Содержание

[ВВЕДЕНИЕ 5](#_Toc28177923)

[1 АНАЛИТИЧЕСКАЯ ЧАСТЬ 6](#_Toc28177924)

[1.1 Анализ предметной области 6](#_Toc28177925)

[1.2 Постановка задач 6](#_Toc28177926)

[2 Разработка логической модели базы данных 8](#_Toc28177927)

[2.1 Построение диаграммы «сущность-связь» в нотации П.Чена 8](#_Toc28177928)

[2.1.1 Построение сложной сетевой структуры 8](#_Toc28177929)

[2.1.2 Построение простой сетевой структуры 9](#_Toc28177930)

[2.1.3 Построение древовидной структуры 10](#_Toc28177931)

[2.1.4 Построение диаграмма «сущность-связь» в нотации П.Чена на основе предыдущих диаграмм 11](#_Toc28177932)

[2.2 Построение модели, основанной на ключах 12](#_Toc28177933)

[2.3. Построение полной̆ атрибутивной̆ модели в нотации IDEF1X 12](#_Toc28177934)

[2.4 Нормализация БД 13](#_Toc28177935)

[2.4.1 Первая нормальная форма 13](#_Toc28177936)

[2.4.2 Вторая нормальная форма 14](#_Toc28177937)

[2.4.3 Третья нормальная форма 15](#_Toc28177938)

[2.4.4 Нормальная форма Бойса-Кодда 15](#_Toc28177939)

[2.5 Результат приведения БД к 3НФ и НФБК 15](#_Toc28177940)

[3 Разработка физической модели базы данных 17](#_Toc28177941)

[3.1 Выбор аппаратной и программной платформы для реализации БД 17](#_Toc28177942)

[3.2 Реализация базы данных 17](#_Toc28177943)

[3.3 Тестирование базы данных 19](#_Toc28177944)

[3.3.1 Заполнение таблиц 19](#_Toc28177945)

[3.3.2 Запросы для тестирования БД 20](#_Toc28177946)

[3.3.2 Проверка целостности 22](#_Toc28177947)

[3.4 Разграничение прав доступа 23](#_Toc28177948)

[3.5 Расчет информационных параметров базы данных 23](#_Toc28177949)

[4 Разработка клиентского приложения 25](#_Toc28177950)

[4.1 Обоснование выбора языка программирования 25](#_Toc28177951)

[4.2 Разработка интерфейса пользователя и тестирование приложения 25](#_Toc28177952)

[4.3 Алгоритм работы каждого из модулей 29](#_Toc28177953)

[ЗАКЛЮЧЕНИЕ 31](#_Toc28177954)

[СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ 33](#_Toc28177955)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc28177956)

[ПРИЛОЖЕНИЕ А Листинг запросов 35](#_Toc28177957)

[ПРИЛОЖЕНИЕ Б Листинг программы 39](#_Toc28177958)

ВВЕДЕНИЕ

В современном мире вся информация, хранимая и использующаяся в процессе работы, должна быть жестко структурирована и упорядочена. Это необходимо для того, чтобы упростить поиск, работу и сохранение данных.

В ходе выполнения курсового проекта разрабатывается физическая модель базы данных для учета и хранения заказов транспортной компании (междугородние пассажирские перевозки) в рамках технического задания.

Целями курсового проектирования является закрепление теоретических знаний, полученных в результате изучения курсов «Теория баз данных» и «Управление данными», формирование практических навыков в области информационного моделирования и проектирования баз данных, разработка физической модели базы данных.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

выбор варианта задания детализация постановки задачи и анализ предметной области;

разработка логической модели базы данных; ¾ разработка и тестирование физической модели БД;

разработка клиентского приложения.

Данная пояснительная записка состоит из: содержания, введения, а также других разделов. Раздел «Аналитическая часть», в котором описывается предметная область. Раздел «Разработка логической модели базы данных», в котором отражены логические связи между сущностями. Раздел «Разработка физической модели базы данных» описывает физические свойства базы данных (типы данных, размер полей, индексы). Раздел «Разработка клиентского приложения» в котором находятся сведения о выбранной платформе реализации, описание взаимодействие модулей, а также описание работы пользователя с ней.

1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ предметной области

Предметная область представляет собой большую ИС автовокзал, направленную на сбор, обработку информации для предоставления услуг автоперевозок. ИС автовокзала является связующим звеном между поставщиком услуг - автокомпанией и их потребителями - пассажиров. Поэтому для оптимального взаимодействия выделенных сторон, существует необходимость автоматизации информационных процессов, что ведёт к их быстродействию и качеству.

1.2 Постановка задач

Необходимо разработать базу данных "Автовокзал". Проектирумая БД предназначена для ИС диспетчеров автовокзала и обслуживающего персонала, для управления и учёта выездов всех автобусов, для автоматизации продажи билетов на междугородние и пригородные автобусные перевозки на уровне отдельного автовокзала. БД должна решать довольно узкий круг задач, связанный сопоставлением расписания и фактических выездов автобусов по различным маршрутам. Выделены следующие объекты: перевозчик, автобус, сотрудники, билет, населенные пункты, рейс, путевой лист.

Связи между объектами:

перевозчику принадлежат автобусы;

автобус указывается в путевых листах;

сотрудник указывается в билетах, которые продал, и в путевых листах, которые составил;

билет указывается в путевом листе;

насленный пункт указывается в рейсах;

рейс указывается в путевом листе.

Таким образом, были сформированы некоторые требования к разработке логической базы данных, которую необходимо будет реализовать в виде клиентского приложения.

Исходя из аналитической части, а именно из анализа предметной области и постановки задачи, получив необходимые данных (в ходе общения со специалистами, выделены основные сущности, определены первичные связи между ними), можно переходить к разработке логической модели БД.

2 Разработка логической модели базы данных

Проектирование баз данных (БД) – процесс создания схемы базы данных и определения необходимых ограничений целостности. Проектирование базы данных сводится к двум последовательным этапам:

1. Логическое проектирование – преобразование требований к данным в структуры данных.

2. Физическое проектирование – определение особенностей хранения данных, методов доступа и т.д. в зависимости от СУБД.

2.1 Построение диаграммы «сущность-связь» в нотации П.Чена

2.1.1 Построение сложной сетевой структуры

Разработка «сущность-связь» основывается на переходе от сложной сетевой структуры к древовидной и построения диаграммы.

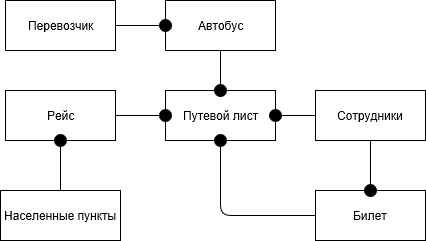


Рисунок 2.1 – Сложная сетевая структура

2.1.2 Построение простой сетевой структуры

Простая сетевая структура создается удалением связей многие ко многим. В данной сетевой структуре таких нет.

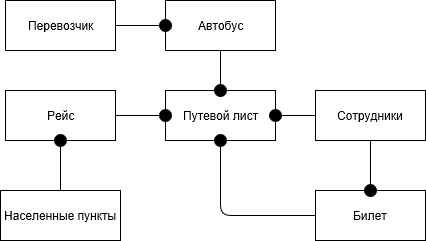


Рисунок 2.2 – Простая сетевая структура

Таким образом, сформированы 7 основных сущностей: «Перевозчик», «Автобус», «Населенные\_пункты», «Рейс», «Путевой\_лист», «Сотрудники», «Билет».

сущность «Перевозчик» содержит код перевозчика, название, адрес, телефон;

сущность «Автобус» содержит государственный номер, марку, модель, количество мест, регистрационный номер, код перевозчика;

сущность «Сотрудники» содержит ИИН, ФИО сотрудника, должность, адрес проживания, контактный телефон, стаж;

сущность «Билет» содержит № билета, дата продажи, стоимость, дата отправления, ИИН сотрудника;

сущность «Рейс» содержит № рейса, время отправления, время прибытия, расстояние, график движения, пункт отправления, код населенного пункта;

сущность «Путевой\_лист» содержит № продажи, № билета, государственный номер, ИИН сотрудника, № рейса, Код населенного пункта, Код перевозчика;

сущность «Населенные\_пункты» содержит Код населенного пункта, название населенного пункта.

Таким образом были выделенны основные сущности БД и их атрибуты.

2.1.3 Построение древовидной структуры

На основе простой сетевой структуры была построена древовидная структура, отображающая взаимосвязь ближайших объектов базы, а также, показывающая будущие внешние ключи.

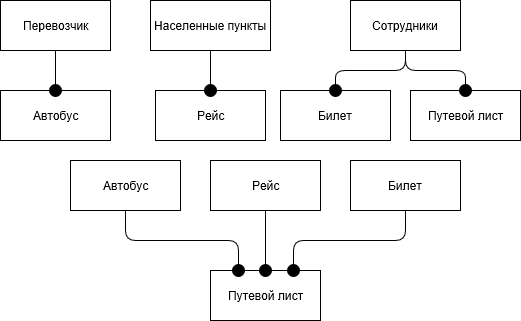


Рисунок 2.3 – Древовидная структура

2.1.4 Построение диаграмма «сущность-связь» в нотации П.Чена на основе предыдущих диаграмм

На основе прошлых таблиц была синтезирована ER диаграмма, отображающая взаимосвязь объектов, а также основные элементы взаимодействующих объектов.

