключения к ней, представлен ниже:

```
#ifndef DBASE_H
        #define DBASE_H
        # include <QtSql/QtSql>
        # include <QObject>
        #define BASE_NAME "projectDB"
        #define BASE_SEATS "seats"
        #define BASE_CARS "cars"
        #define NAME_CARS "carname"
        #define TYPE_CARS "typeofseats"
        #define COUNT_CARS "nseats"
        #define NAME_SEAT "carnm"
        #define TYPE_SEAT "typeofseat"
        #define COUNT_SEAT "nseat"
        #define COST_CAR "cost"
        #define TRY_BEGIN try {
        #define CATCH_BEGIN } catch(
        #define CATCH_ARG ) {
        #define CATCH_END }

        class QTableView;
        class QSqlTableModel;
        class Train : public QObject {
        Q_OBJECT
        public:
        Train();
        Train(QTableView* cars, QTableView* seats, QObject *parent =nullptr);
        virtual ~Train();
        void addCar(QString& carname,QString& type, QString& nSeats,QString& cost);
        //!< добавляет вагон
        void remSeat(QString& name,QString& type,QString& num);
        void remCar(QString& name,QString& type);
        //!< удаляет вагон
```

```
private:
void exec(QString);
//!< пытается выполнить запрос
QSqlDatabase m_db;
//!< база данных
QSqlQuery *m_query;
//!< запрос к базе
QSqlRecord m_rec;
//!< строка таблицы (ответ на запрос)
public:
QSqlTableModel *m_carsModel;
//!< модель таблицы автомобилей
QSqlTableModel *m_seatsModel;
//!< модель таблицы посадочных мест
};

#endif // DBASE_H
```

Из данного класса можно увидеть инкаплсулированные данные, которые будут определять запросы к БД, саму БД и её табличное отображение.

Видно, что конструктор с параметром класса `Train` переопределен и имеет в своих принимаемых аргументах две таблицы, которые должны предварительно располагаться в форме интерфейса приложения.

Далее следуют методы класса отвечающие за запросы к БД:

```
void addCar(QString& carname,QString& type, QString& nSeats,QString& cost);
//!< добавляет вагон
void remSeat(QString& name,QString& type,QString& num);
//!< удаляет место
void remCar(QString& name,QString& type);
//!< удаляет вагон
private:
void exec(QString&);
//!< пытается выполнить запрос
```

Приложение-клиент основано на таком же классе за исключением отсутствия методов добавления и удаления вагона и удаления места:

```
#ifndef CLIENTBASE_H
        #define CLIENTBASE_H
        # include <QtSql/QtSql>
        # include <QObject>
        #define BASE_NAME "projectDB"
        #define BASE_SEATS "seats"
        #define BASE_CARS "cars"
        #define NAME_CARS "carname"
        #define TYPE_CARS "typeofseats"
        #define COUNT_CARS "nseats"
        #define NAME_SEAT "carnm"
        #define TYPE_SEAT "typeofseat"
        #define COUNT_SEAT "nseat"
        #define TRY_BEGIN try {
        #define CATCH_BEGIN } catch(
        #define CATCH_ARG ) {
        #define CATCH_END }

        class QTableView;
        class QSqlTableModel;
        class Train : public QObject {
        Q_OBJECT
        public:
        Train(QTableView* cars, QObject *parent =nullptr);
        virtual ~Train();
        void addSeat(QString& name,QString& type, QString& snCar);
        //!< добавляет место
        private:
        void exec(QString);
        //!< пытается выполнить запрос
        QString qs(QString);
        //!< выделяет строку одинарными кавычками

        QSqlDatabase m_db;
        //!< база данных
        QSqlQuery *m_query;
        //!< запрос к базе
        QSqlRecord m_rec;
```

```
        //!< строка таблицы (ответ на запрос)
        QSqlTableModel *m_carsModel;
        //!< модель таблицы автомобилей
        QSqlTableModel *m_seatsModel;
        //!< модель таблицы посадочных мест

        };

        #endif // CLIENTBASE_H
```

Далее следует определить классы интерфейсов приложений они в целом похожи за исключением нескольких методов.

