

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждениевысшего образования

**«МИРЭА — Российский технологический университет»**

# **РТУ МИРЭА**

|  |  |  |
| --- | --- | --- |
| **Институт кибернетики**  **Базовая кафедра № 252 «Информационная безопасность»** | | |
|  |

## **КУРСОВАЯ РАБОТА**

по дисциплине «*Методы программирования*»

**Тема курсовой работы** Программа учета заявок на авиабилеты

**Студент группы** ККСО-06-20 Швецов Дмитрий Андреевич \_\_\_\_\_\_\_\_\_\_

*(учебная группа, фамилия, имя, отчество студента) (подпись студента)*

**Руководитель курсовой работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_**

*должность, звание, ученая степень (подпись руководителя)*

**Рецензент** (при наличии) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*должность, звание, ученая степень ( подпись рецензента)*

Работа представлена к защите \_\_\_\_\_\_\_\_\_\_\_\_\_ 202\_ г.

Допущен к защите «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 202\_\_г.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Оглавление**

1. [ВВЕДЕНИЕ 3](#_Toc121264442)
2. [ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 3](#_Toc121264443)
   1. [Общие сведения о базах данных и системах управления БД 3](#_Toc121264444)
   2. [Реляционные базы данных 7](#_Toc121264445)
   3. [Жизненный цикл базы данных 8](#_Toc121264446)
3. [Среды разработки приложений по работе с БД 9](#_Toc121264447)
   1. [Microsoft Visual Studio 9](#_Toc121264448)
   2. [.NET Core 9](#_Toc121264449)
   3. [Windows Forms 10](#_Toc121264450)
4. [ПРАКТИЧЕСКАЯ ЧАСТЬ 10](#_Toc121264451)
   1. [Класс Database 11](#_Toc121264452)
   2. [Реализация функции AddRecord & EditRecord 12](#_Toc121264453)
   3. [Реализация функции save() 13](#_Toc121264454)
   4. [Реализация функции load() 14](#_Toc121264455)
   5. [Реализация функции printDatabase() 15](#_Toc121264456)
   6. [Реализация функции search() 15](#_Toc121264457)
   7. [Вызов функции AddRecord 16](#_Toc121264458)
   8. [Вызов функции EditRecord 18](#_Toc121264459)
   9. [Реализация функции DeleteRecord 21](#_Toc121264460)

4.10.[Реализация меню консольного интерфейса 22](#_Toc121264461)

4.11.[Реализация main() 23](#_Toc121264462)

5. [ЗАКЛЮЧЕНИЕ 24](#_Toc121264463)

6. [Источники 25](#_Toc121264464)

# **ВВЕДЕНИЕ**

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. Целью данной курсовой работы является разработка базы данных для учёта заявок на авиабилеты. При создании программы использовалась IDE “Visual Studio Code”. Программа была написана с помощью языка программирования высокого уровня С++. C++ относится к семье языков с C-подобным синтаксисом. Язык имеет ряд важных достоинств:

1)Производительность. Благодаря своей вычислительной мощности язык обеспечивает высокую скорость исполнения кода

2)Отсутствие существенной нагрузки. Язык не утяжеляет программы, позволяет использовать их даже на старых устройствах.

3)Универсальность. Возможно, этот главный плюс C++. Он кроссплатформенный и подходит для любых целей.

В настоящее время, С++ по прежнему пользуется спросом, несмотря на 1985 год появления.

# **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

## **Общие сведения о базах данных и системах управления базами данных**

База данных – это набор данных для информационных сетей и пользователей, хранящихся в особом, организованном виде. Вид хранения данных определяется заданной структурой (схемой) базы данных и правилами ее управления.

Существуют следующие классификации баз данных:

− по модели данных;

− по среде постоянного хранения;

− по степени распределенности;

По модели данных выделяют следующие категории:

− иерархическая, когда база данных представлена в виде древовидной структуры, состоящей из объектов.

− сетевая, которая является расширением иерархической модели данных.

− реляционная, когда база данных представлена в виде сущностей, между которыми есть связь.

− объектно-ориентированная, когда данные в базе данных моделируются в виде объектов, их атрибутов, методов и классов.