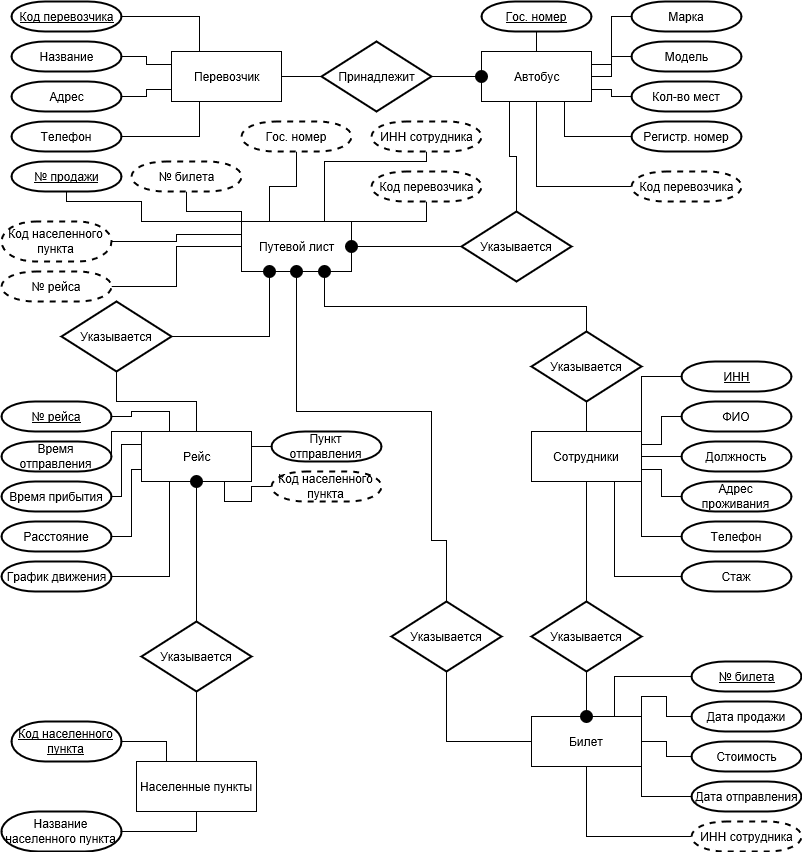


Рисунок 2.4 – Диаграмма «сущность-связь» в нотации П.Чена

2.2 Построение модели, основанной на ключах

Для построения модели основанной на ключах, сущности диаграммы «сущность-связь» были идентифицированы посредствам выбора ключевых атрибутов, которые однозначно определяют каждую из них. Результат - структура, изображенная на рисунке 2.5.

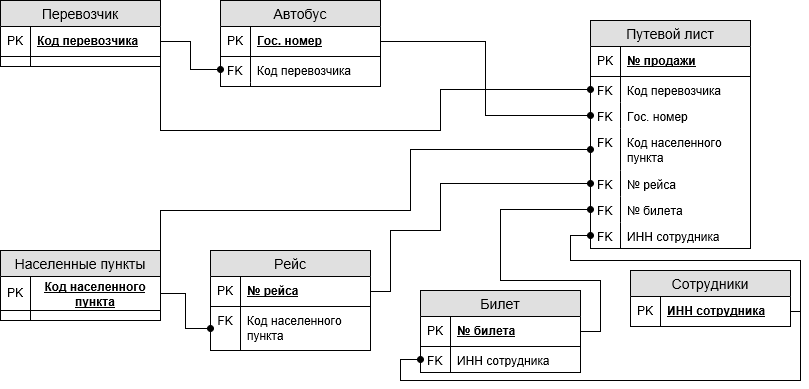


Рисунок 2.5 – Модель, основанная на ключах

Модель, основанная на ключах, строится на основе ERD, но отображает больше деталей: ключевые поля, внешние ключи, таким образом, построение модели, основанной на ключах, позволяет оценить взаимосвязи между сущностями с помощью определение первичных и внешних ключей.

2.3. Построение полной̆ атрибутивной̆ модели в нотации IDEF1X

Полная атрибутивная модель – более подробное представление данных, чем модель, основанная на ключах. Она включает описание всех сущностей, внешних и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

На рисунке 2.6 изображена полная атрибутивная модель БД «Автовокзал»

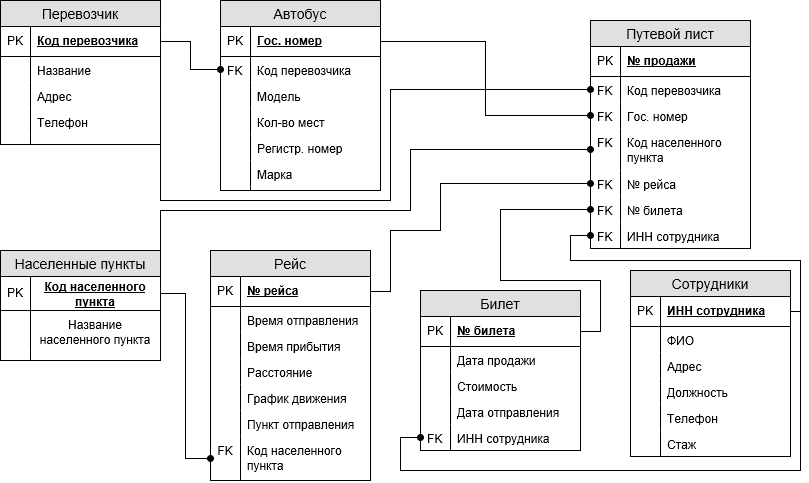


Рисунок 2.6 – Полная атрибутивная модель в нотации IDEFIX

2.4 Нормализация БД

На основе полной̆ атрибутивной̆ модели, можно провести нормализацию отношений. Номализация будет проводится до третьей нормальной , а так же нормальной фомы Бойса-Кодда.

2.4.1 Первая нормальная форма

Чтобы отношение соответствовало первой нормальной форме необходимо привести это отношение в соответствие трем условиям:

1) определить первичные ключи;

2) устранить повторяющиеся группы (одинаковые кортежи);

3) привести поля отношения к атомарности

Исследуя БД можно заметить, что отношение «Перевозчик» не соответствует 1НФ, т. к. поле «Адрес» является составным, оно состоит из «Город», «Улица», «№ дома». Также в отношении «Сотрудники» атрибуты «ФИО» и «Адрес» являются делимыми. Атрибут «Адрес» можно разбить на: «Улица», «Город», «Дом», «Квартира». А «ФИО» - «Фамилия», «Имя», «Отчество». И в отношении «Населенные пункты» «Название населенного пункта» - делимый атрибут, разбиваемый на 2 атрибута «Название страны» и «Название города». Необходима декомпозиция.

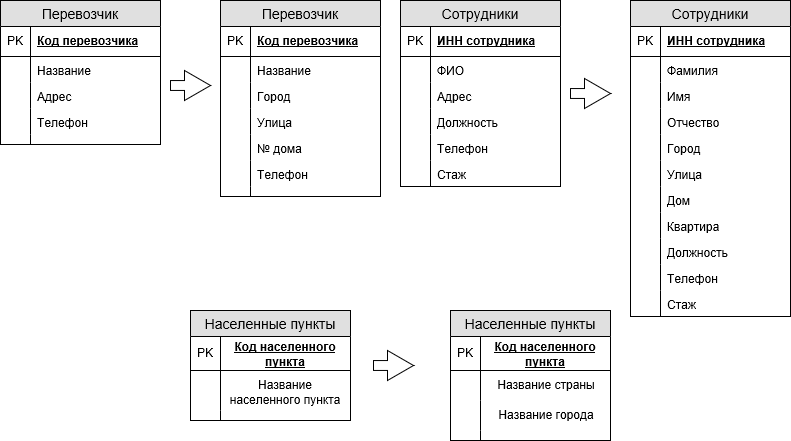


Рисунок 2.7 – Пример декомпозиции для приведения БД к 1НФ

2.4.2 Вторая нормальная форма

Отношение находится во второй нормальной форме, если оно находится в 1НФ, и каждый не ключевой атрибут характеризуется полной функциональной зависимостью от первичного ключа (т.е. все не ключевые атрибуты зависят только от ключа целиком, а не от какой-то его части).

Если посмотреть на БД, то будет видно, что все отношения не имеют составных ключей, следовательно, БД находится в 2НФ.

2.4.3 Третья нормальная форма

Отношение находится в третьей нормальной форме, если оно находится во 2НФ и никакой не ключевой атрибут функционально не зависит от другого не ключевого атрибута, т.е. нет транзитивных зависимостей.

Все неключевые поля таблиц зависят от первичного ключа и не имеют зависимости друг от друга, следовательно, модель приведена к 3НФ

2.4.4 Нормальная форма Бойса-Кодда

Нормальная форма Бойса-Кодда (Далее НФБК) считается уточнением 3НФ. Она учитывает все потенциальные ключи, которые входят в отношения. Если отношение имеет единственный потенциальный ключ, то 3НФ и НФБК – эквивалентны. Считается, что отношение, находящееся в НФБК, если каждый его детерминант является потенциальным ключом. Чтобы убедиться, что отношение находится в НФБК необходимо отыскать все его детерминанты и убедиться, что они являются потенциальными ключами

БД находится в НФБК т. к. в каждом отношении все поля можно объединить в составной первичный ключ, т. е. все поля являются потенциальными ключами.

