**Содержание**

[Введение 4](#_Toc65347889)

[1. Постановка задачи 5](#_Toc65347890)

[1.1. Описание предметной области 5](#_Toc65347891)

[1.2. Функциональное назначение 6](#_Toc65347892)

[2. Проектирование реляционных баз данных 7](#_Toc65347893)

[2.1. Описание инструментов разработки 7](#_Toc65347894)

[2.2. Концептуальное проектирование 12](#_Toc65347895)

[2.3. Физическое проектирование БД 15](#_Toc65347896)

[3. Программная реализация 31](#_Toc65347897)

[3.1. Описание разработанных компонентов базы данных 31](#_Toc65347898)

[3.2. Описание разработанных модулей приложения 33](#_Toc65347899)

[4. Тестирование 42](#_Toc65347900)

[Заключение 45](#_Toc65347901)

[Список использованных источников 46](#_Toc65347902)

# Введение

Важнейшая задача компьютерных систем – хранение и обработка данных. Для её решения были предприняты усилия, которые привели к появлению в конце 60-х – начале 70-х годов специализированного программного обеспечения – систем управления базами данных (database management systems).[1] СУБД позволяют структурировать, систематизировать и организовать данные для их компьютерного хранения и обработки. Невозможно представить себе деятельность современного предприятия или учреждения без использования профессиональных СУБД. Несомненно, они составляют фундамент информационной деятельности во всех сферах – начиная с производства и заканчивая финансами и телекоммуникациями.

Курсовой проект посвящен изучению теории и практики разработки и проектирования распределённых баз данных. В данном курсовом проекте объектом исследования является среда MicrosoftStudio, как платформа для разработки приложений использования баз данных в информационных системах.

Предметом исследования является система организации и ведения распределённых баз данных в Интернет. Целью курсового проекта является разработка приложения автоматизации рабочего места диспетчера железнодорожного вокзала.

Курсовой проект предполагает выполнение следующих задач:

спроектировать базу данны, описать предметную область, создать модели БД;

разработать программу для работы с созданной базой данных.

В качестве практической части в рамках курсового проекта создаётся база данных и приложение для ее работы с использованием технологий языка программирования C# и СУБД MySQL.

# Постановка задачи

## Описание предметной области

Железнодорожный транспорт играет большую роль в стране. Он остаётся самым доходным для государства и самым популярным у граждан нашей страны. По сравнению с другими видами транспорта, эти перевозки сравнительно недороги и менее опасны, чем, скажем, перевозки воздушным транспортом [2].

Железные дороги – это развитая сеть с большим количеством сотрудников. Территория нашей страны обширна, перемещение по ней удобно и быстро осуществляется по средствам железных дорог. При этих условиях необходимо организовывать слаженную деятельность железнодорожных касс на высоком уровне: автоматизировать процессы, связанные с финансовой стороной вопроса (продажа билетов), использовать удобные для пассажиров и кассиров формы представления информации. Для этого для сотрудников касс разрабатываются специальные системы управления - базы данных, в которых хранится вся информация о поездах, билетах и пассажирах.

Железнодорожные кассы являются предприятием, которое ежедневно работает с большим числом клиентов. Работают с клиентами сотрудники, которые разделены по должностям.

Основным элементом вокзала являются маршруты. Они состоят из названия маршрутов, даты отправки и прибытия в конечное место, а также пунктов остановки и поезда, который этот маршрут проезжает.

На каждый маршрут собираются пассажиры, учет которых необходимо вести как единично, так и при приобретении билета на маршрут.

## Функциональное назначение

В данном разделе необходимо описать функциональное назначение курсового проекта, а именно постановка цели, описание задач для достижения поставленной це-ли, перечисление функций разрабатываемого приложения.

Целью курсового проекта является разработка базы данных для автоматизации рабочего места диспетчера железнодорожного вокзала и приложения, для использования созданной базы данных.

Для достижения заданной цели, необходимо выполнить множество задач:

* Спроектировать базу данных
* Выбрать инструменты разработки приложения
* Разработать приложение
* Протестировать разработанное приложение

Полученная программа должна выполнять следующие функции

* Просмотр, добавление, изменение и удаление должностей
* Просмотр, добавление, изменение и удаление сотрудников
* Просмотр, добавление, изменение и удаление пассажиров
* Просмотр, добавление, изменение и удаление поездов
* Просмотр, добавление, изменение и удаление областей
* Просмотр, добавление, изменение и удаление районов
* Просмотр, добавление, изменение и удаление пунктов назначения
* Просмотр, добавление, изменение и удаление маршрутов
* Просмотр, добавление, изменение и удаление пунктов каждого маршрута
* Просмотр, добавление, изменение и удаление билетов пассажиров

# Проектирование реляционных баз данных

## Описание инструментов разработки

Для написания приложения была выбрана среда разработки Visual Studio 2019, язык программирования C# и СУБД для работы с данными MySQL Server, краткое описание выбранной среды разработки, языка программирования и СУБД приведена ниже.

Для реализации клиентского приложения выбран язык объектно-ориентированного программирования C#, среда разработки Visual Studio. Microsoft Visual Studio— средство для разработчиков ПО, которое позволяет решать основные задачи разработки: система упрощает создание, отладку и развёртывание приложений на различных платформах, включая облачную среду.[3]

Основными преимуществами Visual Studio являются:

* использование вычислительных мощностей локального компьютера и облака;
* простая реализация общих задач и индивидуальный подход;
* быстрое создание высококачественного кода;
* функция поддержки нескольких мониторов;
* возможность реализации идей и решений для широкого спектра платформ, включая Windows, Windows Server, веб-среду, облачную среду, Office и SharePoint.

Microsoft Visual Studio 2019 - это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию. Эти инструменты предназначены для максимально эффективной совместной работы; все они доступны в интегрированной среде разработки (IDE) Visual Studio.

Visual Studio можно использовать для создания различных типов приложений, от простых приложений для магазина и игр для мобильных клиентов до больших и сложных систем, обслуживающих предприятия и центры обработки данных. Вы можете создавать:

* приложения и игры, которые выполняются не только на платформе Windows, но и на Android и iOS;
* веб-сайты и веб-службы на основе ASP.NET, JQuery, AngularJS и других популярных платформ;
* приложения для самых разных платформ и устройств, включая, но не ограничиваясь: Office, Sharepoint, Hololens, Kinect и "Интернета вещей";
* игры и графические приложения для разных устройств Windows, включая Xbox, с поддержкой DirectX.

По умолчанию Visual Studio обеспечивает поддержку C#, C и C++, JavaScript, F# и Visual Basic. Visual Studio хорошо работает и интегрируется со сторонними приложениями, например, Unity и Apache Cordova, с помощью расширений Набор средств Visual Studio для Unity и инструментов Visual Studio для Apache Cordova соответственно. Вы также можете самостоятельно расширить Visual Studio, создав собственные инструменты для выполнения специализированных задач.

Visual Studio включает один или несколько из следующих компонентов:

Visual Basic .NET, а до его появления - Visual Basic, Visual C++, Visual C#, Visual J#.

Многие варианты поставки также включают: Microsoft SQL Server либо MSDE, Visual Source Safe - файл-серверная система управления версиями.