− объектно-реляционная, которая основывается на объединении концепции реляционной модели данных с дополнительными объектно-ориентированными возможностями.

− функциональная, которая оперирует функциями и классами вместо таблиц и полей.

По среде постоянного хранения выделяют следующие категории:

− во вторичной памяти, когда средой постоянного хранения является периферийная энергонезависимая память.

− в оперативной памяти, когда все данные на стадии исполнения находятся в оперативной памяти.

− в третичной памяти, когда средой постоянного хранения является отсоединяемое от сервера устройство массового хранения.

По степени распределенности выделяют следующие категории:

− централизованная, когда база данных полностью поддерживается на одном компьютере.

− распределенная, когда составные части базы данных размещаются в различных узлах компьютерной сети.

− однородная, когда фрагменты распределенной базы данных в разных узлах поддерживаются средствами одной и той же СУБД.

− неоднородная, когда фрагменты распределенной базы данных в разных узлах поддерживаются средствами более чем одной СУБД.

− фрагментированная, когда методом распределения данных является горизонтальное или вертикальное фрагментирование.

− тиражированная, когда методом распределения данных является репликация.

Система управления базами данных – комплекс программных и языковых средств, которые обеспечивают управление созданием и использованием баз данных. Выделяют следующие функции СУБД:

− управление данными во внешней памяти (на дисках).

− управление данными в оперативной памяти с использованием дискового кэша.

− журнализация изменений (сохранение истории), резервное копирование и восстановление базы данных после сбоев.

− поддержка языков БД (язык определения данных, язык манипулирования данными). Каждая СУБД основывается на какой-либо модели данных, поэтому классификация СУБД включает в себя столько же категорий, сколько и классификация моделей данных.

Наиболее популярными СУБД являются Oracle, MySQL, Microsoft SQL Server, Postgree SQL. MySQL считается одной из самых распространенных СУБД.

MySQL — реляционная СУБД с открытым исходным кодом, главными плюсами которой являются ее скорость и гибкость, которая обеспечена поддержкой большого количества различных типов таблиц. Кроме того, это надежная бесплатная система с простым интерфейсом и возможностью синхронизации с другими базами данных. В совокупности эти факторы позволяют использовать MySQL как крупным корпорациям, так и небольшим компаниям.

Microsoft SQL Server оптимальна для использования в операционных системах семейства Windows, однако может работать и с Linux. Система позволяет синхронизироваться с другими программными продуктами компании Microsoft, а также обеспечивает надежную защиту данных и простой интерфейс, однако отличается высокой стоимостью лицензии и повышенным потреблением ресурсов. В целом, однако, сохраняет свою популярность, в немалой степени из-за того, что продукты корпорации Microsoft используются многими компаниями.

СУБД PostgreSQL — еще одна популярная и бесплатная система. Наибольшее применение нашла для управления БД веб-сайтов и различных сервисов. Она универсальна, то есть подойдет для работы с большинством популярных платформ. При этом PostgreSQL — объектно-реляционная СУБД, что дает ей некоторые преимущества над другими бесплатными СУБД, в большинстве являющимися реляционными.

Первая версия Oracle, которая является объектно-реляционной СУБД, появилась в конце 70-х, и с тех пор зарекомендовала себя как надежная, функциональная и практичная. СУБД Oracle постоянно развивается и дорабатывается, упрощая установку и первоначальную настройку и расширяя функционал. Однако существенным минусом данной СУБД является высокая стоимость лицензии, поэтому она используется в основном крупными компаниями и корпорациями, работающими с огромными объемами данных

## **Реляционные базы данных**

Реляционные базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют ключом. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных. В первых базах данных данные каждого приложения хранились в отдельной уникальной структуре. Если разработчик хотел создать приложение для использования таких данных, он должен был хорошо знать конкретную структуру, чтобы найти необходимые данные. Такой метод организации был неэффективен, сложен в обслуживании и затруднял оптимизацию эффективности приложений.