Описание класса интерфейса приложения-редактора БД:

```
#ifndef DIALOG_H
        #define DIALOG_H
        #include <Qdialog>

        namespace Ui {
        class Dialog;
        }
        class Train;
        class Dialog : public QDialog
        {
        Q_OBJECT

        public:
        explicit Dialog(QWidget *parent = nullptr);
        ~Dialog();
        public slots:
        void Slot_Del();
        void on_carDell();
        void chcekform();
        void on_carAdd();
        void Resize();
        void Seat_Dell();
        private:
        Train *m_db;
        Ui::Dialog *ui;
```

```
QShortcut* keyDell,* keyCtrl_Dell;
};

#endif // DIALOG_H
```

Класс интерфейса приложения-клиента:

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QtWidgets>

namespace Ui {
class Dialog;
}
class Train;
class Dialog : public QDialog
{
Q_OBJECT
public:
explicit Dialog(QWidget *parent = nullptr);
~Dialog();
public slots:
void timer();
void addSeat();
void show_seats();
private:
QTabWidget * tab;
QTimer *time_counter;
Train *m_db;
Ui::Dialog *ui;
};

#endif // DIALOG_H
```

Как видно из выше объявленных классов интерфейса приложений отличие заключается в слотах. Слот это функция, которая вызывается в ответ на определенный сигнал. Виджеты Qt имеют много предопределенных сигналов и

слотов, но мы всегда можем сделать дочерний класс и добавить наши сигналы и слоты в нем. Каждый слот отвечает за определенное действие с интерфейсом пр иложений.

Так же для удобства было реализованны другие окна по типу «Проданные билеты» или вывод на экран сообщения об ошибке запроса.

Пример такого класса:

```
#ifndef CAREXEP_H
        #define CAREXEP_H

        #include <QDialog>
        #include "dbase.h"
        namespace Ui {
        class CarExep;
        }

        class CarExep : public QDialog
        {
        Q_OBJECT

        public:
        explicit CarExep(QWidget *parent = nullptr);
        ~CarExep();
        public slots:
        private:
        Ui::CarExep *ui;
        };

        #endif // CAREXEP_H
```

Как видно описание данного класса очень просто, это связанно с упрощенным функционалом данных окон.

2.4 Описание основных функций

В соответствии с основными принципами реализации баз данных. Требовалось реализовать основной функционал для комфортного функционирования приложений.

Соответственно первое что требовалось это реализовать создание и подключение к базе:

```
Train::Train(QTableView* cars, QTableView* seats, QObject *parent)
        : QObject(parent) {
        m_db = QSqlDatabase::addDatabase("QSQLITE");
        m_db.setDatabaseName(QDir::currentPath()+"/"+BASE_NAME);
        QString PathToDb = QCoreApplication::applicationDirPath() + BASE_NAME;
        qDebug()<<PathToDb;
        if (false == m_db.open())
        throw "can't open/create DB";

        m_query = new QSqlQuery(m_db);
        if (false == m_db.tables().contains(BASE_SEATS))
        exec("CREATE TABLE " BASE_SEATS " (" // таблица посадочных мест
        NAME_SEAT" VARCHAR(250) NOT NULL , "// маршрут поезда
        TYPE_SEAT" VARCHAR(250) NOT NULL,"
        COUNT_SEAT" INTEGER, "// номер посадочного места
        "PRIMARY KEY (" NAME_SEAT "," TYPE_SEAT "," COUNT_SEAT ")"
        ");"
        );
        if (false == m_db.tables().contains(BASE_CARS))
        exec("CREATE TABLE " BASE_CARS " (" // таблица вагонов
        NAME_CARS" VARCHAR(250) NOT NULL , "// маршрут поезда
        TYPE_CARS" VARCHAR(250) NOT NULL , "
        COUNT_CARS" INTEGER, " // количество посадочных мест
        COST_CAR" INTEGER, "
        "PRIMARY KEY (" NAME_CARS "," TYPE_CARS ")"
        ");"
        );