В прошлом, в состав Visual Studio также входили продукты: Visual InterDev, Visual J++, Visual FoxPro.

Visual Basic .NET (VB.NET) - это объектно-ориентированный язык программирования, который можно рассматривать как очередной виток эволюции Visual Basic (VB), реализованный на платформе Microsoft .NET.

В отличие от «классического» VB, VB.NET - полностью объектно-ориентированный язык программирования, поддерживающий полиморфизм, наследование и другие ключевые возможности ООП.

VB.NET не имеет обратной совместимости с VB6. Открытие проектов старых версий (\*.vbp) возможно только после предварительной конвертации их в формат VB.NET специальным мастером (Migration Wizard).

Важнейшими изменениями в VB.NET являются:

* поддержка концепций объектно-ориентированного программирования;
* компиляция в байт-код (intermediate language, IL), исполняемый с помощью виртуальной машины common language runtime (CLR);
* использование всего набора объектных библиотек, входящих в .NET Framework, включающих мощные средства по работе с формами (Windows Forms), базами данных (ADO.NET), графикой (GDI+), средствами обеспечения безопасности, веб-страницами (ASP.NET) и тому подобные;
* появление жёсткой проверки типов (Option Strict);
* поддержка свободной многопоточности;
* поддержка структурной обработки исключений (structurized exception handling, SEH).

С переходом на платформу .NET, изменилась сама парадигма программирования (и это наиболее важное изменение).

Существующие версии VB.NET:

* Visual Basic .NET - выпущен в 2002 году, вместе с Visual C# и ASP.NET;
* Visual Basic .NET 2003 - выпущена вместе с .NET Framework 1.1; имеется русскоязычная версия с русским MSDN;
* Visual Basic 2005 - выпущена в конце октября 2005 года, в составе Visual Studio 2005;
* Visual Basic 2005 Express - выпущена в начале ноября 2005 года совместно с другими продуктами линейки Express, в том числе SQL Server 2005 Express Edition. Предлагается для бесплатного скачивания и использования. Требует .NET Framework 2.0.

Microsoft Visual C++ (MSVC) - интегрированная среда разработки приложений на языке C++, разработанная фирмой Microsoft и поставляемая либо как часть комплекта Microsoft Visual Studio, либо отдельно в виде функционально ограниченного комплекта Visual C++ Express Edition.

Visual C++ .NET поддерживает разработку приложений как на Managed C++, так и на обычном (неуправляемом) C++, и тем самым позволяет генерировать код как для платформы .NET Framework, так и для исполнения в среде «чистой» Windows. В этом отношении Visual C++ .NET является уникальным среди других языковых средств, предоставляемых средой Visual Studio, поскольку ни Visual Basic .NET, ни Visual J# не способны генерировать код для чистого Win32, в отличие от предыдущих версий (Visual Basic и Visual J++ соответственно).

C# (произносится си-шарп) - язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998-2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде Visual Studio.[4]

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. Переняв многое от своих предшественников - языков C++, Delphi, Modula и Smalltalk - С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++) или вывода типов (в отличие от Haskell).

Система содержит Библиотеку из более 100 повторно используемых визуальных компонент, которые перетаскиваются мышью на форму и сразу становятся элементами управления прототипа вашей программы. Помимо известных элементов управления Windows (кнопки, линейки прокрутки, поля редактирования, простые и комбинированные списки и т.д.) Библиотека содержит новые компоненты поддержки диалогов, обслуживания баз данных и многие другие

После размещения компонент на форме. Инспектор объектов поможет вам устанавливать их свойства и предписывать событиям коды обработки. Ваш проект будет строиться постепенно, на фоне производимых вами изменений в свойствах, событиях и функциях используемых элементов. Хорошо продумано разделение и редактирование программного модуля по двум его частям: интерфейсной и собственно кодовой.

Для реализации задачи по разработке информационной системы продовольственного магазина была выбрана СУБД Mу SQL Server. Данная СУБД является одной из наиболее популярных систем управления и обслуживанию баз данных. В ней применяются новейшие разработки и последние достижения в области проектирования, построения и обслуживания баз данных. Так же My SQL Server часто выступает в роли стандартного источника данных для решений Microsoft, что облегчает его интеграцию с различными программными пакетами.[5]

Таким образом, My SQL Server - одна из наиболее мощных систем работы с базами данных в архитектуре "клиент-сервер". СУБД позволяет создавать базы данных как для мелких организаций, так и для целых корпораций.

## Концептуальное проектирование

Основными задачами концептуального проектирования являются определение предметной области системы и формирование взгляда на предметную область с позиций сообщества будущих пользователей БД, т. е. инфологической модели предметной области. [6]

Концептуальная модель предметной области представляет собой описание структуры и динамики предметной области, характера информационных потребностей пользователей в терминах, понятных пользователю и не зависимых от реализации БД. Это описание выражается в терминах не отдельных объектах предметной области и связей между ними, а их типов, связанных с ними ограничений целостности и тех процессов, которые приводят к переходу предметной области из одного состояния в другое.

Рассмотрим основные подходы к созданию концептуальной модели предметной области:

1. Функциональный подход к проектированию БД.

Этот метод реализует принцип "от задач" и применяется тогда, когда известны функции некоторой группы лиц и/или комплекса задач, для обслуживания информационных потребностей которых создаётся рассматриваемая БД.

2. Предметный подход к проектированию БД.

Предметный подход к проектированию БД применяется в тех случаях, когда у разработчиков есть чёткое представление о самой предметной области и о том, какую именно информацию они хотели бы хранить в БД, а структура запросов не определена или определена не полностью. Тогда основное внимание уделяется исследованию предметной области и наиболее адекватному её отображению в БД с учётом самого широкого спектра информационных запросов к ней.

3. Проектирование с использованием метода «сущность-связь»

Метод "сущность–связь" (entity–relation, ER–method) является комбинацией двух предыдущих и обладает достоинствами обоих. Этап инфологического проектирования начинается с моделирования предметной области. Проектировщик разбивает её на ряд локальных областей, каждая из которых (в идеале) включает в себя информацию, достаточную для обеспечения запросов отдельной группы будущих пользователей или решения отдельной задачи (подзадачи). Каждое локальное представление моделируется отдельно, затем они объединяются.

Выбор локального представления зависит от масштабов предметной области. Обычно она разбивается на локальные области таким образом, чтобы каждая из них соответствовала отдельному внешнему приложению и содержала 6 - 7 сущностей.

Сущность – это объект, о котором в системе будет накапливаться информация. Сущности бывают как физически существующие (например, СОТРУДНИК или АВ-ТОМОБИЛЬ), так и абстрактные (например, ЭКЗАМЕН или ДИАГНОЗ).

Для сущностей различают тип сущности и экземпляр. Тип характеризуется именем и списком свойств, а экземпляр – конкретными значениями свойств.

Типы сущностей можно классифицировать как сильные и слабые. Сильные сущности существуют сами по себе, а существование слабых сущностей зависит от существования сильных. Например, читатель библиотеки – сильная сущность, а абонемент этого читателя – слабая, которая зависит от наличия соответствующего читателя. Слабые сущности называют подчинёнными (дочерними), а сильные – базовыми (основными, родительскими).