Реляционная модель была разработана, чтобы устранить потребность в использовании разнообразных структур данных. Она обеспечила стандартный способ представления данных и отправки запросов, которые могли быть использованы в любых приложениях. Разработчики уяснили, что таблицы являются ключевым преимуществом реляционных баз данных, так как обеспечивают интуитивно понятный, эффективный и гибкий способ хранения структурированной информации и получения к ней доступа. Со временем, когда разработчики стали использовать язык структурированных запросов (SQL) для записи данных в базу и отправки запросов, стало очевидным и другое преимущество реляционной модели. Вот уже на протяжении многих лет SQL широко используется в качестве языка запросов в базах данных. Он основан на алгоритмах реляционной алгебры и четкой математической структуре, что обеспечивает простоту и эффективность при оптимизации любых запросов к базе данных.

Для сравнения: при использовании других подходов приходится создавать отдельные, уникальные запросы.

Реляционная модель данных включает следующие компоненты:

− Структурный аспект — данные в базе данных представляют собой набор отношений;

− Аспект целостности — отношения отвечают определённым условиям целостности. РМД поддерживает декларативные ограничения целостности уровня домена (типа данных), уровня отношения и уровня базы данных;

− Аспект обработки — РМД поддерживает операторы манипулирования отношениями.

## **Жизненный цикл базы данных**

Жизненный цикл базы данных – период создания и использования базы данных, охватывающий ее различные состояния, начиная с момента возникновения необходимости в данной базе данных и заканчивая моментом ее полного выхода из употребления в информационной системе.

Жизненный цикл базы данных включает в себя следующие этапы:

− этап начальной разработки;

− проектирование базы данных;

− реализация и загрузка;

− тестирование и оценка;

− функционирование;

− сопровождение;

− вывод из эксплуатации.

# **Среды разработки приложений по работе с БД**

## **Microsoft Visual Studio**

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов или инструментов для прочих аспектов процесса разработки программного обеспечения.

## **.NET Core**

.NET — это модульная платформа для разработки программного обеспечения с открытым исходным кодом. Совместима с такими операционными системами как Windows, Linux и macOS. Была выпущена компанией Microsoft. Поддерживает такие языки программирования, как C#, Visual Basic .NET и F#. NET Core основана на .NET Framework. Платформа .NET Core отличается от неё модульностью, кроссплатформенностью, возможностью применения облачных технологий, и тем, что в ней произошло разделение между библиотекой CoreFX и средой выполнения CoreCLR.

.NET Core — модульная платформа. Каждый её компонент обновляется через менеджер пакетов NuGet, а значит можно обновлять её модули по отдельности, в то время как .NET Framework обновляется целиком. Каждое приложение может работать с разными модулями и не зависит от единого обновления платформы.

## **Windows Forms**

Windows Forms — это платформа пользовательского интерфейса для создания классических приложений Windows. Она обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в Visual Studio. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений. В Windows Forms можно разрабатывать графически сложные приложения, которые просто развертывать, обновлять, и с которыми удобно работать как в автономном режиме, так и в сети. Приложения Windows Forms могут получать доступ к локальному оборудованию и файловой системе компьютера, на котором работает приложение.

# **ПРАКТИЧЕСКАЯ ЧАСТЬ**

Задачей курсовой работы была написание программы учета авиабилетов. Полный код программы состоит из двух частей . Библиотеки .h, с объявлением функций , и файл .срр, содержащий реализацию функций. Сама же программа реализована с использованием текстового файла .txt, базового и самого простого метода реализации БД.

Для начала, будет задан класс Database, где объявляются функции, а так же переменные , которые будут записаны в БД. **Public** — публичный. Классы и методы, объявленные public, могут быть доступны как внутри самого класса, так и в дочерних классах и в объектах, реализовавших класс. **Protected** — объявляет метод или свойство защищенными. То есть такими, которые не могут быть доступны из объекта, реализующего класс, но вполне может быть использовано в дочерних классах.