2.5 Результат приведения БД к 3НФ и НФБК

На рисунке 2.8 изображена полная атрибутивная модель БД в нотации IDEFIX в 3НФ и НФБК.

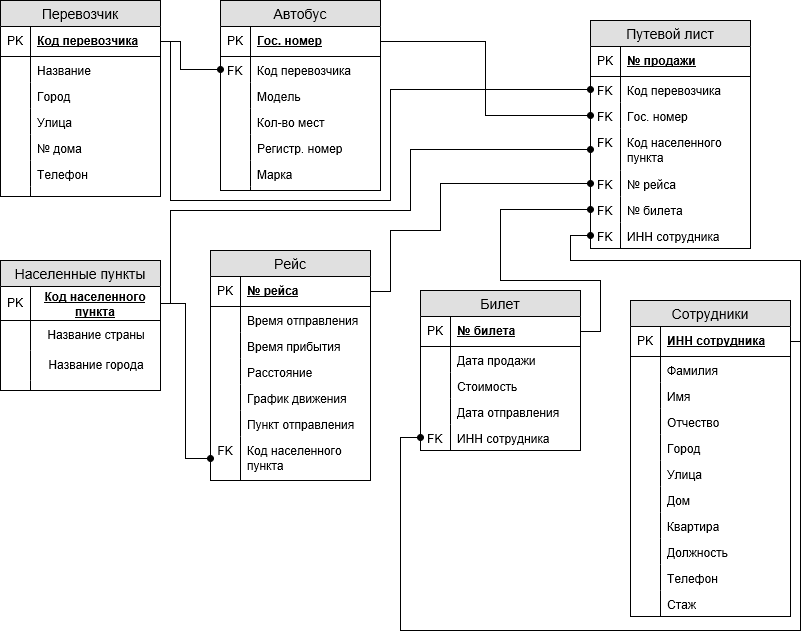


Рисунок 2.8 - Полная атрибутивная модель в нотации IDEFIX в 3НФ и НФБК

В разделе 2 были разработаны сложная сетевая сруктура, на ее основе была спроектирована простая сетевая структура, далее древовидная структура, показывающая внешние ключи, связывающие сущности и, наконец, диграмма «сущность связь», показывающая будужие атрибуты сущностей, и их взамодействия с другими сущностями. Также была подробно рассмотрена нормализация отношений от 1НФ к нормальной форме Бойса-Кодда включительно. Построение логической модели аргументировано приведением к нормальной форме Бойса-Кодда для улучшения структура базы данных перед реализацией.

3 Разработка физической модели базы данных

3.1 Выбор аппаратной и программной платформы для реализации БД

В качестве СУБД служит MySQL, которая была выбрана из-за того, что к этой базе относительно просто подключаться, есть встроенный графический интерфейс, который позволяет ускорить работу с вводом и выводом данных, бесплатная лицензия для некоммерческих проектов. Данная СУБД была выбрана из-за наличия следующих достоинств:

Весьма высокое быстродействие;

Высокий уровень безопасности;

Бесплатное распространение программы для некоммерческих целей;

Открытость кода. Благодаря этому вы сможете сами добавлять в пакет нужные функции, расширяя его функциональность так, как вам требуется;

Переносимость. Существуют версии программы для большинства распространенных компьютерных платформ. Это говорит о том, что не навязывается определенная операционная система;

Бд будет реализована в Open Server Panel - портативная серверная платформа и программная среда, созданная специально для веб-разработчиков

3.2 Реализация базы данных

Полный листинг SQL скриптов представлен в разделе Приложение B. Физическая модель была разработана в приложении MySQL Workbench.

На рисунке 3.1 изображена физическая модель БД.

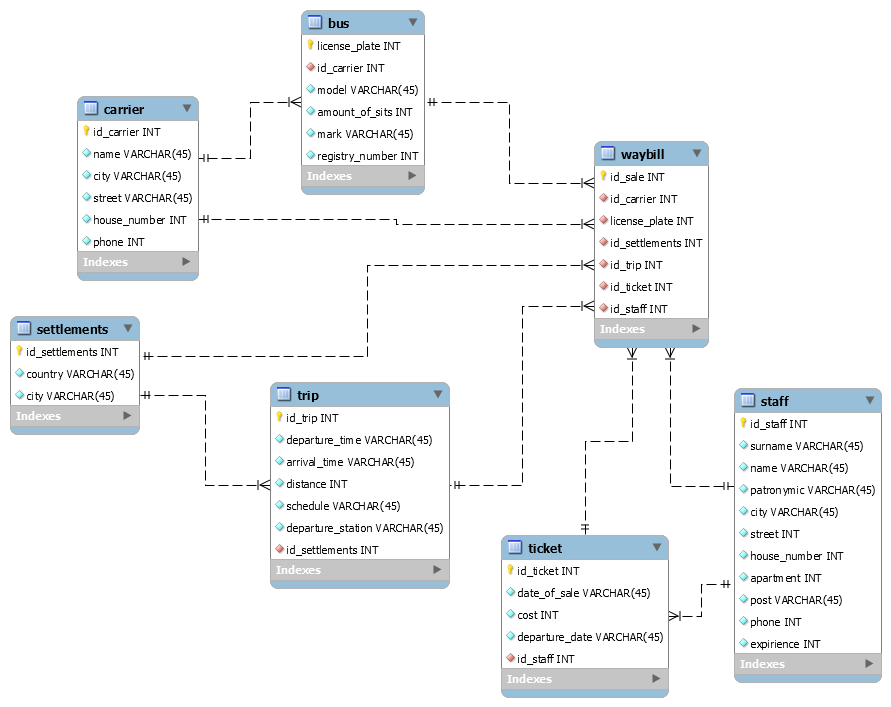


Рисунок 3.1 – Cкриншот представления разработанной базы данных

3.3 Тестирование базы данных

3.3.1 Заполнение таблиц

На рисунке 3.2 изображен результат заполнение таблицы «Перевозчик».

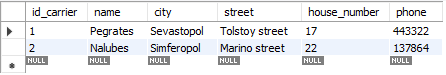


Рисунок 3.2 – Заполнение таблицы «Перевозчик»

На рисунке 3.3 изображен результат заполнение таблицы «Автобус».

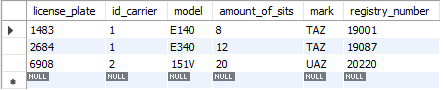


Рисунок 3.3 – Заполнение таблицы «Автобус»

На рисунке 3.4 изображен результат заполнение таблицы «Сотрудники»

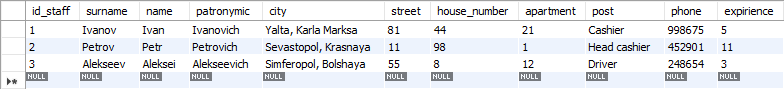


Рисунок 3.4 – Заполнение таблицы «Сотрудники»

На рисунке 3.5 изображен результат заполнение таблицы «Рейс»

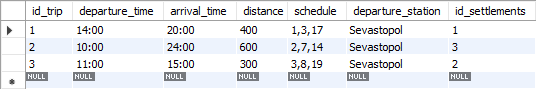


Рисунок 3.5 – Заполнение таблицы «Рейс»

На рисунке 3.6 изображен результат заполнение таблицы «Билет»

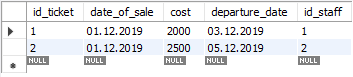


Рисунок 3.6 – Заполнение таблицы «Билет»

На рисунке 3.7 изображен результат заполнение таблицы «Населенные пункты».

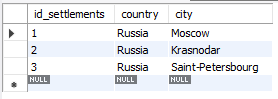


Рисунок 3.7 – Заполнение таблицы «Населенные пункты»

На рисунке 3.8 изображен результат заполнение таблицы «Путевой лист».

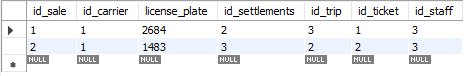


Рисунок 3.8 – Заполнение таблицы «Путевой лист»

3.3.2 Запросы для тестирования БД

Выберем перевозчиков из города «Sevastopol» (рисунок 3.9).

SELECT id\_carrier, name

FROM bus\_station.carrier

WHERE city = "Sevastopol";



Рисунок 3.9 – Результат запроса на выборку клиентов перевозчиков из города «Sevastopol»

Выберем количество автобусов у каждого перевозчика (рисунок 3.10).

SELECT bus\_station.carrier.name, COUNT(bus\_station.bus.license\_plate)

FROM bus\_station.carrier, bus\_station.bus

WHERE bus\_station.carrier.id\_carrier = bus\_station.bus.id\_carrier

GROUP BY bus\_station.carrier.name;



Рисунок 3.10 – Результат запроса на выборку количества автобусов у каждого перевозчика

Выберем населенные пункты, которых нет ни в 1 путевом листе c (рисунок 3.11).