        m_carsModel = new QSqlTableModel(this, m_db);
        m_carsModel->setTable(BASE_CARS);

        m_carsModel->setEditStrategy(QSqlTableModel::OnFieldChange);
        m_carsModel->setHeaderData(0,Qt::Horizontal, tr("Маршрут"));
        m_carsModel->setHeaderData(1,Qt::Horizontal,tr("Тип Мест"));
        m_carsModel->setHeaderData(2,Qt::Horizontal,tr("Кол-во Мест"));
```

```
m_carsModel->setHeaderData(3,Qt::Horizontal,tr("Стоимость билета"));
m_carsModel->select();
cars->setModel(m_carsModel);
cars->resizeColumnsToContents();

m_seatsModel = new QSqlTableModel(this, m_db);
m_seatsModel->setTable(BASE_SEATS);
m_seatsModel->setEditStrategy(QSqlTableModel::OnFieldChange);
m_seatsModel->setHeaderData(0,Qt::Horizontal,tr("Маршрут"));
m_seatsModel->setHeaderData(1,Qt::Horizontal,tr("Тип места"));
m_seatsModel->setHeaderData(2,Qt::Horizontal,tr("Место"));
m_seatsModel->select();
seats->setModel(m_seatsModel);

}
```

Подключение к базе происходит через конструктора основных классов приложений. Данная реализация справедлива реализациям двух основных классов, описанных в пункте 2.3.

На данном этапе можно наблюдать построение базы при помощи SQL запросов:

```
if (false == m_db.tables().contains(BASE_SEATS))
        exec("CREATE TABLE " BASE_SEATS " (" // таблица посадочных мест
        NAME_SEAT" VARCHAR(250) NOT NULL , "// маршрут поезда
        TYPE_SEAT" VARCHAR(250) NOT NULL,"
        COUNT_SEAT" INTEGER, "// номер посадочного места
        "PRIMARY KEY (" NAME_SEAT "," TYPE_SEAT "," COUNT_SEAT ")"
        ");"
        );
        if (false == m_db.tables().contains(BASE_CARS))
        exec("CREATE TABLE " BASE_CARS " (" // таблица вагонов
        NAME_CARS" VARCHAR(250) NOT NULL , "// маршрут поезда
        TYPE_CARS" VARCHAR(250) NOT NULL , "
        COUNT_CARS" INTEGER, " // количество посадочных мест
        COST_CAR" INTEGER, "
        "PRIMARY KEY (" NAME_CARS "," TYPE_CARS ")"
        ");"
              );
```

Соответственно наша база данных состоит из двух таблиц, в состав которых входят поля и составные ключи, таблицы построены по основным правилом построение баз данных.

Для описанного выше конструктора по всем правилам требуется деструктор:

```
Train::~Train() {
	delete m_query;
	}
```

Для приложения-редактора создано несколько запросов, как оговаривалось выше, ниже будет приведена их реализация:

Реализация запроса на добавление поезда:

```
void Train::addCar(QString& carname,QString& type, QString& nSeats,QString& cost) {
	exec(tr("INSERT INTO " BASE_CARS " (" NAME_CARS "," TYPE_CARS "," COUNT_CARS "," COST_CAR
") VALUES ('")
	+ carname+"','"+type+"',"+nSeats +","+cost+ ")");
	this->m_carsModel->select();

	}
```

Реализация запроса на удаление поезда:

```
void Train::remCar(QString& name,QString& type) {

	exec(tr("DELETE FROM " BASE_SEATS " WHERE " NAME_SEAT " = ") +"'"+name+"'"+" AND "
TYPE_SEAT " = '"+type+"'");
	exec(tr("DELETE FROM " BASE_CARS " WHERE " NAME_CARS " = ") +"'"+name+"'"+" AND " TYPE_CARS
" = '"+type+"'");
	this->m_carsModel->select();
	this->m_seatsModel->select();
	}
```

Реализация запроса на удаления места:

```
void Train::remSeat(QString& name,QString& type,QString& num) {
	exec(tr("DELETE FROM " BASE_SEATS " WHERE " NAME_SEAT " = '") + name +"'"+" AND " TYPE_SEAT
" = '"+type+"' AND " COUNT_SEAT " =" + num);
	this->m_seatsModel->select();
	}
```

Для приложения-клиента создан всего один запрос, как оговаривалось выше, это обоснованно тем, что в целях полноценного функционирования приложений, следует ограничить доступ покупателя к базе данных на столько насколько это возможно, ниже будет приведена его реализация:

```
void Train::addSeat(QString& name,QString& type,QString& num) {
	exec(tr("SELECT * FROM " BASE_CARS " WHERE " NAME_CARS " = ")+"'"+name+"'"+" AND "
TYPE_CARS " = "+"'"+type+"'");
	this->m_query->first();
	if (this->m_query->value(this->m_rec.indexOf(COUNT_CARS)).toString().toInt() < num.toInt())
	throw 1;
	exec(tr("INSERT INTO " BASE_SEATS " (" NAME_SEAT "," TYPE_SEAT "," COUNT_SEAT ") VALUES('")
+ name+ "','"+type+"'," + num + ")");
	this->m_seatsModel->select();
	}
```

В этих запросах можно наблюдать весь функционал SQL запросов. Пришлось реализовать различные проверки на возможность запроса к БД, использую различные конструкции из SQL запросов.

Далее следует определить реализацию некоторых особо важных методов и слотов классов интерфейсов приложений.

Конструктора у двух данных классов по причине схожести работы, то есть в двух конструкторах имеется инстанцирование объектов классов, отвечающих за соединение с БД.

Реализация приведена ниже:

```
Dialog::Dialog(QWidget *parent) :
	QDialog(parent),
	ui(new Ui::Dialog)
	{
	ui->setupUi(this);
	time_counter=new QTimer();
	time_counter->start(1000);
	QPixmap icon(":/new/client/images/Add_Action.png");
	QIcon Add_Icon(icon);
	ui->pushButton->setIcon(Add_Icon);
```

```
setWindowFlags(windowFlags() & ~Qt::WindowContextHelpButtonHint);
m_db = new Train(ui->cars,this);
connect(ui->pushButton,SIGNAL(clicked()),SLOT(addSeat()));
connect(time_counter,SIGNAL(timeout()),SLOT(timer()));
connect(ui->commandLinkButton,SIGNAL(clicked()),SLOT(show_seats()));

}
```

Для данных классов так же описаны и деструкторы:

```
Dialog::~Dialog()
{
        delete time_counter;
        delete ui;
        delete m_db;
}
```

Для двух приложений реализованы проверки на валидность запроса к БД. Данные проверки различаются только количеством проверяемых элементов, соответственно механизм реализован один и тот же.

Ниже приведен пример одной из таких проверок, в связи с объемами кода для проверки остальные будут опущены:

```
void Dialog::addSeat(){
    if(this->ui->travel->text().isEmpty()&&!this->ui->checkBox->isChecked()&&!this->ui->checkBox_2->isChecked()&&this->ui->spinBox->value()==0)
    {
    QMessageBox empty_all(QMessageBox::Critical,tr("Внимание!"),tr("Заполните поля для ввода!"),QMessageBox::Ok);
    empty_all.exec();
    return;
    }
    if(this->ui->travel->text().isEmpty())
    {
    QMessageBox text_empty(QMessageBox::Critical,tr("Внимание!"),tr("Введите желаемый маршрут!"),QMessageBox::Ok);
    text_empty.exec();
    return;
    }
    if(this->ui->spinBox->value()==0)
    {
```

