Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования
Кафедра проектирования информационно-компьютерных систем
Дисциплина «Структуры и базы данных»

«К защите допустить»
Руководитель курсового проекта
магистр техн. наук, ассистент
_____А.В. Шелест
____.____.2019

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:

«БАЗА ДАННЫХ ДЛЯ ПОДДЕРЖКИ РАБОТЫ ЖЕЛЕЗНОДОРОЖНОГО ВОКЗАЛА»

БГУИР КП 1-39 03 02 024 ПЗ

Выполнил студент группы 713802
СИНИЛО Артур Викторович

_____
(подпись студента)
Курсовой проект представлен на проверку ____.____.2019
_____
(подпись студента)

Минск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день практически любая информационная система, предполагающая какую-либо работу с упорядоченными наборами данных (создание, хранение, доступ, редактирование, удаление и т. д.), использует для этих целей системы управления базами данных (СУБД).

Исторически первыми появились реляционные базы данных. Базы данных такого типа состоят из сущностей, связанных между собой отношениями (*1:1*, *1:n* или *m:n*). Сущности одного типа хранятся вместе и образуют таблицы. Каждая сущность может содержать произвольное количество полей различных типов: целочисленные, дробные с фиксированной или плавающей запятой, строковые, перечисления (*enum*) и др. Над сущностями возможно проведение различных операций реляционной алгебры: объединение, соединение, декартово произведение и т. д. Примеры реляционных СУБД: *MySQL*, *PostgreSQL*, *MariaDB*, *Oracle* [1].

Однако прогресс не стоит на месте. Количество активных пользователей Интернета растёт с каждым годом. Как следствие, увеличиваются объёмы информации, которую нужно хранить. Кроме того, пользователи Сети разбросаны по всей планете. Поэтому с целью уменьшения задержки крупные компании стали устанавливать свои серверы как можно ближе к конечным потребителям. В результате возникла необходимость в распределённых базах данных, когда данные хранятся на множестве отдельных серверных кластеров. К сожалению, обычные реляционные базы данных плохо подходят для этой цели. Поэтому были разработаны так называемые *NoSQL* базы данных. Одним из подклассов *NoSQL* являются хранилища типа «ключ-значение» – по сути огромные распределённые хэш-таблицы. Ещё один подкласс *NoSQL* – документоориентированные базы данных, но существуют и другие. Примеры *NoSQL* СУБД: *MongoDB*, *Berkeley DB*, *Redis*, *Apache Cassandra* [2].

Несмотря на кажущиеся очевидными преимущества *NoSQL* баз данных над традиционными, в реальности использование *NoSQL* обычно оправдано только для крупных транснациональных корпораций, которые предоставляют свои услуги по всему миру. Для более типичных сценариев использования реляционные БД обычно оказываются эффективнее.

К базе данных, используемой в любом серьёзном проекте, обычно предъявляются следующие требования:

- стабильность и надёжность работы, непосредственно влияющие на устойчивую работу всего приложения;
- минимальная избыточность;
- целостность данных, обеспечиваемая с помощью различных средств, таких как ограничения *NOT NULL* и *UNIQUE*, триггеры, валидация на стороне клиента и другие;
- наличие всех необходимых сущностей для моделирования предметной области в полной мере;
- быстрая обработка запросов, наиболее часто встречающихся при решении рассматриваемой задачи;
- поддержка современных средств разработки и администрирования.

Цель курсового проекта – разработка web-сайта железнодорожной компании. Он должен содержать следующие функции:

– просмотр списка поездов, следующих между двумя станциями, на определённую дату;

– регистрация, аутентификация и авторизация пользователей;

– покупка билетов на поезд от одной станции до другой;

– возврат купленных билетов;

– виртуальное табло для любой станции;

– панель администратора с возможностью добавлять и удалять станции, а также редактировать их названия;

– адаптивный дизайн для удобства работы с мобильных устройств.

Задачи, которые необходимо решить для достижения конечной цели:

– проектирование в MySQL Workbench базы данных с учётом особенностей предметной области и типичных сценариев использования;

– написание серверного приложения на языке программирования *Rust* с использованием каркаса веб-приложений *actix-web*, шаблонизатора *yarte* и библиотеки объектно-реляционного отображения *diesel*;

– создание шаблонов страниц сайта, выбор дизайна и, если необходимо, написание *JavaScript* кода для избежания нежелательных полных обновлений страницы при изменении небольшой части содержимого;

– тестирование и исправление недоработок в проекте;

– создание *EER* и *UML* диаграмм по результатам проектирования.