SELECT \* FROM bus\_station.settlements

WHERE bus\_station.settlements.city NOT IN (SELECT city FROM bus\_station.settlements, bus\_station.waybill WHERE bus\_station.settlements.id\_settlements = bus\_station.waybill.id\_settlements);

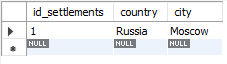


Рисунок 3.11 – Результат запроса на выборку населенных пунктов

Выберем сколько билетов куплено на автобус с самым наименьшим количеством мест (рисунок 3.12).

SELECT COUNT(bus\_station.waybill.id\_ticket)

FROM bus\_station.waybill

WHERE bus\_station.waybill.license\_plate IN (SELECT license\_plate FROM bus\_station.bus WHERE bus\_station.bus.amount\_of\_sits IN (SELECT MIN(bus\_station.bus.amount\_of\_sits) FROM bus\_station.bus));

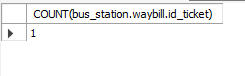


Рисунок 3.12 – Результат запроса на выборку количества билетов

Выберем рейсы водителей со стажем меньше 5 лет (рисунок 3.13).

SELECT bus\_station.staff.surname, bus\_station.staff.name, bus\_station.trip.id\_trip

FROM bus\_station.trip, bus\_station.waybill, bus\_station.staff

WHERE bus\_station.trip.id\_trip = bus\_station.waybill.id\_trip

AND bus\_station.waybill.id\_staff = bus\_station.staff.id\_staff

AND bus\_station.staff.post = "Driver"

AND bus\_station.staff.expirience < 5;

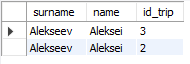


Рисунок 3.13 – Результат запроса на выборку рейсов водителей со стажем меньше 5 лет

Выберем путевые листы, где населенный пункт начинается на букву К (рисунок 3.14).

SELECT bus\_station.waybill.id\_sale

FROM bus\_station.waybill, bus\_station.settlements

WHERE bus\_station.waybill.id\_settlements = bus\_station.settlements.id\_settlements

AND bus\_station.settlements.city LIKE "K%";



Рисунок 3.14 – Результат запроса на выборку путевых листов, где населенный пункт начинается на букву К

3.3.2 Проверка целостности

Проверим целостность БД на основе отношений «Населенные пункты», «Рейс», «Путевой лист».

Удалим одну запись в таблице «Населенные пункты», в результате чего в таблицах «Рейс» и «Путевой лсит» удалятся все записи, кортежи которых ссылаются на удаленного владельца.

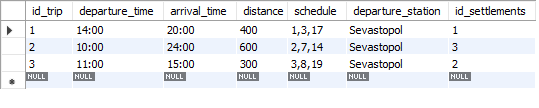


Рисунок 3.15 – Таблица «Рейс» до удаления

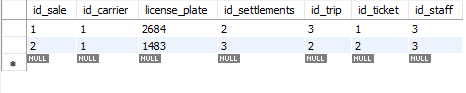


Рисунок 3.16 – Таблица «Путевой лист» до удаления

DELETE FROM bus\_station.settlements

WHERE city = "Krasnodar";

На рисунках 3.17 и 3.18 видно, что поля были удалены, так как внешние ключи, таблиц «Рейс» и «Путевой лист», имеют действие при удалении CASCADE.

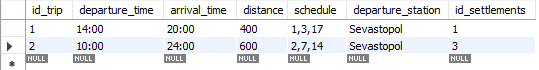


Рисунок 3.17 – Таблица «Рейс» после удаления владельца



Рисунок 3.18 – Таблица «Путевой лист» после удаления владельца

3.4 Разграничение прав доступа

Данная БД создана для работников автовокзала,следовательно в ее права входит: создание, удаление, редактирование полей всех таблиц и запрет на удаление таблиц и базы данных.

3.5 Расчет информационных параметров базы данных

Для переменных длин полей возьмем максимальную длину поля, следовательно, размер типов данных для хранения в памяти будет составлять:

1. integer – 2 байта

2. Varchar(x) – x байт

Были расчитаны длины логических записей:

Таблица «bus»:

Таблица «carrier»:

Таблица «settlements»:

Таблица «staff»:

Таблица «ticket»:

Таблица «trip»:

Таблица «waybill»:

Предположим, что в таблицах «bus» 10 записей (L1 = 98), «carrier» 10 записей (L2 = 141), «settlements » 10 постояльцев (L3 = 92), «staff » 10 записей (L4 = 237), «ticket » 10 записей (L5 = 96), «trip» 10 записей (L6 = 186), «waybill» 10 записей (L7 = 14).

Рассчитаем объем памяти, необходимой для размещения информационного фонда:

Приращение информационного фонда для интенсивности добавления записей в 0.01 [1/байт2]: ΔI = (1/байт)

Время резервного копирования всей структуры занимает:

3.2 (секунды)

В разделе 3, была выбрана программная платформа, на которой будет риализована БД. После чего следовала реализация физической модели БД, а далее ее заполнение. Были проведены тесты БД, проверка её целостноти, ограничений и работы простых запросов на вставку, обновление и удаление записей. Так же в БД реализовано разграничение прав доступа на менеджера и администратора. Окончательным этапом реализации физической модели БД был расчет информационных параметров. Они были рассчитаны с допущениями, что БД включает полную наполненность всех полей атрибутов.

4 Разработка клиентского приложения

4.1 Обоснование выбора языка программирования

Программная структура организована с помощью форм – именно в них происходит вся обработка информации.

В качестве языка программирования был выбран Java: язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования. Его плюсами является объектно-ориентированное программирование, высокоуровневость программирования с простым синтаксисом, кроссплатформенность.

Для связи с БД был выбран JDBC API - платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), реализованный в виде пакета java.sql, входящего в состав [Java SE](https://ru.wikipedia.org/wiki/Java_SE).

4.2 Разработка интерфейса пользователя и тестирование приложения

Программная структура организована с помощью форм – именно в них происходит вся обработка информации. Полный листинг представлен в разделе «Приложение Б»

На рисунке 4.1 изображено гланое окно «Автовокзала». На нем можно выбрать куда перейти дальше - на страницу с информацией или на страницу продажи билетов.

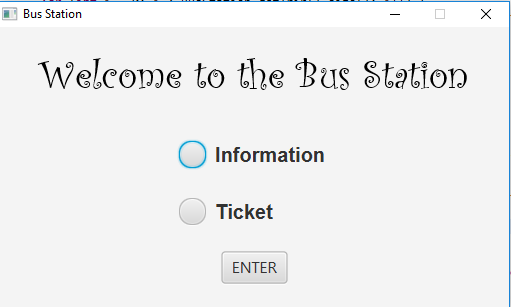


Рисунок 4.1 – Главное окно «Автовокзала»

На рисунке 4.2 изображено окно информации. В нем присутствуют кнпоки просмотра рейсов, автобусов, персонала, а также кнопка выхода.

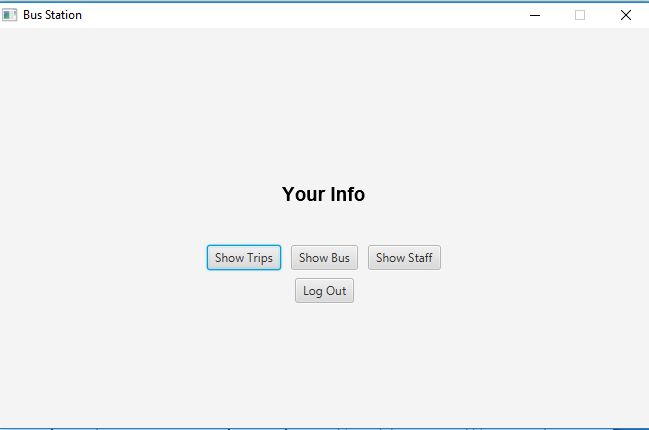


Рисунок 4.2 – Окно «Информация»

На рисунке 4.3 изображена таблица, которая выводится при нажатии на кнопку «Показать рейсы».

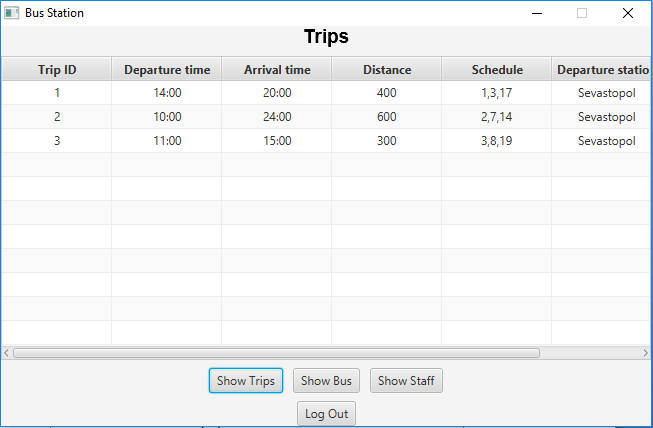


Рисунок 4.3 – Окно «Информация» после нажатия на кнопку «Показать рейсы»

На рисунке 4.4 изображена таблица, которая выводится при нажатии на кнопку «Показать автобусы».