Для каждой сущности выбираются свойства (атрибуты):

Идентифицирующие и описательные атрибуты. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются потенциальными ключами. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. Кроме того, ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными и заключают в себе интересующие свойства сущности.

Составные и простые атрибуты. Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, ФИО или адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от характера его обработки и формата пользовательского представления этого атрибута.

Однозначные и многозначные атрибуты (могут иметь соответственно одно или много значений для каждого экземпляра сущности).

Основные и производные атрибуты. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст учащегося вычисляется на основе даты его рождения и текущей даты).

Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности – множества значений (или домена), которые может принимать данный атрибут.

Далее осуществляется спецификация связей внутри локального представления. Связи могут иметь различный содержательный смысл (семантику). Различают связи типа «сущность – сущность», «сущность - атрибут» и «атрибут - атрибут» для отношений между атрибутами, которые характеризуют одну и ту же сущность или одну и туже связь типа «сущность – сущность».

Каждая связь характеризуется именем, обязательностью, типом и степенью. Различают факультативные и обязательные связи. Если вновь порождённый объект одного типа оказывается по необходимости связанным с объектом другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

По типу различают множественные связи «один к одному» (1:1), «один ко многим» (1:N) и «многие ко многим» (M:N).

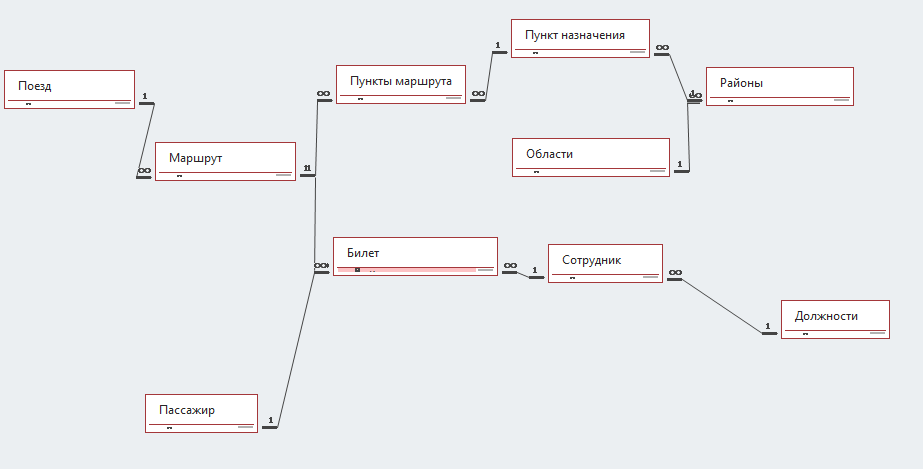


Рисунок 2.1 – Схема отношений сущностей

## Физическое проектирование БД

Этап физического проектирования заключается в увязке логической структуры БД и физической среды хранения с целью наиболее эффективного размещения данных, т. е. отображении логической структуры БД в структуру хранения. Решается вопрос размещения хранимых данных в пространстве памяти, выбора эффективных методов доступа к различным компонентам «физической» БД. Результаты этого этапа документируются в форме схемы хранения на языке определения данных (DDL). Принятые на этом этапе решения оказывают определяющее влияние на производительность системы. [7] Между физическим и логическим проектированием существует обратная связь, так как иногда с целью повышения эффективности приходится менять структуру БД. Это возникает, если полная нормализация данных не позволяет достичь требуемой эффективности обработки информации. Принимается решение о денормализации отношений. При этом следует учитывать следующее:

денормализация усложняет реализацию БД;

денормализация снижает гибкость системы;

денормализация ускоряет выборку данных, но снижается скорость обновле-ния данных.

Иногда денормализацию называют оптимизацией исполнения.

Одной из важнейших составляющих проекта базы данных является разработка средств защиты БД. Защита данных имеет два аспекта: защита от сбоев и защита от несанкционированного доступа. Для защиты от сбоев разрабатывается стратегия резервного копирования. Для защиты от несанкционированного доступа каждому пользователю доступ к данным предоставляется только в соответствии с его правами доступа.

Итогом физического проектирования должно быть представление метаданные всех таблиц. Полученная схема базы данных отображена на рисунке 2.2.

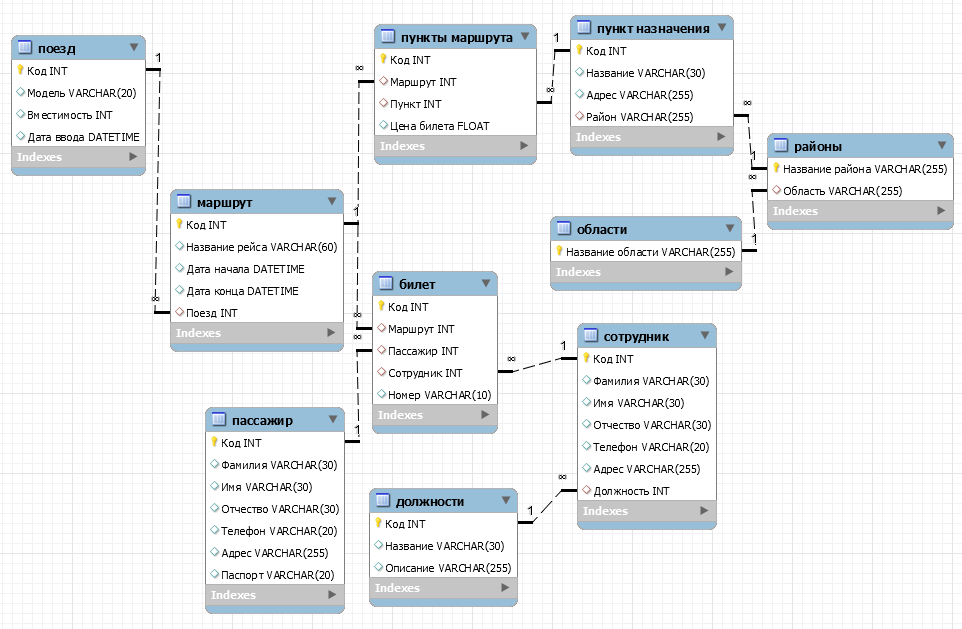


Рисунок 2.2 – Схема базы данных

Код создания базы данных представлен ниже.