Курсовой проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 85%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников». Скриншот приведён в приложении.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЁ ФОРМАЛИЗАЦИЯ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ

1.1 Описание предметной области

Предметной областью данного курсового проекта являются железнодорожные пассажирские перевозки. Железнодорожные перевозки включают в себя множество аспектов. В обязанности работников железной дороги входит обеспечение бесперебойного функционирования всех узлов системы: железнодорожного полотна, поездов, стрелочных переходов, светофоров, вокзалов, системы продажи билетов и т. д.

Система продажи билетов хорошо ложится на общую концепцию реляционных баз данных. Поэтому в качестве предмета проектирования было решено выбрать систему продажи проездных документов.

1.2 Анализ информационных потребностей пользователей

Проведя анализ информационных потребностей пользователей, можно сделать следующие выводы. Сайт железнодорожной компании, с одной стороны, должен позволять пользователям регистрироваться, авторизоваться и смотреть расписание поездов. С другой стороны, руководство железной дороги должно иметь возможность вносить изменения в базу данных, например, добавлять, удалять или изменять названия станций.

Систематизируя вышесказанное, к основным функциям сайта с точки зрения гостей сайта (неавторизованных пользователей) можно отнести:

– получение сведений о расписании поездов;

– просмотр виртуального табло;

– регистрация и вход в систему.

Функции сайта с точки зрения авторизованных пользователей:

– получение сведений о расписании поездов;

– просмотр виртуального табло;

– просмотр информации об аккаунте.

К основным функциям сайта с точки зрения администрации железной дороги можно отнести:

– изменение типов вагонов;

– изменение списка станций.

1.3 Определение требований и ограничений к базе данных с точки зрения предметной области

В качестве базы данных для данного курсового проекта было принято решение использовать *MariaDB*. Данное решение связано с удобством моделирования базы данных в программе *MySQL Workbench*. *MariaDB* является форком базы *MySQL*.

Проанализировав предметную область, можно сформулировать следующие требования и ограничения к разрабатываемой базе данных:

– запрещено регистрировать несколько пользователей с одним и тем же адресом электронной почты;

– не разрешается создание станций с одинаковым названием;

– названия типов поездов также не могут повторяться;

– номер поезда, номера вагонов, количество мест в вагоне определённого типа, а также стоимость билета являются неотрицательными целыми числами;

– на абсолютное большинство полей наложено ограничение *NOT NULL*, исключением являются параметры рейса, которые могут быть неизвестны в определённые моменты времени (например, номера пути и платформы, куда прибывает поезд, неизвестны, пока поезд находится вдали от железнодорожных станций);

– в целях простоты изменения различных полей таблиц в качестве основных ключей во всех таблицах используются суррогатные ключи, имеющие тип беззнакового целого.

С добавлением поездов, содержащих большое количество вагонов, растёт нагрузка на аппаратное обеспечение. Поэтому в качестве сервера базы данных желательно использовать достаточно мощный компьютер, чтобы избежать зависаний и отказов в обслуживании.

1.4 Постановка решаемой задачи

Целью данного курсового проекта является создание веб-приложения для обеспечения работы пассажирских железнодорожных перевозок.

Для достижения поставленной задачи следует спроектировать и создать базу данных, а также разработать веб-сайт, подходящий для удобного использования администрацией железной дороги, зарегистрированными пользователями и гостями.

Проектирование базы данных включает следующие этапы:

– исследование предметной области;

– анализ данных (сущностей и их атрибутов);

– определение отношений между сущностями и определение первичных и внешних ключей.

Последовательность выполнения задачи: проектирование и создание базы данных с помощью *MySQL Workbench*, оформление сайта с помощью языка гипертекстовой разметки *HTML* и таблиц стилей *CSS (Bootstrap),* написание серверного приложения на языке программирования *Rust* с использованием фреймворка *actix-web*, шаблонизатора *yarte* и библиотеки объектно-реляционного отображения *diesel.*

2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ДЛЯ ОСНОВНОГО ВИДА ДЕЯТЕЛЬНОСТИ РАССМАТРИВАЕМОЙ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Разработка инфологической модели предметной области базы данных

Для начала необходимо разобраться, что такое инфологическая модель предметной области базы данных. Во-первых, модель базы данных – это описание базы данных с помощью определенного языка. Инфологическая модель данных представляет собой описание предметной области без привязки к СУБД и предназначена для людей. На рисунке 1 отображена семантическая инфологическая модель, где прерывистыми линиями обозначены связи между объектами.