## **Класс Database**

*Листинг 1*

**class** Database

{

**public**:

Database(){count = 0;}

**void** addRecord(string name, string lastname, string flightnum, string date, string time);

string getNameById(**int** id);

string getLastNameById(**int** id);

string getFlightNumById(**int** id);

string getDateById(**int** id);

string getTimeById(**int** id);

**int** getCount();

**void** save();

**void** load();

**void** editRecord(**int** id, string name, string lastname, string flightnum, string date, string time);

**void** printDatabase();

**void** search();

**protected**:

**int** count;

string name[100];

string lastname[100];

string flightnum[100];

string date[100];

string time[100];

};

Первым делом, для того чтобы начать использование базы данных – нужно добавить первую запись в txt файл, чтобы после – вносить какие-либо изменения. Для точечных изменений записи – требуется задать каждой записи свой индекс начиная с 0, а после, с помощью заданной нумерации , возможно внести изменения путём замены данных переменных. Указатель this — это указатель, доступный только в нестатических функциях-членах class struct типа или union типа. Он указывает на объект, для которого вызывается функция-член. Статические функции-члены не имеют указателя this .

## **Реализация функции AddRecord & EditRecord**

*Листинг 2*

**int** Database::getCount()

{

**return** count;

}

**void** Database::addRecord(string name, string lastname, string flightnum, string date, string time)

{

**this**->name[count] = name;

**this**->lastname[count] = lastname;

**this**->flightnum[count] = flightnum;

**this**->date[count] = date;

**this**->time[count]=time;

++count;

}

**void** Database::editRecord(**int** id, string name, string lastname, string flightnum, string date, string time)

{

**this**->name[id] = name;

**this**->lastname[id] = lastname;

**this**->flightnum[id] = flightnum;

**this**->date[id]=date;

**this**->time[id]=time;

}

string Database::getNameById(**int** id)

{

**return** name[id];

}

string Database::getLastNameById(**int** id)

{

**return** lastname[id];

}

string Database::getFlightNumById(**int** id)

{

**return** flightnum[id];

}

string Database::getDateById(**int** id)

{

**return** date[id];

}

string Database::getTimeById(**int** id)

{

**return** time[id];

}

Реализация функции save() . Для работы с файлами необходимо подключить заголовочный файл <fstream>**.**В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream>**—**файловый ввод и  <ofstream>**—**файловый вывод.

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам cin иcout.

Например, необходимо создать текстовый файл и записать в него строку . Для этого необходимо проделать следующие шаги:

1. создать объект класса ofstream**;**
2. связать объект класса с файлом, в который будет производиться запись;
3. записать строку в файл;
4. закрыть файл.

Почему необходимо создавать объект класса ofstream, а не класса ifstream? Потому, что нужно сделать запись в файл, а если бы нужно было считать данные из файла, то создавался бы объект класса ifstream

Если же появятся проблемы с открытием файла – с помощью cerr будет выведена ошибка. Так же происходит проверка переменных на %delete. Таким флагом будут далее помечаться данные после функции удаления.

## **Реализация функции save()**

*Листинг 3*

**void** Database::save()

{

ofstream output;

output.open("database.txt");

**if**(!output.is\_open())

{

cerr << "Error open database.txt" << endl;

exit(1);

}

output.clear();

string delete\_flag = "%delete";

**for**(**int** i=0; i < count; ++i)

{

**if**(getNameById(i) != delete\_flag && getLastNameById(i) != delete\_flag && getFlightNumById(i) != delete\_flag && getDateById(i) != delete\_flag && getTimeById(i) != delete\_flag)

{

output << getNameById(i) << " " << getLastNameById(i) << " " << getFlightNumById(i) << " " << getDateById(i) << " " << getTimeById(i) << endl;

}

}

output.close();

}

Функция load() требуется для загрузки уже существующих данных из файла в переменные. В данном случае аналогично функции save() – задаём в этот раз ifstream. В случае успешного открытия файла – с помощью цикла происходит считывание данных в случае, если переменная name не пустая.

## **Реализация функции load()**

*Листинг 4*

**void** Database::load()

{

ifstream input;

input.open("database.txt");

**if**(!input.is\_open())

{

cerr << "Error open database.txt" << endl;

exit(1);

}

**while**(!input.eof())

{

string name;

input >> name;

string lastname;

input >> lastname;

string flightnum;

input >> flightnum;

string date;

input >> date;

string time;

input >> time;

**if**(name != "")

{

addRecord(name,lastname,flightnum,date,time);

}

}

input.close();

}

Функция вывода и поиска совпадений реализована с помощью циклов и проверки на совпадения.