Листинг1 – код создания базы данных

DROP DATABASE IF EXISTS `AisOfTheRailwayStationDispatcher`;

CREATE DATABASE IF NOT EXISTS `AisOfTheRailwayStationDispatcher`;

USE `AisOfTheRailwayStationDispatcher`;

#

# Table structure for table 'Должности'

#

DROP TABLE IF EXISTS `Должности`;

CREATE TABLE `Должности` (

`Код` INTEGER AUTO\_INCREMENT,

`Название` VARCHAR(30),

`Описание` VARCHAR(255),

PRIMARY KEY (`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Должности'

#

INSERT INTO `Должности` (`Код`, `Название`, `Описание`) VALUES (1, 'Диспетчер', 'В основном назначет билеты');

INSERT INTO `Должности` (`Код`, `Название`, `Описание`) VALUES (2, 'Проводник', 'Иногда назначает билеты, а так разносит напитки');

INSERT INTO `Должности` (`Код`, `Название`, `Описание`) VALUES (3, 'Работник', NULL);

# 3 records

#

# Table structure for table 'Сотрудник'

#

DROP TABLE IF EXISTS `Сотрудник`;

CREATE TABLE `Сотрудник` (

`Код` INTEGER AUTO\_INCREMENT,

`Фамилия` VARCHAR(30),

`Имя` VARCHAR(30),

`Отчество` VARCHAR(30),

`Телефон` VARCHAR(20),

`Адрес` VARCHAR(255),

`Должность` INTEGER,

PRIMARY KEY (`Код`),

FOREIGN KEY(`Должность`) REFERENCES `Должности`(`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Сотрудник'

#

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (1, 'Круглова', 'Екатерина', 'Юрьевна', '093-22-69', '123458, Москва, ул.3-я Лыковская, 62, кв.52', 1);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (2, 'Гордеева', 'Арина', 'Егоровна', '759-96-55', '111625, Москва, ул.Розы Люксембург, 58, кв.72', 2);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (3, 'Гончаров', 'Роман', 'Тимофеевич', '983-82-91', '117292, Москва, ул.Львова, 56, кв.63', 3);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (4, 'Журавлева', 'Сафия', 'Александровна', '601-68-19', '105118, Москва, ул.10-я Соколиной Горы, 81, кв.87', 2);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (5, 'Плотников', 'Лука', 'Даниилович', '021-29-16', '125080, Москва, ул.Дубосековская, 96, кв.89', 1);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (6, 'Тарасов', 'Арсений', 'Максимович', '756-41-11', '119332, Москва, ул.Московская, 57, кв.73', 3);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (7, 'Макарова', 'Полина', 'Савельевна', '049-24-06', '119620, Москва, ул.6-я Прудная, 92, кв.68', 1);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (8, 'Безруков', 'Артём', 'Всеволодович', '076-88-57', '111621, Москва, ул.Оренбургская, 64, кв.31', 1);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (9, 'Фролова', 'Мелания', 'Алексеевна', '891-70-74', '123060, Москва, ул.Маршала Конева, 63, кв.36', 3);

INSERT INTO `Сотрудник` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Должность`) VALUES (10, 'Лукьянова', 'Мария', 'Елисеевна', '311-42-35', '127411, Москва, ул.Яхромская, 21, кв.25', 2);

# 10 records

#

# Table structure for table 'Области'

#

DROP TABLE IF EXISTS `Области`;

CREATE TABLE `Области` (

`Название области` VARCHAR(255) NOT NULL,

PRIMARY KEY (`Название области`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Области'

#

INSERT INTO `Области` (`Название области`) VALUES ('Минская');

INSERT INTO `Области` (`Название области`) VALUES ('Могилевкая');

INSERT INTO `Области` (`Название области`) VALUES ('Гродненская');

INSERT INTO `Области` (`Название области`) VALUES ('Гомельская');

INSERT INTO `Области` (`Название области`) VALUES ('Брестская');

INSERT INTO `Области` (`Название области`) VALUES ('Витебская');

# 6 records

#

# Table structure for table 'Районы'

#

DROP TABLE IF EXISTS `Районы`;

CREATE TABLE `Районы` (

`Название района` VARCHAR(255) NOT NULL,

`Область` VARCHAR(255),

PRIMARY KEY (`Название района`),

FOREIGN KEY(`Область`) REFERENCES `Области`(`Название области`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Районы'

#

INSERT INTO `Районы` (`Название района`, `Область`) VALUES ('Минский', 'Минская');

INSERT INTO `Районы` (`Название района`, `Область`) VALUES ('Молодечно', 'Витебская');

INSERT INTO `Районы` (`Название района`, `Область`) VALUES ('Столбческий', 'Минская');

INSERT INTO `Районы` (`Название района`, `Область`) VALUES ('Тракторский', 'Могилевкая');

# 4 records

#

# Table structure for table 'Пункт назначения'

#

DROP TABLE IF EXISTS `Пункт назначения`;

CREATE TABLE `Пункт назначения` (

`Код` INTEGER AUTO\_INCREMENT,

`Название` VARCHAR(30),

`Адрес` VARCHAR(255),

`Район` VARCHAR(255),

PRIMARY KEY (`Код`),

FOREIGN KEY(`Район`) REFERENCES `Районы`(`Название района`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Пункт назначения'

#

INSERT INTO `Пункт назначения` (`Код`, `Название`, `Адрес`, `Район`) VALUES (1, 'Азино', NULL, 'Минский');

INSERT INTO `Пункт назначения` (`Код`, `Название`, `Адрес`, `Район`) VALUES (2, 'Молодечно', NULL, 'Молодечно');

INSERT INTO `Пункт назначения` (`Код`, `Название`, `Адрес`, `Район`) VALUES (3, 'Вототище', NULL, 'Минский');

INSERT INTO `Пункт назначения` (`Код`, `Название`, `Адрес`, `Район`) VALUES (4, 'Столбцы', NULL, 'Столбческий');

INSERT INTO `Пункт назначения` (`Код`, `Название`, `Адрес`, `Район`) VALUES (5, 'Хвоево', NULL, 'Столбческий');

INSERT INTO `Пункт назначения` (`Код`, `Название`, `Адрес`, `Район`) VALUES (6, 'Турбен', NULL, 'Тракторский');

# 6 records

#

# Table structure for table 'Поезд'

#

DROP TABLE IF EXISTS `Поезд`;

CREATE TABLE `Поезд` (

`Код` INTEGER AUTO\_INCREMENT,

`Модель` VARCHAR(20),

`Вместимость` INTEGER,

`Дата ввода` DATETIME,

PRIMARY KEY (`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Поезд'

#

INSERT INTO `Поезд` (`Код`, `Модель`, `Вместимость`, `Дата ввода`) VALUES (1, '84', 720, '2000-02-05 00:00:00');

INSERT INTO `Поезд` (`Код`, `Модель`, `Вместимость`, `Дата ввода`) VALUES (2, '984', 800, '2000-02-05 00:00:00');

INSERT INTO `Поезд` (`Код`, `Модель`, `Вместимость`, `Дата ввода`) VALUES (3, '648', 720, '2000-02-05 00:00:00');

# 3 records

#

# Table structure for table 'Пассажир'

#

DROP TABLE IF EXISTS `Пассажир`;

CREATE TABLE `Пассажир` (

`Код` INTEGER AUTO\_INCREMENT,

`Фамилия` VARCHAR(30),

`Имя` VARCHAR(30),

`Отчество` VARCHAR(30),

`Телефон` VARCHAR(20),

`Адрес` VARCHAR(255),

`Паспорт` VARCHAR(20),

PRIMARY KEY (`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Пассажир'