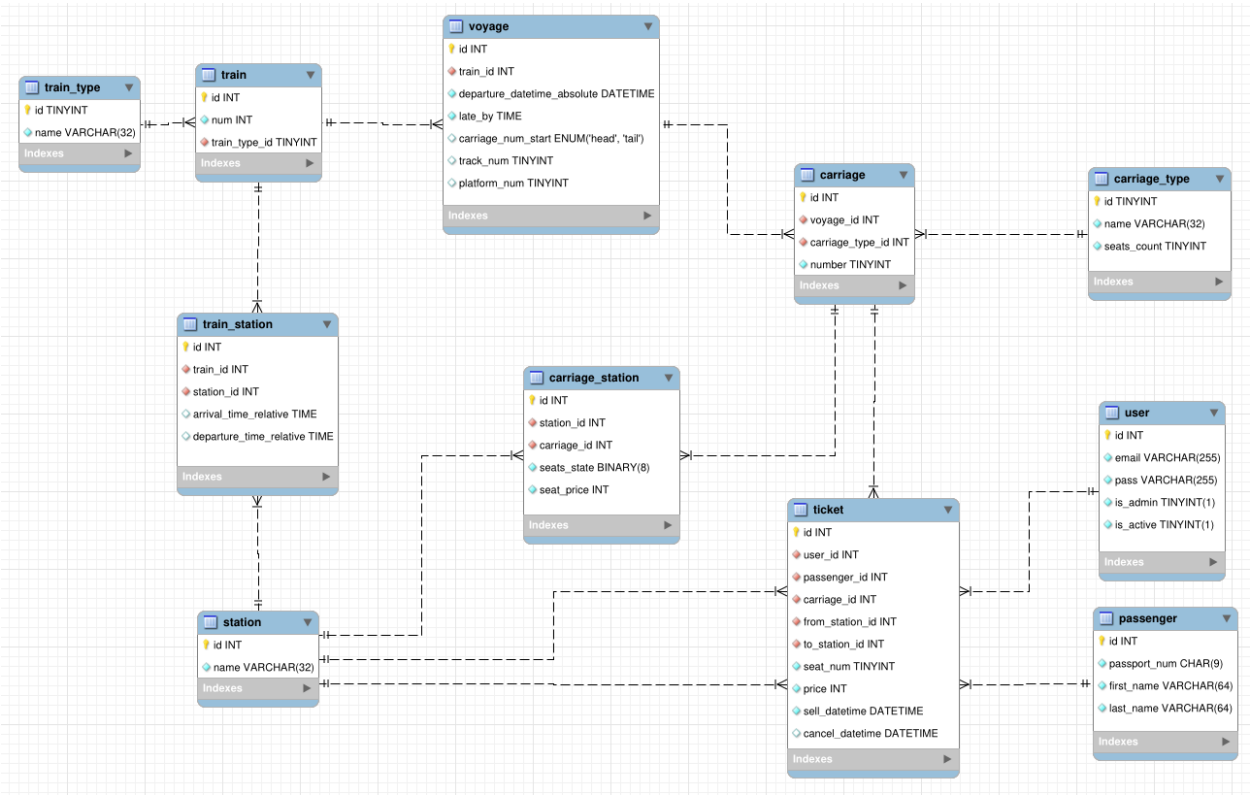


Рисунок 2.1 – *ER*-диаграмма (инфологическая модель) базы данных

Инфологическая модель была разработана, основываясь на изысканиях, проведённых в разделе 1.

Для поддержания БД в устойчивом состоянии используется ряд механизмов, которые получили обобщённое название средств поддержки

целостности. Приведение структуры БД в соответствие этим ограничениям называется нормализацией.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться единственный раз, поскольку дублирование может привести к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации [3].

Для реляционных моделей данных существует порядка восьми нормальных форм. На практике при проектировании БД достижение третьей нормальной формы (далее 3НФ) считается достаточным.

Первая нормальная форма (далее 1НФ) говорит, что каждый атрибут отношения должен хранить атомарное значение, каждое отношение (строка в таблице) должно содержать одинаковое количество атрибутов (столбцов):

- запрещает повторяющиеся столбцы (содержащие одинаковую по смыслу информацию);
- запрещает множественные столбцы (содержащие значения типа списка и т.п.);
- требует определить первичный ключ для таблицы, то есть тот столбец или комбинацию столбцов, которые однозначно определяют каждую строку.