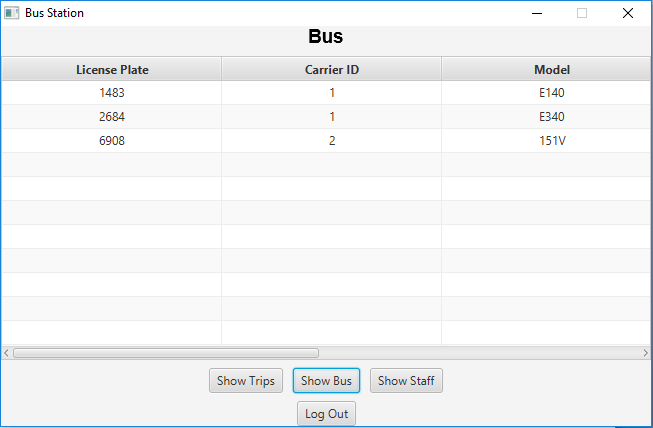


Рисунок 4.4 – Окно «Информация» после нажатия на кнопку «Показать автобусы»

На рисунке 4.5 изображена таблица, которая выводится при нажатии на кнопку «Показать персонал».

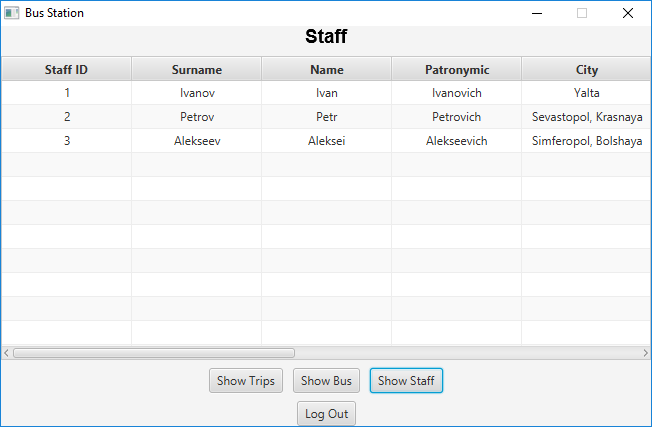


Рисунок 4.5 – Окно «Информация» после нажатия на кнопку «Показать персонал»

На рисунке 4.6 изображено окно продажи билетов.

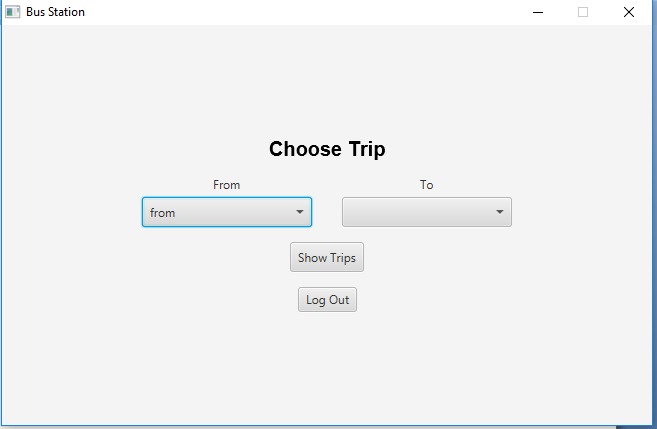


Рисунок 4.6 – Окно продажи

На рисунке 4.7 изображено окно выбора рейса и подтверждения продажи билета.

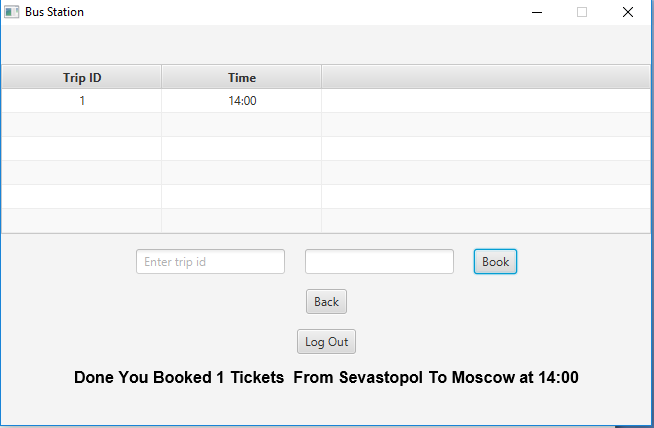


Рисунок 4.7 – Окно продажи билетов

4.3 Алгоритм работы каждого из модулей

Самым главным модулем является DbConnection.java, так как он связывается с БД и предоставляет возможность остальным частям программы производить манипуляции с БД.

Загрузка данных из БД осуществляется функцией readDataFromDB().

Вот ее реализация:

**public** **static** **void** readDataFromDB() **throws** SQLException, ClassNotFoundException {

*con* = DbConnection.*connectToDB*();

Statement s = *con*.createStatement();

*rs* = s.executeQuery("Select \* from bus");

// read bus data

**while** (*rs*.next())

*bus*.add(**new** Bus(*rs*.getInt(1), *rs*.getInt(2), *rs*.getString(3), *rs*.getInt(4), *rs*.getString(5), *rs*.getInt(6)));

// read staff data

*rs* = s.executeQuery("Select \* from staff");

**while** (*rs*.next())

*staff*.add(**new** Staff(*rs*.getInt(1), *rs*.getString(2), *rs*.getString(3), *rs*.getString(4), *rs*.getString(5),

*rs*.getInt(6), *rs*.getInt(7),*rs*.getInt(8),*rs*.getString(9),*rs*.getInt(10),*rs*.getInt(11)));

// read trip data

*rs* = s.executeQuery("Select \* from trip");

**while** (*rs*.next())

*trip*.add(**new** Trips(*rs*.getInt(1), *rs*.getString(2), *rs*.getString(3), *rs*.getInt(4), *rs*.getString(5),

*rs*.getString(6), *rs*.getInt(7)));

// read settlement data

*rs* = s.executeQuery("Select \* from settlements");

**while** (*rs*.next())

*settlement*.add(**new** Settlements(*rs*.getInt(1), *rs*.getString(2), *rs*.getString(3)));

}

Для удобства работы с базой данных было реализованно приложения, осуществляющий взаимодействие с ней. Самым первым этапом в разработке приложения былы выбор основного языка программирования, а также программной среды. Затем была реализована клиентская часть и структура будущего приложения. Следующим этапом стало написание приложения для взаимодействия с базой данных. В разделе 4 было описана реализация клиентского приложения на платформе Java, с помощью её библиотеки JDBC, алгоритм работы приложения, а также были проведены тесты, проверяющие правильность его работы, а также описание основных модулей.

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте была спроектирована реляционная база данных на тему «База данных транспортной компании (междугородние пассажирские перевозки)».

Для выбранной предметной области был проведен анализ, построена логическая модель и проведена нормализация, которая позволяет решить проблемы рационального выбора вариантов схем отношений из возможного множества альтернативных решений. В полученной в результате нормализации логической схеме содержатся следующие таблицы: перевозчик, автобус, населенные пункты, сотрудники, билет, путевой лист, рейс.

Была разработана база данных, эмулирующая предметную область «Автовокзал» предназначенная для перевозки пассажиров между городами. К базе данных в качестве интерфейса было создано приложение для кассира (или другого сотрудника).

В процессе выполнения были достигнуты следующие задачи:

- описана предметная область;

- выбраны информационные технологии, подходящих для разработки информационной системы;

- разработаны логическая и физическая модель базы данных;

- сделано физическое проектирование базы, данных в выбранной СУБД;

- разработано программное обеспечение, реализующее и формы для базы данных;

- проведено тестирование базы данных и приложения;

- рассчитаны информационные параметры БД.

Выполнение работы включало в себя множество различных этапов и шагов. Сначала была получена задача на курсовое проектирование. Далее было проведено подробное исследование предметной области. На основе полученных данных была спроектирована логическая модель данных, содержащая в себе перечень сущностей предметной области, перечень свойств этих сущностей и связей между ними. Для оптимизации выполнения запросов была проведена нормализация структуры логической модели. Затем на основе проекта была реализована физическая модель данных в СУБД MySQL Workbench. Реализация включала в себя подробное тестирование ограничений целостности базы данных и различные запросы реляционной алгебры.

Для демонстрации работы существующей базы данных был реализован программный продукт, осуществляющий взаимодействие с ней. Создание этого продукта также включало в себя множество этапов. Сначала были выбраны основные языки программирования и разметки, а также программные среды. Затем была реализована клиентская часть и структура будущего приложения с учетом разграничения прав доступа. Следующим этапом стало написание приложения для взаимодействия с базой данных.

Для работы с базой был выбран язык Java, а для подключения к БД библиотека JDBC.

По результатам тестирования разработанного продукта, можно сделать вывод, что задача, поставленная на курсовое проектирование, была выполнена в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

ИС - информационная система.

БД - база данных.

1НФ – первая нормальная форма.

2НФ – вторая нормальная форма.

3НФ – третья нормальная форма.

НФБК – норбальная форма Бойса-Кодда.

БД – база банных.