#

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (1, 'Кондрашова', 'София', 'Владиславовна', '800-85-23', '107014, г.Москва, ул.Б.Тихоновская, д.33, кв.58', '4025 177240');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (2, 'Голубев', 'Артём', 'Макарович', '521-37-33', '107014, г.Москва, ул.5-я Сокольническая, д.32, кв.88', '4831 276205');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (3, 'Киселева', 'Елизавета', 'Марковна', '168-73-55', '119332, г.Москва, ул.Матросова, д.2, кв.98', '4832 261927');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (4, 'Смирнова', 'Валерия', 'Константиновна', '950-75-43', '121293, г.Москва, ул.Дениса Давыдова, д.87, кв.25', '4959 566217');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (5, 'Кравцова', 'Екатерина', 'Александровна', '442-06-55', '119634, г.Москва, ул.Приречная, д.88, кв.40', '4198 725508');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (6, 'Никитина', 'Лейла', 'Сергеевна', '875-50-92', '123060, г.Москва, ул.Маршала Соколовского, д.65, кв.60', '4748 392631');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (7, 'Сизов', 'Владимир', 'Сергеевич', '619-07-47', '143350, г.Москва, ул.Майская, д.69, кв.14', '4229 159229');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (8, 'Белова', 'Анна', 'Борисовна', '083-52-79', '119361, г.Москва, ул.Наташи Ковшовой, д.98, кв.42', '4650 768608');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (9, 'Макаров', 'Николай', 'Мирославович', '217-97-31', '109235, г.Москва, ул.Батюнинская, д.68, кв.18', '4791 800548');

INSERT INTO `Пассажир` (`Код`, `Фамилия`, `Имя`, `Отчество`, `Телефон`, `Адрес`, `Паспорт`) VALUES (10, 'Баранов', 'Артемий', 'Матвеевич', '654-15-18', '113054, г.Москва, ул.Зацепский Вал, д.29, кв.70', '4467 670030');

# 10 records

#

# Table structure for table 'Маршрут'

#

DROP TABLE IF EXISTS `Маршрут`;

CREATE TABLE `Маршрут` (

`Код` INTEGER AUTO\_INCREMENT,

`Название рейса` VARCHAR(60),

`Дата начала` DATETIME,

`Дата конца` DATETIME,

`Поезд` INTEGER,

PRIMARY KEY (`Код`),

FOREIGN KEY(`Поезд`) REFERENCES `Поезд`(`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Маршрут'

#

INSERT INTO `Маршрут` (`Код`, `Название рейса`, `Дата начала`, `Дата конца`, `Поезд`) VALUES (1, 'Минск-Могилев', '2021-02-26 00:00:00', '2021-02-27 00:00:00', 1);

INSERT INTO `Маршрут` (`Код`, `Название рейса`, `Дата начала`, `Дата конца`, `Поезд`) VALUES (2, 'Могилев-Минск', '2021-02-27 00:00:00', '2021-02-27 00:00:00', 2);

# 2 records

#

# Table structure for table 'Пункты маршрута'

#

DROP TABLE IF EXISTS `Пункты маршрута`;

CREATE TABLE `Пункты маршрута` (

`Код` INTEGER AUTO\_INCREMENT,

`Маршрут` INTEGER,

`Пункт` INTEGER,

`Цена билета` FLOAT NULL,

PRIMARY KEY (`Код`),

FOREIGN KEY(`Маршрут`) REFERENCES `Маршрут`(`Код`),

FOREIGN KEY(`Пункт`) REFERENCES `Пункт назначения`(`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Пункты маршрута'

#

INSERT INTO `Пункты маршрута` (`Код`, `Маршрут`, `Пункт`, `Цена билета`) VALUES (1, 1, 2, 10);

INSERT INTO `Пункты маршрута` (`Код`, `Маршрут`, `Пункт`, `Цена билета`) VALUES (2, 1, 4, 20);

INSERT INTO `Пункты маршрута` (`Код`, `Маршрут`, `Пункт`, `Цена билета`) VALUES (3, 1, 5, 30);

INSERT INTO `Пункты маршрута` (`Код`, `Маршрут`, `Пункт`, `Цена билета`) VALUES (4, 2, 5, 10);

INSERT INTO `Пункты маршрута` (`Код`, `Маршрут`, `Пункт`, `Цена билета`) VALUES (5, 2, 4, 20);

INSERT INTO `Пункты маршрута` (`Код`, `Маршрут`, `Пункт`, `Цена билета`) VALUES (6, 2, 2, 30);

# 6 records

#

# Table structure for table 'Билет'

#

DROP TABLE IF EXISTS `Билет`;

CREATE TABLE `Билет` (

`Код` INTEGER AUTO\_INCREMENT,

`Маршрут` INTEGER,

`Пассажир` INTEGER,

`Сотрудник` INTEGER,

`Номер` VARCHAR(10),

PRIMARY KEY (`Код`),

FOREIGN KEY(`Маршрут`) REFERENCES `Маршрут`(`Код`),

FOREIGN KEY(`Пассажир`) REFERENCES `Пассажир`(`Код`),

FOREIGN KEY(`Сотрудник`) REFERENCES `Сотрудник`(`Код`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

SET autocommit=1;

#

# Dumping data for table 'Билет'

#

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (1, 1, 1, 1, '1а12');

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (2, 1, 2, 4, '1а13');

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (3, 1, 4, 5, '1а43');

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (4, 1, 8, 7, 'и22');

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (5, 2, 2, 2, 'и24');

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (6, 2, 3, 6, 'и206');

INSERT INTO `Билет` (`Код`, `Маршрут`, `Пассажир`, `Сотрудник`, `Номер`) VALUES (7, 2, 10, 6, 'и800');

# 7 records

# Программная реализация

## Описание разработанных компонентов базы данных

В таблицах 2.1 – 2.10 приведено описание состава таблиц спроектированной базы данных.

Таблица 2.1 – Билет

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Маршрут | Число | Нет | - |
| Нет | Пассажир | Число | Нет | - |
| Нет | Сотрудник | Число | Нет | - |
| Нет | Номер | Строка | Нет | 10 |

Таблица 2.2 – Должности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Название | Строка | Нет | 30 |
| Нет | Описание | Строка | Нет | 255 |

Таблица 2.3 – Маршрут

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Id | Число | Нет | - |
| Нет | Название рейса | Строка | Нет | 30 |
| Нет | Дата начала | Дата | Нет | - |
| Нет | Дата конца | Дата | Нет | - |
| Нет | Поезд | Число | Нет | - |

Таблица 2.4 – Области

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Название области | Строка | Нет | 255 |

Таблица 2.5 – Пассажир

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Фамилия | Строка | Нет | 30 |
| Нет | Имя | Строка | Нет | 30 |
| Нет | Отчество | Строка | Нет | 30 |
| Нет | Телефон | Строка | Нет | 20 |
| Нет | Адрес | Строка | Нет | 255 |
| Нет | Паспорт | Строка | Нет | 20 |

Таблица 2.6 – Поезд

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Модель | Строка | Нет | 30 |
| Нет | Вместимость | Число | Нет | - |
| Нет | Дата ввода | Дата | Нет | - |

Таблица 2.7 – Пункт назначения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Название | Строка | Нет | 30 |
| Нет | Адрес | Строка | Нет | 255 |
| Нет | Район | Число | Нет | - |

Таблица 2.8 – Пункты маршрута

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Маршрут | Число | Нет | - |
| Нет | Пункт | Число | Нет | - |
| Нет | Цена билета | Число | Нет | - |

Таблица 2.9 – Районы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Название района | Строка | Нет | 255 |
| Нет | Область | Строка | Нет | 255 |

Таблица 2.10 – Сотрудник

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Размер |
| PK | Код | Число | Нет | - |
| Нет | Фамилия | Строка | Нет | 30 |
| Нет | Имя | Строка | Нет | 30 |
| Нет | Отчество | Строка | Нет | 30 |
| Нет | Телефон | Строка | Нет | 20 |
| Нет | Адрес | Строка | Нет | 255 |
| Нет | Должность | Число | Нет | 30 |

## Описание разработанных модулей приложения

Приложение состоит из десяти форм, каждая из которых отвечает за доступ к своему модулю. Диаграмма классов показана на рисунке 3.1.