## **Реализация функции printDatabase()**

*Листинг 5.1*

**void** Database::printDatabase()

{

**for**(**int** i=0; i < count; ++i)

{

cout << i << "." << getNameById(i) << " " << getLastNameById(i) << " "<< getFlightNumById(i) << " " << getDateById(i) << " " << getTimeById(i) << endl;

}

}

## ***Реализация функции search()***

Листинг 5.2

**void** Database::search()

{

string data;

cout <<"Enter the some data from the ticket: ";

cin >> data;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**for**(**int** i=0; i < count; ++i)

{

**if** ((data == getNameById(i)) || (data == getLastNameById(i)) || (data == getFlightNumById(i)) || (data == getDateById(i)) || (data == getTimeById(i)))

{

cout << i << "." << getNameById(i) << " " << getLastNameById(i) << " "<< getFlightNumById(i) << " " << getDateById(i) << " " << getTimeById(i) << endl;

}

}

}

Самая важная функция любой базы данных – ввод этих самых данных, но эти самые данные обязаны пройти проверку на соответствие заданным шаблонам, чтобы избежать ошибок пользователей, которые будут не знакомы с функционалом программы.

Для того, чтобы пользователь не возвращался в самое начало, если будет допущена единичная ошибка в одной из переменных – был создан цикл. В случае, если все проверки будут пройдены – цикл прекратится. В противном случае , с помощью goto при выполнении else, будет произведён возврат в точку повторного ввода. Как только все проверки пройдены – происходит вызов заданной ранее функции addRecord, для записи полученных данных , а после – сохранения их в файл с помощью save() функции.

Для того, чтобы избежать принятия данных , которые пользователь случайно ввёл бы при записи переменной через пробел – была использована cin.ignore. Суть заключается в том, чтобы игнорировать все последующие введённые данные после первых. С помощью же find\_if – программа производит проверку имени и фамилии на наличие чисел, ибо в международно признанных нормах – числа не используются в именах. При прохождении проверки на числа – начинается проверка формата ввода номера билета. В данной БД форматом является запись типа “ХХХ-ХХХ-ХХХ”, возможно использование как чисел, так и букв или других специальных символов. Происходит сверка длины введенных данных, а так же наличие тире. Таким же образом реализована проверка остальных данных, как дата и врем, за исключением дополнительной проверки на наличие только чисел в записи, ибо буквы не могут быть задействованы в написании даты или времени.

## **Вызов функции AddRecord**

*Листинг 6.1*

**void** AddRecord(Database& server)

{

**bool** con = **true**;

**while**(con)

{

        nm:

cout << "Enter the passenger's first name: ";

string name;

cin >> name;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(find\_if(name.begin(), name.end(),(**int**(\*)(**int**))isdigit) == name.end())

{

            lm:

cout << "Enter the passengers's last name: ";

string lastname;

cin >> lastname;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(find\_if(lastname.begin(), lastname.end(),(**int**(\*)(**int**))isdigit) == lastname.end())

{

                jum1:

cout << "Enter the flight number (XXX-XXX-XXX) : ";

string flightnum;

cin >> flightnum;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(flightnum.length() == 11 && flightnum[3] == '-' && flightnum[7] == '-')

{

                    jum2:

cout << "Enter the date of the flight (DD-MM-YYYY): ";

string date;

cin >> date;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(date.length() == 10 && date[2] == '-' && date[5] == '-' && date[0] >= '0' && date[0] <= '9'&& date[1] >= '0' && date[1] <= '9'&& date[3] >= '0' && date[3] <= '9'&& date[4] >= '0' && date[4] <= '9'&& date[6] >= '0' && date[6] <= '9'&& date[7] >= '0' && date[7] <= '9'&& date[8] >= '0' && date[8] <= '9'&& date[9] >= '0' && date[9] <= '9')