ERD — Entity-Relationship Diagrams (диаграмма «сущность-связь»)

JDBC API — (Java DataBase Connectivity) соединение с базами данных на Java

[Java SE](https://ru.wikipedia.org/wiki/Java_SE) —  Java Platform Standard Edition (стандартная версия платформы Java)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Java. Промышленное программирование: практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск: УниверсалПресс, 2007. – 704 с

SQL: Пер. с англ. – М.: ДМК Пресс. – 456 с.: ил. (Серия «Quick Start»)

Swing: Эффектные пользовательские интерфейсы издание 2-ое: Издательство “Лори”, 2011. – 591 c.

Дейт, К. Дж. Введение в системы баз данных; К.: Диалектика; Издание 6-е – Москва, 2010. – 784 с.

Кузнецов С.Д. Основы баз данных; Бином. Лаборатория знаний, Интернет-университет информационных технологий – Москва, 2010. – 488 с.

ПРИЛОЖЕНИЕ А  
Листинг запросов

-- -----------------------------------------------------

-- Schema Bus\_station

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `Bus\_station` DEFAULT CHARACTER SET utf8 ;

USE `Bus\_station` ;

-- -----------------------------------------------------

-- Table `Bus\_station`.`carrier`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`carrier` (

`id\_carrier` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`city` VARCHAR(45) NOT NULL,

`street` VARCHAR(45) NOT NULL,

`house\_number` INT NOT NULL,

`phone` INT NOT NULL,

PRIMARY KEY (`id\_carrier`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`bus`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`bus` (

`license\_plate` INT NOT NULL,

`id\_carrier` INT NOT NULL,

`model` VARCHAR(45) NOT NULL,

`amount\_of\_sits` INT NOT NULL,

`mark` VARCHAR(45) NOT NULL,

`registry\_number` INT NOT NULL,

PRIMARY KEY (`license\_plate`),

INDEX `carrier\_idx` (`id\_carrier` ASC) VISIBLE,

CONSTRAINT `carrier`

FOREIGN KEY (`id\_carrier`)

REFERENCES `Bus\_station`.`carrier` (`id\_carrier`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`settlements`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`settlements` (

`id\_settlements` INT NOT NULL AUTO\_INCREMENT,

`country` VARCHAR(45) NOT NULL,

`city` VARCHAR(45) NOT NULL,

PRIMARY KEY (`id\_settlements`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`trip`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`trip` (

`id\_trip` INT NOT NULL AUTO\_INCREMENT,

`departure\_time` VARCHAR(45) NOT NULL,

`arrival\_time` VARCHAR(45) NOT NULL,

`distance` INT NOT NULL,

`schedule` VARCHAR(45) NOT NULL,

`departure\_station` VARCHAR(45) NOT NULL,

`id\_settlements` INT NOT NULL,

PRIMARY KEY (`id\_trip`),

INDEX `settlements\_idx` (`id\_settlements` ASC) VISIBLE,

CONSTRAINT `settlements`

FOREIGN KEY (`id\_settlements`)

REFERENCES `Bus\_station`.`settlements` (`id\_settlements`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`staff`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`staff` (

`id\_staff` INT NOT NULL,

`surname` VARCHAR(45) NOT NULL,

`name` VARCHAR(45) NOT NULL,

`patronymic` VARCHAR(45) NOT NULL,

`city` VARCHAR(45) NOT NULL,

`street` INT NOT NULL,

`house\_number` INT NOT NULL,

`apartment` INT NOT NULL,

`post` VARCHAR(45) NOT NULL,

`phone` INT NOT NULL,

`expirience` INT NOT NULL,

PRIMARY KEY (`id\_staff`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`ticket`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`ticket` (

`id\_ticket` INT NOT NULL AUTO\_INCREMENT,

`date\_of\_sale` VARCHAR(45) NOT NULL,

`cost` INT NOT NULL,

`departure\_date` VARCHAR(45) NOT NULL,

`id\_staff` INT NOT NULL,

PRIMARY KEY (`id\_ticket`),

INDEX `staff\_idx` (`id\_staff` ASC) VISIBLE,

CONSTRAINT `staff`

FOREIGN KEY (`id\_staff`)

REFERENCES `Bus\_station`.`staff` (`id\_staff`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`waybill`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`waybill` (

`id\_sale` INT NOT NULL AUTO\_INCREMENT,

`id\_carrier` INT NOT NULL,

`license\_plate` INT NOT NULL,

`id\_settlements` INT NOT NULL,

`id\_trip` INT NOT NULL,

`id\_ticket` INT NOT NULL,

`id\_staff` INT NOT NULL,

PRIMARY KEY (`id\_sale`),

INDEX `carrier\_idx` (`id\_carrier` ASC) VISIBLE,

INDEX `plate\_idx` (`license\_plate` ASC) VISIBLE,

INDEX `settlements\_idx` (`id\_settlements` ASC) VISIBLE,

INDEX `trip\_idx` (`id\_trip` ASC) VISIBLE,

INDEX `ticket\_idx` (`id\_ticket` ASC) VISIBLE,

INDEX `staff\_idx` (`id\_staff` ASC) VISIBLE,

CONSTRAINT `wb\_carrier`

FOREIGN KEY (`id\_carrier`)

REFERENCES `Bus\_station`.`carrier` (`id\_carrier`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_plate`

FOREIGN KEY (`license\_plate`)

REFERENCES `Bus\_station`.`bus` (`license\_plate`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_settlements`

FOREIGN KEY (`id\_settlements`)

REFERENCES `Bus\_station`.`settlements` (`id\_settlements`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_trip`

FOREIGN KEY (`id\_trip`)

REFERENCES `Bus\_station`.`trip` (`id\_trip`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_ticket`

FOREIGN KEY (`id\_ticket`)

REFERENCES `Bus\_station`.`ticket` (`id\_ticket`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_staff`

FOREIGN KEY (`id\_staff`)

REFERENCES `Bus\_station`.`staff` (`id\_staff`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

ПРИЛОЖЕНИЕ Б  
Листинг программы

Actions.java

**package** design;

**import** javafx.collections.FXCollections;

**import** javafx.collections.ObservableList;

**import** javafx.scene.control.TableColumn;

**import** javafx.scene.control.TableView;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** busStation.BusStation;

**import** busStation.DbConnection;

**import** busStation.Settlements;

**import** busStation.Staff;

**import** busStation.Trips;

**import** busStation.Bus;

**import** javafx.application.Application;

**import** javafx.scene.text.Font;

**import** javafx.scene.text.FontPosture;

**import** javafx.scene.text.FontWeight;

**import** javafx.stage.Stage;

**public** **class** Actions **extends** GUIDesign {

**public** **static** **void** main(String[] args) {

**try** {

Application.*launch*(args);

} **catch** (RuntimeException ex) {

System.***out***.println("Please Run the MySQL Script attached in project folder first");

}

}

**public** **void** start(Stage finalStage) **throws** ClassNotFoundException, SQLException {

BusStation.*readDataFromDB*();

// calling parent method body

**super**.start(stage);

welcomeWindowActions();

}

**int** check = 0;

**public** **void** welcomeWindowActions() {

welcomeWindow();

enter.setOnAction(e -> {

**if** (chooseEmployee.isSelected()) {

informationPageActions();

check = 1;

} **else** **if** (chooseCustomer.isSelected()) {

BookTicketActions();

check = 2;

}

});

}

// employee's account options,either driver or manager

@SuppressWarnings({ "unchecked", "rawtypes" })

**public** **void** informationPageActions() {

informationPage();

// logout of employee account

logout.setOnAction(e -> {

stage.setScene(scene1);

welcomeWindowActions();

});

// Manager option of showing all trips in station

showTripsManager.setOnAction(e -> {

TableView table = **new** TableView();

ObservableList<Trips> data = FXCollections.*observableArrayList*(BusStation.*getTrip*());

TableColumn<Trips, String> idTripCol = **new** TableColumn<Trips, String>("Trip ID");

tablesDesign(idTripCol, "idTrip");

TableColumn<Trips, String> departureTimeCol = **new** TableColumn<Trips, String>("Departure time");

tablesDesign(departureTimeCol, "departureTime");

TableColumn<Trips, String> arrivalTimeCol = **new** TableColumn<Trips, String>("Arrival time");

tablesDesign(arrivalTimeCol, "arrivalTime");

TableColumn<Trips, String> distanceCol = **new** TableColumn<Trips, String>("Distance");

tablesDesign(distanceCol, "distance");

TableColumn<Trips, String> scheduleCol = **new** TableColumn<Trips, String>("Schedule");

tablesDesign(scheduleCol, "schedule");

TableColumn<Trips, String> departureStationCol = **new** TableColumn<Trips, String>("Departure station");

tablesDesign(departureStationCol, "departureStation");

TableColumn<Trips, String> idSettlementsCol = **new** TableColumn<Trips, String>("Settlements ID");

tablesDesign(idSettlementsCol, "idSettlements");

table.setItems(data);

table.getColumns().addAll(idTripCol, departureTimeCol, arrivalTimeCol, distanceCol, scheduleCol,

departureStationCol, idSettlementsCol);

completeInfo.setFont(Font.*font*("Arial", FontWeight.***NORMAL***, FontPosture.***REGULAR***, 15));

emplyeeInfo.setText("Trips");

employeeBox.getChildren().clear();

employeeBox.getChildren().addAll(emplyeeInfo, table, managerOption, logout);

});

// Manager option of showing all Vehicles in Station

showVehiclesManager.setOnAction(e -> {

TableView table = **new** TableView();