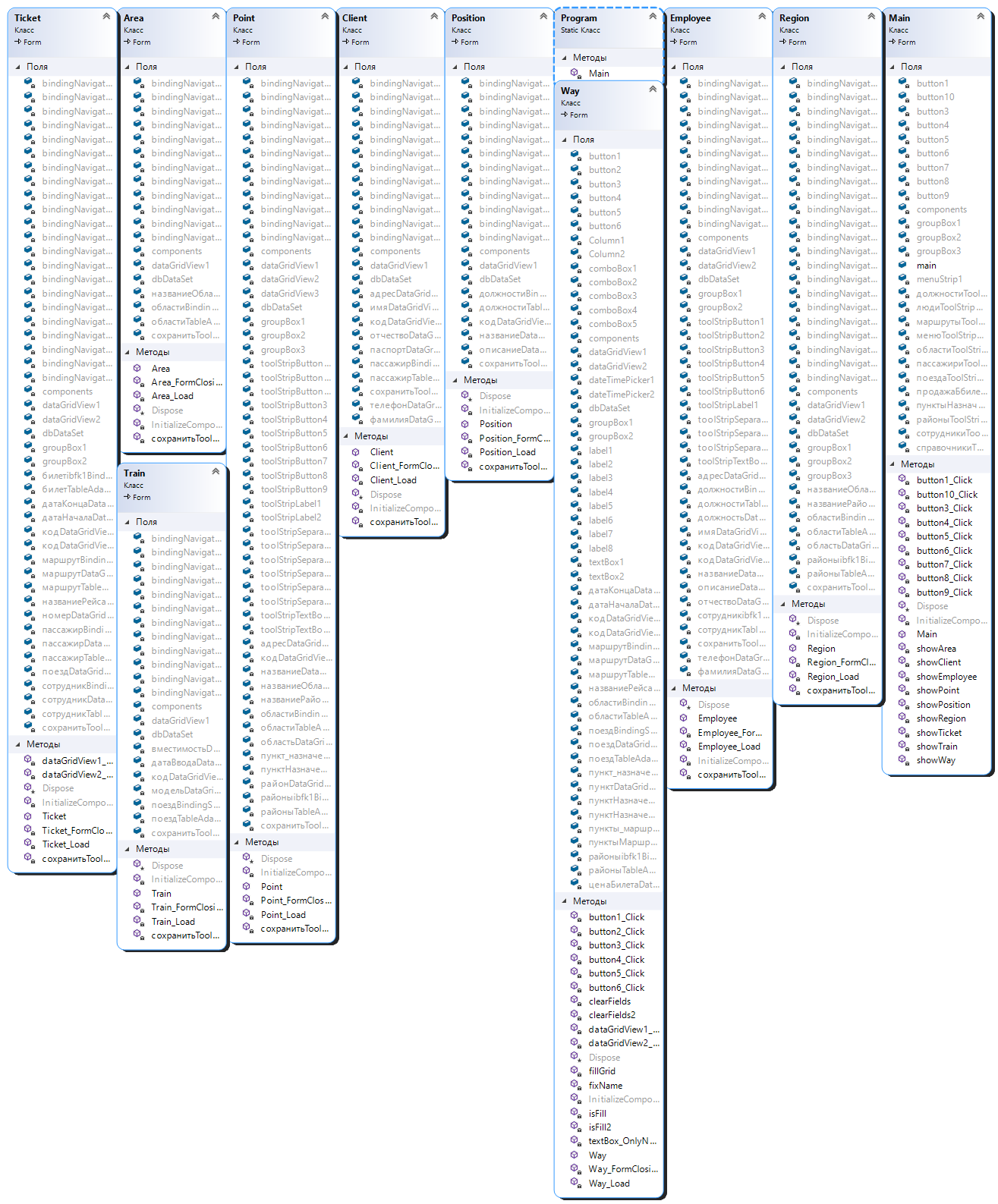
****

Рисунок 3.1 – диаграмма классов

Главный модуль представлен на рисунке 3.2.

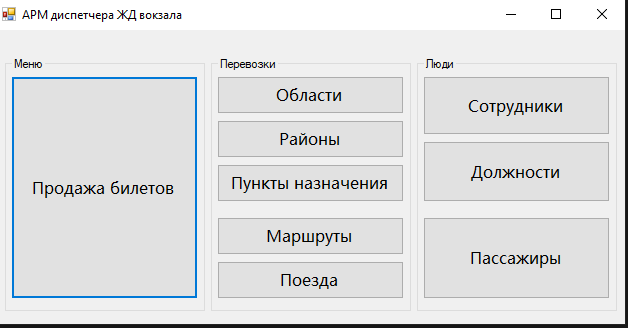


Рисунок 3.2 – Главная форма

На этой форме пользователь может перейти к любой другой или выйти из приложения. Для удобства, форма разделена на такие части как: Меню, Перевозки и Люди.

Переходя к меню Пассажиры, пользователь увидит следующую форму:

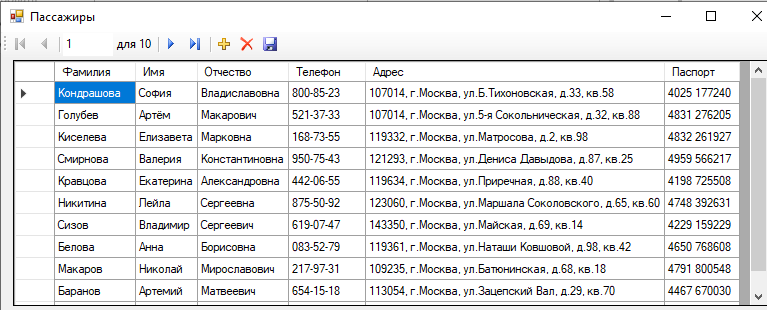


Рисунок 3.3 – Форма пассажиры

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о пассажирах.

Перейдя с главного меню на должности, пользователь увидит следующую форму.

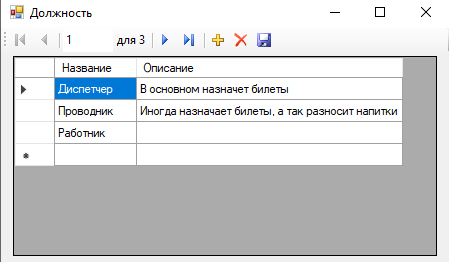


Рисунок 3.4 – Форма должности

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о должностях и их описании.