```
        QMessageBox null(QMessageBox::Critical,tr("Внимание!"),tr("Введнено нулевое
место!"),QMessageBox::Ok);

        null.exec();

        return;

        }

        if(this->ui->checkBox->isChecked()&&this->ui->checkBox_2->isChecked())

        {

        QMessageBox checked_all(QMessageBox::Warning,tr("Внимание!"),tr("Вы хотите купить оба типа
места?"),QMessageBox::Yes|QMessageBox::No);

        if(checked_all.exec()==QMessageBox::Yes)

        {


        TRY_BEGIN

        this->m_db->addSeat(this->ui->travel->text(),this->ui->checkBox->text(),this->ui->spinBox-
>text());

        CATCH_BEGIN int CATCH_ARG

        QMessageBox selectexep(QMessageBox::Critical,tr("Внимание!"),tr("Введены некорректные
данные.Проверьте правильность заполнения!"),QMessageBox::Ok);

        selectexep.exec();

        return;

        CATCH_BEGIN...CATCH_ARG

        QMessageBox addexep(QMessageBox::Critical,tr("Внимание!"),tr("Введены некорректные
данные.Возможно место уже занято!"),QMessageBox::Ok);

        addexep.exec();

        return;

        CATCH_END

        QMessageBox succsesful(QMessageBox::Information,tr("Успешно"),"Место "+this->ui->travel-
>text()+" "+this->ui->checkBox->text()+" № "+this->ui->spinBox->text()+"
куплено!",QMessageBox::Ok);

        succsesful.exec();

        TRY_BEGIN

        this->m_db->addSeat(this->ui->travel->text(),this->ui->checkBox_2->text(),this->ui-
>spinBox->text());

        CATCH_BEGIN int CATCH_ARG

        QMessageBox selectexep(QMessageBox::Critical,tr("Внимание!"),tr("Введены некорректные
данные.Проверьте правильность заполнения!"),QMessageBox::Ok);

        selectexep.exec();

        return;

        CATCH_BEGIN...CATCH_ARG

        QMessageBox addexep(QMessageBox::Critical,tr("Внимание!"),tr("Введены некорректные
данные.Возможно место уже занято!"),QMessageBox::Ok);

        addexep.exec();

        return;
```

```
        CATCH_END

        QMessageBox succsesful_2(QMessageBox::Information,tr("Успешно"),"Место "+this->ui->travel-
>text()+" "+this->ui->checkBox_2->text()+" № "+this->ui->spinBox->text()+"
куплено!",QMessageBox::Ok);

        succsesful_2.exec();


        }

        else

        {

        QMessageBox checked_all(QMessageBox::Warning,tr("Внимание!"),tr("Вам следуте выбрать один
тип места!"),QMessageBox::Ok);

        checked_all.exec();

        }

        return;

        }

        TRY_BEGIN

        this->m_db->addSeat(this->ui->travel->text(),this->ui->checkBox->text(),this->ui->spinBox-
>text());

        CATCH_BEGIN int CATCH_ARG

        QMessageBox selectexep(QMessageBox::Critical,tr("Внимание!"),tr("Введены некорректные
данные.Проверьте правильность заполнения!"),QMessageBox::Ok);

        selectexep.exec();

        return;

        CATCH_BEGIN...CATCH_ARG

        QMessageBox addexep(QMessageBox::Critical,tr("Внимание!"),tr("Введены некорректные
данные.Возможно место уже занято!"),QMessageBox::Ok);

        addexep.exec();

        return;

        CATCH_END

        QMessageBox succsesful(QMessageBox::Information,tr("Успешно"),tr("Место
куплено!"),QMessageBox::Ok);

        succsesful.exec();