ObservableList<Bus> data = FXCollections.*observableArrayList*(BusStation.*getBus*());

TableColumn<Bus, String> licensePlateCol = **new** TableColumn<Bus, String>("License Plate");

tablesDesign(licensePlateCol, "licensePlate");

licensePlateCol.setMinWidth(220);

TableColumn<Bus, String> IdCarrierCol = **new** TableColumn<Bus, String>("Carrier ID");

tablesDesign(IdCarrierCol, "idCarrier");

IdCarrierCol.setMinWidth(220);

TableColumn<Bus, String> modelCol = **new** TableColumn<Bus, String>("Model");

tablesDesign(modelCol, "model");

modelCol.setMinWidth(220);

TableColumn<Bus, String> amountOfSitsCol = **new** TableColumn<Bus, String>("Amount of sits");

tablesDesign(amountOfSitsCol, "amountOfSits");

amountOfSitsCol.setMinWidth(220);

TableColumn<Bus, String> markCol = **new** TableColumn<Bus, String>("Mark");

tablesDesign(markCol, "mark");

markCol.setMinWidth(220);

TableColumn<Bus, String> registryNumberCol = **new** TableColumn<Bus, String>("Registry number");

tablesDesign(registryNumberCol, "registryNumber");

registryNumberCol.setMinWidth(220);

table.setItems(data);

table.getColumns().addAll(licensePlateCol, IdCarrierCol, modelCol, amountOfSitsCol, markCol,

registryNumberCol);

completeInfo.setFont(Font.*font*("Arial", FontWeight.***NORMAL***, FontPosture.***REGULAR***, 18));

emplyeeInfo.setText("Bus");

employeeBox.getChildren().clear();

employeeBox.getChildren().addAll(emplyeeInfo, table, managerOption, logout);

});

// Manager Option of showing all Employees

showEmployeesManager.setOnAction(e -> {

TableView table = **new** TableView();

ObservableList<Staff> data = FXCollections.*observableArrayList*(BusStation.*getStaff*());

TableColumn<Staff, String> idStaffCol = **new** TableColumn<Staff, String>("Staff ID");

tablesDesign(idStaffCol, "idStaff");

idStaffCol.setMinWidth(130);

TableColumn<Staff, String> surnameCol = **new** TableColumn<Staff, String>("Surname");

tablesDesign(surnameCol, "surname");

surnameCol.setMinWidth(130);

TableColumn<Staff, String> nameCol = **new** TableColumn<Staff, String>("Name");

tablesDesign(nameCol, "name");

nameCol.setMinWidth(130);

TableColumn<Staff, String> patronymicCol = **new** TableColumn<Staff, String>("Patronymic");

tablesDesign(patronymicCol, "patronymic");

patronymicCol.setMinWidth(130);

TableColumn<Staff, String> cityCol = **new** TableColumn<Staff, String>("City");

tablesDesign(cityCol, "city");

cityCol.setMinWidth(130);

TableColumn<Staff, String> streetCol = **new** TableColumn<Staff, String>("Street");

tablesDesign(streetCol, "street");

streetCol.setMinWidth(130);

TableColumn<Staff, String> houseNumberCol = **new** TableColumn<Staff, String>("House number");

tablesDesign(houseNumberCol, "houseNumber");

houseNumberCol.setMinWidth(130);

TableColumn<Staff, String> apartmentCol = **new** TableColumn<Staff, String>("Apartment");

tablesDesign(apartmentCol, "apartment");

apartmentCol.setMinWidth(130);

TableColumn<Staff, String> postCol = **new** TableColumn<Staff, String>("Post");

tablesDesign(postCol, "post");

postCol.setMinWidth(130);

TableColumn<Staff, String> phoneCol = **new** TableColumn<Staff, String>("Phone");

tablesDesign(phoneCol, "phone");

phoneCol.setMinWidth(130);

TableColumn<Staff, String> expirienceCol = **new** TableColumn<Staff, String>("Expirience");

tablesDesign(expirienceCol, "expirience");

expirienceCol.setMinWidth(130);

table.setItems(data);

table.getColumns().addAll(idStaffCol, surnameCol, nameCol, patronymicCol, cityCol, streetCol,

houseNumberCol, apartmentCol, postCol, phoneCol, expirienceCol);

completeInfo.setFont(Font.*font*("Arial", FontWeight.***NORMAL***, FontPosture.***REGULAR***, 16));

emplyeeInfo.setText("Staff");

employeeBox.getChildren().clear();

employeeBox.getChildren().addAll(emplyeeInfo, table, managerOption, logout);

});

}

@SuppressWarnings({ "rawtypes", "unchecked", "unused" })

**public** **void** BookTicketActions() {

customerBookTicket();

tripStarting.setValue("from");

tripStarting.getItems().clear();

tripEnding.getItems().clear();

tripStarting.getItems().addAll("Sevastopol");

tripEnding.getItems().addAll("Moscow", "Krasnodar", "Saint-Petersbourg");

ArrayList<Trips> choosenTrips = **new** ArrayList<Trips>();

showAvilableTrips.setOnAction(e -> {

**if** (!tripStarting.getValue().equals("from") && !tripEnding.getValue().equals("To")

&& tripStarting.getValue() != tripEnding.getValue() && !tripStarting.getValue().isEmpty()

&& !tripEnding.getValue().isEmpty()) {

**try** {

String ss = "select \* from trip,settlements where trip.departure\_station=? and settlements.city=? and trip.id\_settlements=settlements.id\_settlements";

BusStation.*con* = DbConnection.*connectToDB*();

PreparedStatement ps = BusStation.*con*.prepareStatement(ss);

ps.setString(1, tripStarting.getValue());

ps.setString(2, tripEnding.getValue());

BusStation.*rs* = ps.executeQuery();

String trips = "";

**while** (BusStation.*rs*.next()) {

**for** (**int** i = 0; i < BusStation.*getTrip*().size(); i++) {

**if** (BusStation.*getTrip*().get(i).getIdTrip() == BusStation.*rs*.getInt(1)) {

choosenTrips.add(BusStation.*getTrip*().get(i));

**break**;

}

}

}

**if** (choosenTrips.size() != 0) {

TableView table = **new** TableView();

table.setMaxHeight(170);

ObservableList<Trips> data = FXCollections.*observableArrayList*(choosenTrips);

TableColumn<Trips, String> idTrip = **new** TableColumn<Trips, String>("Trip ID");

bookingTripsTableDesign(idTrip, "idTrip");

TableColumn<Trips, String> departureTime = **new** TableColumn<Trips, String>("Time");

bookingTripsTableDesign(departureTime, "departureTime");

table.setItems(data);

table.getColumns().addAll(idTrip, departureTime);

bookingTicketVBox.getChildren().clear();

bookingTicketVBox.getChildren().addAll(table, choosecertainTripToBook, backToBockNewTickets,

logout);

} **else** {

tripStarting.setValue("");

tripEnding.setValue("");

}

} **catch** (Exception e1) {

}

} **else** {

tripStarting.setValue("");

tripEnding.setValue("");

}

});

doneEnterTripCode.setOnAction(e -> {

**for** (**int** i = 0; i < choosenTrips.size(); i++) {

**try** {

**if** (choosenTrips.get(i).getIdTrip() == Integer.*parseInt*(enterTripCode.getText())) {

ArrayList<Settlements> set = **new** ArrayList<Settlements>();

set = BusStation.*getSettlement*();

String citySetl = **null**;

**for** (Settlements s : set) {

**if**(s.getIdSettlements() == choosenTrips.get(i).getIdSettlements()) {

citySetl = s.getCity();

}

}

bookingDone.setText("Done You Booked " + numberOfTickets.getText() + " Tickets " + " From "

+ choosenTrips.get(i).getDepartureStation() + " To "

+ citySetl + " at "

+ choosenTrips.get(i).getDepartureTime());

enterTripCode.clear();

numberOfTickets.clear();

bookingTicketVBox.getChildren().add(bookingDone);

**break**;

} **else** {

bookingDone.setText("");

}

} **catch** (Exception ex) {

enterTripCode.clear();

numberOfTickets.clear();

}

}

enterTripCode.clear();

numberOfTickets.clear();

});

backToBockNewTickets.setOnAction(e ->

{

choosenTrips.clear();

customerBookTicket();

});

}

}

GUIDesign.java

**package** design;

**import** java.sql.SQLException;

**import** javafx.geometry.Pos;

**import** javafx.stage.Stage;

**import** javafx.scene.Scene;

**import** javafx.scene.layout.VBox;

**import** javafx.scene.layout.HBox;

**import** javafx.scene.text.Font;

**import** javafx.scene.text.FontPosture;

**import** javafx.scene.text.FontWeight;

**import** javafx.scene.text.Text;

**import** javafx.scene.control.Button;

**public** **class** GUIDesign **extends** PanesAndNodes {

Scene scene1 = **null**;

VBox firstWindowPane = **new** VBox();

Stage stage = **new** Stage();

**public** **void** start(Stage primaryStage) **throws** ClassNotFoundException, SQLException {

scene1 = **null**;

scene1 = **new** Scene(firstWindowPane, 500, 270);

primaryStage.setTitle("Bus Station");

primaryStage.setScene(scene1);

primaryStage.show();

primaryStage.setResizable(**false**);