При переходе с главного меню на сотрудников, пользователь увидит следующую форму

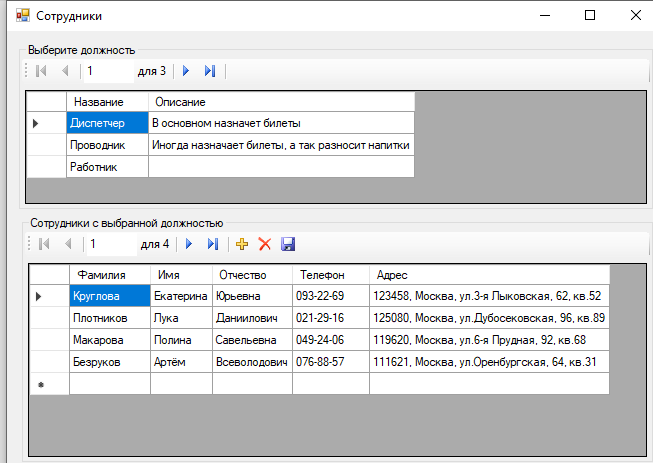


Рисунок 3.5 – Форма сотрудники

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о сотрудниках. Для просмотров сотрудников определенной должности, сперва нужно выбрать ее в верхней таблице.

При переходе с главного меню на поезда, пользователь увидит следующую

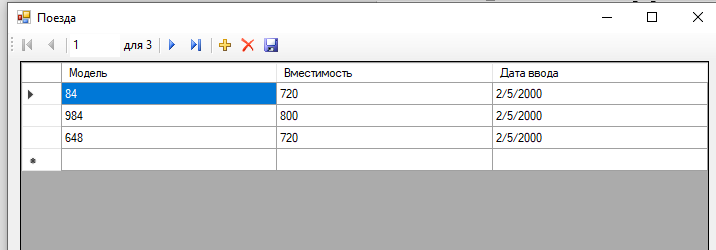


Рисунок 3.6 – Форма поезда

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о поездах.

При переходе с главного меню на области, пользователь увидит следующую форму.

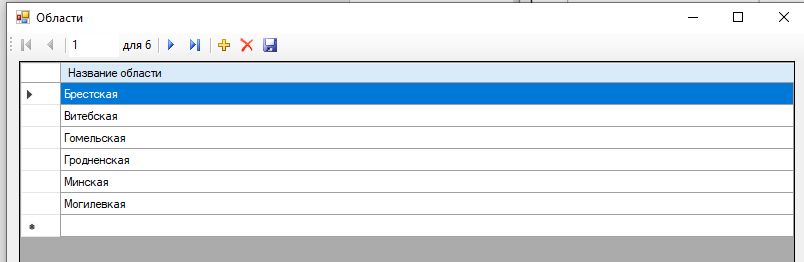


Рисунок 3.7 – Форма области

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные об областях

При переходе с главного меню на районы, пользователь увидит следующую форму.

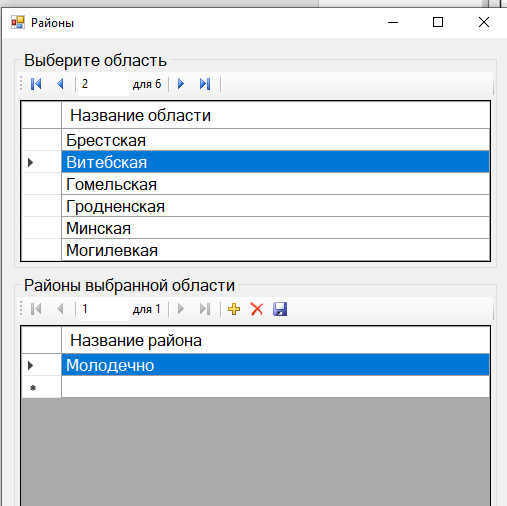


Рисунок 3. – Форма районы

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о районах

При переходе с главного меню на пункты назначения, пользователь увидит следующую форму.

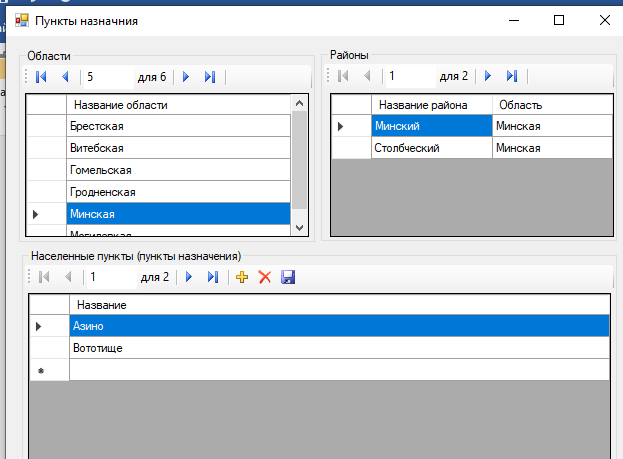
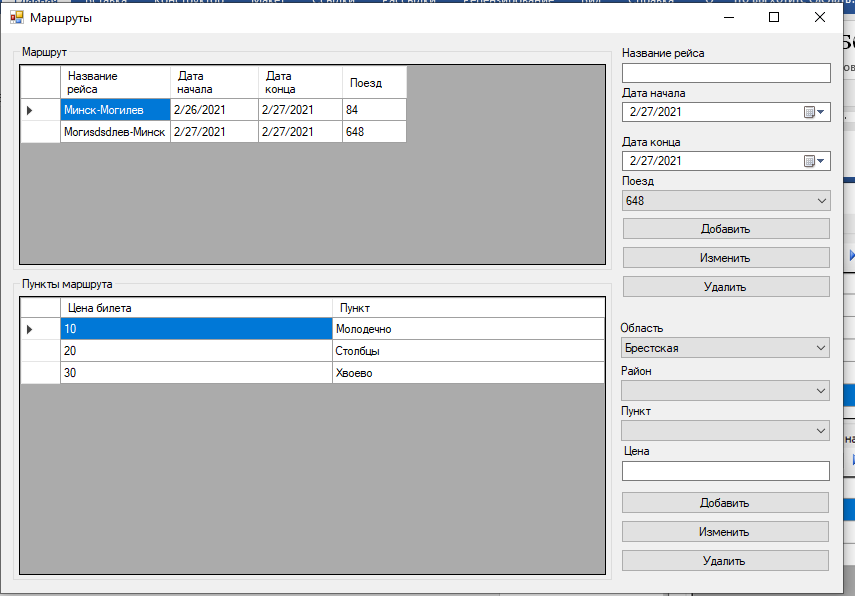


Рисунок 3.9 – Форма пункты назначения

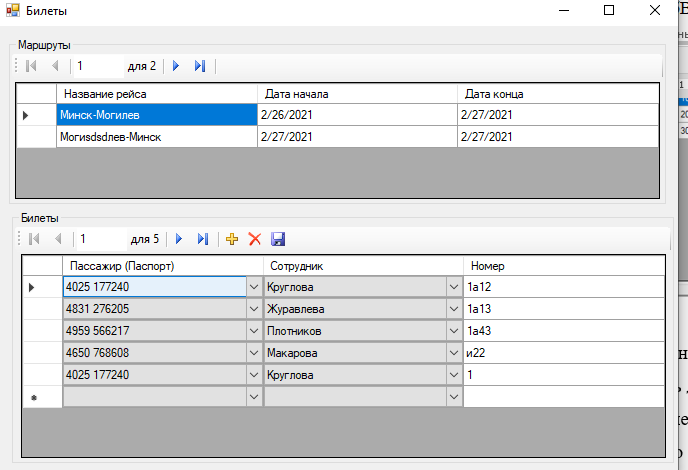
На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о пунктах назначения

При переходе с главного меню на маршруты, пользователь увидит следующую форму.