        }
```

Разбирая данных отрывок кода, можно увидет механизм исключений языка C++(СИ++). Данный подход к написанию кода, открывает широкий спектр возможностей, поэтому начиная со стандарта C++11, все разработчики прикладных программ используют этот механизм в проектах различного масштаба.

ГЛАВА 3
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Ниже приведен отрывок из технического задания моего проекта:

«...Программа должна корректно интерпретировать и соблюдать следующие

основные аспекты работы:

 пользователю интерфейс для работы с базой;

 существуют два приложения каждое отвечает за определенные функции;

 приложение-редактор позволяет пользователю вносить, редактировать, удалять поля справочной службы;

 клиентское приложение осуществляет функции покупки билетов на желаемый поезд;

 каждое приложение имеет определенные поля для ввода, которые соответствуют полям существующей базы данный

 ввод осуществляется при помощи SQL-запросов построенных для конкретной базы;

 в состав базы данных входят две таблицы: таблица маршрутов поездов и таблица проданных мест;

 таблица маршрутов состоит из таких столбцов как: Маршрут, Количество

мест, Тип мест, Цена за одно место;

 таблица мест состоит из таких столбцов как: Маршрут, Номер купленного места, Тип купленного места;

 приложение построено так чтобы нельзя было запустить две копии

одновременно для избежания некорректных запросов к существующей базе, более того нельзя запустить одновременно приложение-редактор и клиент;

 файл базы данных генерируется при первом запуске приложения-редактора;

 при удалении файла базы данных приложение продолжить функционировать, но к данные прошлой базы будут утеряны, будет создан новый файл базы данных;

 интерфейс сконструирован так чтобы исключить некорректное заполнение полей ввода, выводя при неправильном заполнении различные сообщения о допущенной ощибке;

 при корректном заполнении оба приложения выведут сообщение о успешном выполнении запроса;

 подключение к базе данных осуществляется с помощью SQL-драйверов;

2. Программа сохраняет незаконченный файл и загружает базу данных из него.

3. Существует вывод списка поездов и билетов на экран.

4. Управление осуществляется с помощью «мыши» и клавиатуры, так же присутствую определенные сочетания для быстрого удаления записей без заполнения полей ввода

5. Программа предупреждает о некорректном использовании запросов...»(Конец отрывка)

Тестирование приложений проходило в два этапа:

1. Тестирование интерфейса, так называемый «Черный ящик».
2. Тестирование кода, так называемый «Белый тест»

Тестирование интерфейса прошло успешно. Из десяти, написанных мною, тест-кейсов все прошли испытание.

Тестирование кода в свою очередь проходило в несколько этапов:

1. Визуальный осмотр.
    1. Данный этап не выявил ошибок, но по словам тестировщика не хватило комментариев в работе, из-за этого осмотр затянулся.
2. Статический анализ кода.

1. Данный этап был проведен при помощи программы для статического анализа кода на языке C++(СИ++) под названием CppCheck.
2. Анализ выявил ошибки инициализации полей классов без использования списков инициализации. Данная ошибка была определена анализатором как ошибка, влияющая на производительность приложения.

ЗАКЛЮЧЕНИЕ

В результате проделанной мною работы поставленная задача была выполнена на достаточном уровне. Полученные знания из различных сфер проектирования приложений пригодятся для дальнейшей разработки

В дальнейшем исходя из результатов тестирования и собственных пожеланий будет осуществятся доработка приложений.

Данные приложения являются готовым проектом для внедрения на предприятие для, которого они будут удовлетворяющими требованиям.

Данные приложения находятся в открытом доступе и их использование не нарушает ничьих прав.

ЛИТЕРАТУРА

1. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

2. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2016. – 31 с.

4. Шилдт Г. - C++. Базовый курс - 2010

5. Прата С. Язык программирования C++. Лекции и упражнения (6-е издание, 2011)

6. Павловская Т.А. - С С++. Программирование на языке высокого уровня - 2003

7. Бьерн Страуструп - Язык программирования C++ - 2006

8. Шлее М. - Qt 5.10. Профессиональное программирование на C++ (В подлиннике) - 2018

9. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ 2006 г.

10. Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О.В.Чеснокова, А.С.Чмыхало Программирование на языке C++в среде Qt Creator