{

                        jum3:

cout << "Enter the time of the flight (XX:XX): ";

string time;

cin >> time;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(time.length() == 5 && time[0] >= '0' && time[0] <= '9'&& time[1] >= '0' && time[1] <= '9' && time[2] == ':' && time[3] >= '0' && time[3] <= '9'&& time[4] >= '0' && time[4] <= '9')

{

server.addRecord(name,lastname,flightnum,date,time);

server.save();

con=**false**;

}

**else**

{

cout<<"Wrong data format"<<endl;

goto jum3;

}

}

**else**

{

cout<<"Wrong data format"<<endl;

goto jum2;

}

}

**else**

{

cout<<"Wrong data format"<<endl;

goto jum1;

}

}

**else**

{

cout<<"Enter the last name without numbers !"<<endl;

goto lm;

}

}

**else**

{

cout<<"Enter the first name without numbers !"<<endl;

goto nm;

}

}

}

Практически аналогичным образом реализована функция редактирования записи, за исключением появления переменной id , которая существует для определения, какую же запись программа должна отредактировать, а так же проверки этого самого id, путём подсчёта строк в файле , чтобы переменная не выходила за рамки и не содержала буквенные значения.

## **Вызов функции EditRecord**

*Листинг 6.2*

**void** EditRecord(Database& server)

{

    idi:

cout << "Enter the passenger's ID: ";

**int** id;

cin >> id;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(!cin)

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout<<"Only numbers are available"<<endl;

goto idi;

}

**int** numline = 0;

string line;

ifstream myfile("database.txt");

**if**(!myfile.is\_open())

{

cerr << "Error open database.txt" << endl;

goto idi;

}

**while** (getline(myfile, line)){++numline;}

**if**(id > -1 && id < numline)

{

**bool** con = **true**;

**while**(con)

{

            nm:

cout << "Enter the passenger's first name: ";

string name;

cin >> name;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(find\_if(name.begin(), name.end(),(**int**(\*)(**int**))isdigit) == name.end())

{

                lm:

cout << "Enter the passengers's last name: ";

string lastname;

cin >> lastname;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(find\_if(lastname.begin(), lastname.end(),(**int**(\*)(**int**))isdigit) == lastname.end())

{

                    jum1:

cout << "Enter the flight number (XXX-XXX-XXX) : ";

string flightnum;

cin >> flightnum;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(flightnum.length() == 11 && flightnum[3] == '-' && flightnum[7] == '-')

{

                        jum2:

cout << "Enter the date of the flight (DD-MM-YYYY): ";

string date;

cin >> date;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(date.length() == 10 && date[2] == '-' && date[5] == '-' && date[0] >= '0' && date[0] <= '9'&& date[1] >= '0' && date[1] <= '9'&& date[3] >= '0' && date[3] <= '9'&& date[4] >= '0' && date[4] <= '9'&& date[6] >= '0' && date[6] <= '9'&& date[7] >= '0' && date[7] <= '9'&& date[8] >= '0' && date[8] <= '9'&& date[9] >= '0' && date[9] <= '9')

{

                            jum3:

cout << "Enter the time of the flight (XX:XX): ";

string time;

cin >> time;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(time.length() == 5 && time[0] >= '0' && time[0] <= '9'&& time[1] >= '0' && time[1] <= '9' && time[2] == ':' && time[3] >= '0' && time[3] <= '9'&& time[4] >= '0' && time[4] <= '9')

{

server.editRecord(id,name,lastname,flightnum,date,time);

server.save();

con=**false**;

}

**else**

{

cout<<"Wrong data format"<<endl;

goto jum3;

}

}

**else**

{

cout<<"Wrong data format"<<endl;

goto jum2;

}

}

**else**

{

cout<<"Wrong data format"<<endl;

goto jum1;

}

}

**else**

{

cout<<"Enter the last name without numbers !"<<endl;

goto lm;

}

}

**else**

{

cout<<"Enter the first name without numbers !"<<endl;

goto nm;

}

}

}

**else**

{

cout<<"Incorrect ID"<<endl;

goto idi;

}

}

Функция удаления записи так же является неотъемлемой частью любой БД, чтобы была возможность избавиться от лишних или устаревших данных. Для начала, аналогично функции редактирования – проверка значения id на буквы и рамки действия. В случае прохождения всех проверок значения – производится замена данных нужной заявки на delete\_flag - %delete. В случае сохранения , данная строка в БД будет проигнорирована и удалена после перезапуска.