Рисунок 3.10 – Форма маршруты

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о маршрутах.

При переходе с главного меню на продажу билетов, пользователь увидит следующую форму.

Рисунок 3.11 – Форма продажи билетов

На данной форме пользователь может просматривать, изменять, удалять и добавлять данные о продаже билетов на каждый из маршрутов.

# Тестирование

Для тестирования приложения воспользуемся тест-кейсам. ТЕСТ КЕЙС (TEST CASE) – это комплекс исходных данных, условий и ожидаемых результатов, разработанный с целью проверки требуемого свойства продукта. Test cases, собранные в последовательность для достижения некоторой цели образуют test suite (набор тестов).[8]

Еще одно определение test case — это спецификация входных данных, условий выполнения, процедуры тестирования и ожидаемых результатов, которые определяют один тест, который должен быть выполнен для достижения конкретной цели тестирования программного обеспечения, например для выполнения определенного пути программы или для проверки соответствия определенному требованию.

Чтобы больше прояснить ситуацию с терминами и определениями, давайте сопоставим некоторые термины касательно этой темы:

Тест-кейс и тест = синоним

Не путаем чек-лист и тест. Отличие тест кейса от чек-листа заключается в том, что чек-лист это всего лишь идея будущего теста, а тест кейс, как мы говорили выше, набор данных и ожидаемых результатов. Часто чек лист становится хорошим началом атрибута Test description в тест-кейсе.

Не путаем тест план и тест кейс. Что касается мы уже определились, а вот тест план это фундаментальный документ, который описывает разные аспекты процесса тестирования. Подробно о тест плане мы поговорим в отдельной статье.

Тест сценарий и тест кейс, также неравны. Тестовый сценарий (test scenario) – набор тестов (тест-кейсов), собранных в последовательность для достижения некоторой цели. Хороший тестовые сценарии и тест кейсы в нем всегда следуют некоторой логике, например: типичному использованию приложения, удобству тестирования, распределению функций по модулям и т.д.

Таблица 4.1 – Тест-кейсы приложения

|  |  |  |  |
| --- | --- | --- | --- |
| Название  операции | Входной  параметр | Ожидаемый  результат | Фактический  результат |
| Запуск | Запуск файла | Открылась главная форма | Открылась главная форма |
| Переход на страницу должностей | Запустить файл. Перейти на страницу должностей | Открылась форма должностей | Открылась форма должностей |
| Переход на страницу Сотрудников | Запустить файл. Перейти на страницу Сотрудников | Открылась форма Сотрудников | Открылась форма Сотрудников |
| Добавление новой должности | Запустить файл. Перейти на страницу должностей. Ввести данные новой должности. Нажать кнопку сохранения | На экране появилось сообщение об успешном сохранении | На экране появилось сообщение об успешном сохранении |
| Добавление нового сотрудника | Запустить файл. Перейти на страницу сотрудника. Ввести данные нового сотрудника. Нажать кнопку сохранения | На экране появилось сообщение об успешном сохранении | На экране появилось сообщение об успешном сохранении |
| Изменение пункта назначения | Запустить файл. Перейти на страницу пунктов назначения. Выбрать первую область из первой таблицы. Выбрать первый район из второй таблице. Изменить данные в третьей таблице.  Нажать кнопку сохранения. | На экране появилось сообщение об успешном сохранении, после перезахода на страницу отображаются новые данные | На экране появилось сообщение об успешном сохранении, после перезахода на страницу отображаются новые данные |
| Изменение пунктов маршрута | Запустить файл. Перейти на страницу маршрутов. Выбрать маршрут из первой таблицы. Изменить пункт назначения во второй таблице. Нажать кнопку сохранения | На экране появилось сообщение об успешном сохранении, после перезахода на страницу отображаются новые данные | На экране появилось сообщение об успешном сохранении, после перезахода на страницу отображаются новые данные |
| Создание билета пользователю | Запустить файл. Перейти на страницу продажи билета. Выбрать маршрут из первой таблицы. Заполнить новую строку данными пассажира во второй таблице. Нажать кнопку сохранения | На экране появилось сообщение об успешном сохранении, после перезахода на страницу отображаются новые данные | На экране появилось сообщение об успешном сохранении, после перезахода на страницу отображаются новые данные |

Как видно из проведенных тест-кейсов, приложение работает без перебоев и выполняет все возложенные на себя функции.

# Заключение

Целью выполнения курсового проекта является разработанная база данных и приложение для автоматизации рабочего места диспетчера железнодорожного вокзала.

В ходе работы была описана предметная область, а также функциональное назначение разрабатываемой программы. При проектировании базы данных был выбран и описан инструмент разработки, проведены концептуальное и физическое проектирование базы данных.

В результате проведенных работ на базе выбранных технологий была создана база данных и приложение для автоматизации рабочего места диспетчера железнодорожного вокзала.

# Список использованных источников

1. Базы данных [Электронный ресурс]. – Режим доступа: https://www.myunivercity.ru/Информатика/Базы\_данных/47969\_1278432\_страница1.html. – Дата доступа: 27.02.2021.
2. Роль железнодорожного транспорта в обеспечении пассажирских перевозок [Электронный ресурс]. – Режим доступа: https://www.myunivercity.ru/Транспорт/Роль\_железнодорожного\_транспорта\_в\_обеспечении\_пассажирских/445910\_3306553\_страница1.html. – Дата доступа: 27.02.2021.
3. Полезные возможности среды Visual Studio [Электронный ресурс]. – Режим доступа: https://habr.com/ru/sandbox/130042/. – Дата доступа: 27.02.2021.
4. Язык программирования С#: история, специфика, место на рынке [Электронный ресурс]. – Режим доступа: https://geekbrains.ru/posts/yazyk-programmirovaniya-c-sharp-istoriya-specifika-mesto-na-rynke. – Дата доступа: 27.02.2021.
5. Система Microsoft SQL Server [Электронный ресурс]. – Режим доступа: https://intuit.ru/studies/courses/93/93/lecture/28089. – Дата доступа: 27.02.2021.
6. Инфологическое проектирование [Электронный ресурс]. – Режим доступа: https://studopedia.ru/19\_83163\_infologicheskoe-proektirovanie.html. – Дата доступа: 27.02.2021.
7. Логическое проектирование БД [Электронный ресурс]. – Режим доступа: https://rep.bntu.by/bitstream/handle/data/4940/Bazy\_dannyh.pdf?sequence=1&isAllowed=y. – Дата доступа: 27.02.2021.
8. Тест кейс (test case) [Электронный ресурс]. – Режим доступа: https://veraksoff.info/test-kejs-test-case/. – Дата доступа: 27.02.2021.