}

**public** **void** welcomeWindow() {

buttons.setSpacing(30);

welcomeWindowDesign();

buttons.getChildren().clear();

buttons.getChildren().addAll(chooseEmployee, chooseCustomer, enter);

firstWindowPane.getChildren().clear();

firstWindowPane.getChildren().addAll(welcome, buttons);

}

// Employee Account design

HBox managerOption = **new** HBox();

**public** **void** informationPage() {

informationPageNodesDesign();

managerOption.setSpacing(10);

managerOption.getChildren().clear();

managerOption.getChildren().addAll(showTripsManager, showVehiclesManager, showEmployeesManager);

managerOption.setAlignment(Pos.***CENTER***);

employeeBox.getChildren().clear();

employeeBox.getChildren().addAll(welcomeEmployee, emplyeeInfo, completeInfo, managerOption, logout);

**if** (employeeBox.getScene() == **null**) {

Scene scene2 = **new** Scene(employeeBox, 650, 400);

stage.setScene(scene2);

} **else**

stage.setScene(employeeBox.getScene());

}

**int** checkTripType = 0;

Text bookingDone = **new** Text();

Button backToBockNewTickets = **new** Button("Back");

**public** **void** customerBookTicket() {

bookingDone.setFont(Font.*font*("Arial", FontWeight.***BOLD***, FontPosture.***REGULAR***, 16));

bookTicketDesign();

bookingTicketVBox.getChildren().clear();

bookingTicketVBox.getChildren().addAll(chooseTripType, destinationHBox, showAvilableTrips,

logout);

choosecertainTripToBook.getChildren().clear();

choosecertainTripToBook.getChildren().addAll(enterTripCode, numberOfTickets, doneEnterTripCode);

choosecertainTripToBook.setSpacing(20);

destinationHBox.setAlignment(Pos.***CENTER***);

destinationHBox.getChildren().clear();

destinationHBox.getChildren().addAll(from, to);

**if** (bookingTicketVBox.getScene() == **null**) {

Scene scene2 = **new** Scene(bookingTicketVBox, 650, 400);

stage.setScene(scene2);

} **else**

stage.setScene(bookingTicketVBox.getScene());

}

}

PanesAndNodes.java

**package** busStation;

**public** **class** Bus **extends** BusStation{

**private** **int** licensePlate;

**private** **int** idCarrier;

**private** String model;

**private** **int** amountOfSits;

**private** String mark;

**private** **int** registryNumber;

**public** Bus(**int** licensePlate,**int** idCarrier, String model,**int** amountOfSits, String mark, **int** registryNumber){

**this**.licensePlate = licensePlate;

**this**.idCarrier = idCarrier;

**this**.model = model;

**this**.amountOfSits = amountOfSits;

**this**.mark = mark;

**this**.registryNumber = registryNumber;

}

**public** **int** getLicensePlate() {

**return** licensePlate;

}

**public** **int** getIdCarrier() {

**return** idCarrier;

}

**public** String getModel() {

**return** model;

}

**public** **int** getAmountOfSits() {

**return** amountOfSits;

}

**public** String getMark() {

**return** mark;

}

**public** **int** getRegistryNumber() {

**return** registryNumber;

}

}

Bus.java

**package** busStation;

**public** **class** Bus **extends** BusStation{

**private** **int** licensePlate;

**private** **int** idCarrier;

**private** String model;

**private** **int** amountOfSits;

**private** String mark;

**private** **int** registryNumber;

**public** Bus(**int** licensePlate,**int** idCarrier, String model,**int** amountOfSits, String mark, **int** registryNumber){

**this**.licensePlate = licensePlate;

**this**.idCarrier = idCarrier;

**this**.model = model;

**this**.amountOfSits = amountOfSits;

**this**.mark = mark;

**this**.registryNumber = registryNumber;

}

**public** **int** getLicensePlate() {

**return** licensePlate;

}

**public** **int** getIdCarrier() {

**return** idCarrier;

}

**public** String getModel() {

**return** model;

}

**public** **int** getAmountOfSits() {

**return** amountOfSits;

}

**public** String getMark() {

**return** mark;

}

**public** **int** getRegistryNumber() {

**return** registryNumber;

}

}

BusStation.java

package busStation;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.sql.Connection;

import java.sql.ResultSet;

public abstract class BusStation {

private static ArrayList<Bus> bus = new ArrayList<Bus>();

private static ArrayList<Staff> staff = new ArrayList<Staff>();

private static ArrayList<Trips> trip = new ArrayList<Trips>();

private static ArrayList<Settlements> settlement = new ArrayList<Settlements>();

public static Connection con;

public static ResultSet rs;

public static ArrayList<Bus> getBus() {

return bus;

}

public static ArrayList<Staff> getStaff() {

return staff;

}

public static ArrayList<Trips> getTrip() {

return trip;

}

public static ArrayList<Settlements> getSettlement() {

return settlement;

}

public static void readDataFromDB() throws SQLException, ClassNotFoundException {

con = DbConnection.connectToDB();

Statement s = con.createStatement();

rs = s.executeQuery("Select \* from bus");

// read bus data

while (rs.next())

bus.add(new Bus(rs.getInt(1), rs.getInt(2), rs.getString(3), rs.getInt(4), rs.getString(5), rs.getInt(6)));

// read staff data

rs = s.executeQuery("Select \* from staff");

while (rs.next())

staff.add(new Staff(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5),

rs.getInt(6), rs.getInt(7),rs.getInt(8),rs.getString(9),rs.getInt(10),rs.getInt(11)));

// read trip data

rs = s.executeQuery("Select \* from trip");

while (rs.next())

trip.add(new Trips(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getInt(4), rs.getString(5),

rs.getString(6), rs.getInt(7)));

// read settlement data

rs = s.executeQuery("Select \* from settlements");

while (rs.next())

settlement.add(new Settlements(rs.getInt(1), rs.getString(2), rs.getString(3)));

}

}

DbConnection.java

package busStation;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public abstract class DbConnection {

public static Connection connectToDB() throws SQLException, ClassNotFoundException{

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/bus\_station?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC","root","Root1234!");

return con;

}

}

Settlements.java

**package** busStation;

**public** **class** Settlements **extends** BusStation{

**private** **int** idSettlements;

**private** String country;

**private** String city;

**public** Settlements(**int** idSettlements, String country, String city) {

**this**.idSettlements = idSettlements;

**this**.country = country;

**this**.city = city;

}

**public** String getCity() {

**return** city;

}

**public** String getCountry() {

**return** country;

}

**public** **int** getIdSettlements() {

**return** idSettlements;

}

}

Staff.java

**package** busStation;

**public** **class** Staff **extends** BusStation {

**private** **int** idStaff;

**private** String surname;

**private** String name;

**private** String patronymic;

**private** String city;

**private** **int** street;

**private** **int** houseNumber;

**private** **int** apartment;

**private** String post;

**private** **int** phone;

**private** **int** expirience;

**public** Staff(**int** idStaff, String surame, String name, String patronymic, String city, **int** street, **int** houseNumber,

**int** apartment, String post, **int** phone, **int** expirience) {

**this**.idStaff = idStaff;

**this**.surname = surame;

**this**.name = name;

**this**.patronymic = patronymic;

**this**.city = city;

**this**.street = street;

**this**.houseNumber = houseNumber;

**this**.apartment = apartment;

**this**.post = post;

**this**.phone = phone;

**this**.expirience = expirience;

}

**public** **int** getApartment() {

**return** apartment;

}

**public** String getCity() {

**return** city;

}

**public** **int** getExpirience() {

**return** expirience;

}

**public** **int** getHouseNumber() {

**return** houseNumber;

}

**public** **int** getIdStaff() {

**return** idStaff;

}

**public** String getName() {

**return** name;

}

**public** String getPatronymic() {

**return** patronymic;

}

**public** **int** getPhone() {

**return** phone;

}

**public** String getPost() {

**return** post;

}

**public** **int** getStreet() {

**return** street;

}

**public** String getSurname() {

**return** surname;

}

}

Trips.java

**package** busStation;

**public** **class** Trips **extends** BusStation {

**private** **int** idTrip;

**private** String departureTime;

**private** String arrivalTime;

**private** **int** distance;

**private** String schedule;

**private** String departureStation;

**private** **int** idSettlements;

**public** Trips(**int** idTrip, String departureTime, String arrivalTime, **int** distance, String schedule,

String departureStation, **int** idSettlements) {

**super**();

**this**.arrivalTime = arrivalTime;

**this**.departureStation = departureStation;

**this**.departureTime = departureTime;

**this**.distance = distance;

**this**.idSettlements = idSettlements;

**this**.idTrip = idTrip;

**this**.schedule = schedule;

}

**public** String getArrivalTime() {

**return** arrivalTime;

}

**public** String getDepartureStation() {

**return** departureStation;

}

**public** String getDepartureTime() {

**return** departureTime;

}

**public** **int** getDistance() {

**return** distance;

}

**public** **int** getIdSettlements() {

**return** idSettlements;

}

**public** **int** getIdTrip() {

**return** idTrip;

}

**public** String getSchedule() {

**return** schedule;

}

}