## **Реализация функции DeleteRecord**

*Листинг 7*

**void** DeleteRecord(Database& server)

{

    idd:

cout << "Enter the ID of the ticket: ";

**int** id;

cin >> id;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(!cin)

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout<<"Only numbers are available"<<endl;

goto idd;

}

**int** numline = 0;

string line;

ifstream myfile("database.txt");

**if**(!myfile.is\_open())

{

cerr << "Error open database.txt" << endl;

goto idd;

}

**while** (getline(myfile, line)){++numline;}

**if**(id > -1 && id < numline)

{

string delete\_flag = "%delete";

server.editRecord(id,delete\_flag, delete\_flag, delete\_flag, delete\_flag, delete\_flag);

server.save();

}

**else**

{

cout<<"Incorrect ID"<<endl;

goto idd;

}

}

То , без чего не вряд ли обойдется какая-либо база данных – интерфейс, в данном случае он является консольным, но также возможна его реализация в графическом виде, но для этого потребуются некоторые значительные изменения в изначальном коде.

Для начала вызывается функция загрузки базы данных, а так же создание цикла, чтобы отпала необходимость постоянного перезапуска программы, вместо этого написан кейс выхода , путём выхода из цикла меню. Как и было ранее – требуется проверка ввода на наличие букв или же сторонних символов.

## **Реализация меню консольного интерфейса**

*Листинг 8*

**void** menu(Database& server)

{

server.load();

    mn:

**bool** end = **false**;

**while**(!end)

{

cout << "----------BD's Menu---------" << endl << endl;

cout << "1. Record the new data" << endl;

cout << "2. Delete the data" << endl;

cout << "3. Edit the ticket's data" << endl;

cout << "4. Show all tickets" << endl;

cout << "5. Search for matches in the DB" << endl;

cout << "6. Exit" << endl;

cout << "\n\nSelect the function: ";

**int** answer;

cin>>answer;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if**(!cin)

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout<<"Wrong data type"<<endl;

system("pause");

system("cls");

goto mn;

}

**switch**(answer)

{

**case** 1:

AddRecord(server);

system("pause");

system("cls");

**break**;

**case** 2:

DeleteRecord(server);

system("pause");

system("cls");

**break**;

**case** 3:

EditRecord(server);

system("cls");

**break**;

**case** 4:

server.printDatabase();

system("pause");

system("cls");

**break**;

**case** 5:

server.search();

system("pause");

system("cls");

**break**;

**case** 6:

end = **true**;

**break**;

**default**:

cout << "Wrong num " << endl;

system("pause");

system("cls");

**break**;

}

}

}

На последок – функция main, где задаётся переменная server, а так же menu(server) для работы функций.

## **Реализация main()**

*Листинг 9*

**int** main()

{

Database server;

menu(server);

**return** 0;

}

## **ЗАКЛЮЧЕНИЕ**

В ходе курсовой работы были получены теоретические и практические навыки по созданию баз данных с использованием языка C++. Написана программа, позволяющая работать с базой данных , которая хранится в txt файле, имеющее следующие функции:

− извлечение записи;

− добавление записи;

− удаление записи;

− редактирование записи;

− поиска и вывода записей;

Курсовая работа была выполнена в трех этапах:

− изучение теоретического материала;

− проектирование кода базы данных;

− создание программы.

## **Источники**

1. Документация по .NET https://docs.microsoft.com/ru-ru/dotnet/
2. Руководство по Windows Forms <https://metanit.com/sharp/windowsforms/>
3. Система вопросов и ответов о программировании <https://stackoverflow.com/>
4. Система блогов, связанных с информационными технологиями <https://habr.com/>
5. Блог на сайте Oracle о базах данных https://www.oracle.com/ru/database/
6. Осипов Д. С. Технологии проектирования баз данных. – М.: ДМК Пресс, 2019. – 498 с.: ил.
7. Сергей Кузнецов. «Базы данных. Вводный курс». http://citforum.ru/database/advanced\_intro/