Вторая нормальная форма (далее 2НФ) говорит, что отношение находится во 2НФ, если оно находится в 1НФ, и при этом все неключевые атрибуты зависят только от первичного ключа:

- 2НФ требует, чтобы неключевые столбцы таблиц зависели от первичного ключа в целом, но не от его части;
- если таблица находится в первой нормальной форме и первичный ключ у неё состоит из одного столбца, то она автоматически находится и во второй нормальной форме.

Отношение находится в третьей нормальной форме (далее 3НФ), если оно находится во 2НФ и каждый неключевой атрибут зависит только от первичного ключа и не зависят друг от друга. Говоря другими словами, для каждой таблицы должно соблюдаться условие *нетранзитивности*.

В базе данных созданые таблицы и связи между ними на основе механизма внешних ключей (*foreign key*). Всего в базе данных *railway* есть 11 таблиц:

– *train_type* – типы поездов (например, городских, межрегиональных, региональных линий и т. д.);

– *train* – номера поездов;
– *station* – названия станций;
– *carriage_type* – типы вагонов (купе, СВ, плацкарт и т. д.);
– *carriage* – вагоны, количество мест в них;
– *passenger* – личные данные пассажиров (ФИО, номер паспорта);
– *user* – электронные адреса и хэши паролей пользователей;
– *carriage_station* – состояние мест в вагонах (занято/свободно);
– *voyage* – рейсы поездов, дата и время отправления;
– *ticket* – проездные документы;
– *train_station* – маршруты движения поездов.

2.2 Выбор и обоснование используемых типов, данных и ограничений (доменов)

Следующим этапом разработки базы данных является выбор используемых типов данных. В *MariaDB* существует много типов данных, вот некоторые из них:

- числовые данные – как целые, так и вещественные;
- строковые данные;
- календарные, или временные данные – используются для записи даты и времени, поддерживают различный формат записи.

При проектировании базы данных, соответствующей предметной области, были использованы следующие типы данных :

- *INT* – для основных ключей и больших чисел*;*
- *TINYINT* – для малых чисел (номер места в вагоне и т. д.);
- *VARCHAR* – для строковых данных и хэш-сумм паролей;
- *DATETIME* – для даты и времени;
- *TIME* – для времени прибытия, отправления и т. д.
- *ENUM* – для указания, с какой стороны начинается нумерация вагонов состава поезда (с головы или с хвоста).

2.3 Проектирование запросов к базе данных

К базе данных необходимо спроектировать запросы. Простые запросы могут быть сгенерированы автоматически при помощи библиотеки объектно-реляционного отображения, см. рисунки 2.2 и 2.3.

```rust
fn get_stations(pool: web::Data<Pool>) -> Result<Vec<Station>, diesel::resu
    use crate::schema::station::dsl::*;
    let id_stations = station.load(&pool.get().unwrap());
    let stations = id_stations.map(|s| s.iter().map(|st| Station::from(st))
    stations
```

Рисунок 2.2 – Запрос на получение списка станций

```rust
    user.filter(email.eq(mail))
        .first::<IdUser>(&pool.get().unwrap())
        .ok()
```

Рисунок 2.3 – Запрос на поиск текущего пользователя

Другие запросы, как например запрос для вывода расписания поездов между станциями, сложны, поэтому их нужно записывать на языке *SQL*:

```
SELECT
      voyage.id as voyage_id,
      train.num as train_num,
      train_type.name as train_type,
      first_st.name as first_station,
      last_st.name as last_station,
      from_st.name as from_station,
      to_st.name as to_station,
      TIME(ADDTIME(voyage.departure_datetime_absolute,
           from_tr_st.departure_time_relative)) as depart_time,
      TIME(ADDTIME(voyage.departure_datetime_absolute,
           to_tr_st.arrival_time_relative)) as arrival_time,
      TIMEDIFF(
                TIME(ADDTIME(voyage.departure_datetime_absolute,
                    to_tr_st.arrival_time_relative)),
                TIME(ADDTIME(voyage.departure_datetime_absolute,
                from_tr_st.departure_time_relative))
           ) as on_the_way_time
        FROM voyage
        JOIN train ON train.id = voyage.train_id
            AND DATE(voyage.departure_datetime_absolute) = '{date}'
        JOIN train_type ON train_type.id = train.train_type_id

JOIN train_station first_tr_st
     ON first_tr_st.train_id = train.id
     AND first_tr_st.arrival_time_relative IS NULL
JOIN station first_st ON first_st.id = first_tr_st.station_id

JOIN train_station last_tr_st ON last_tr_st.train_id = train.id
            AND last_tr_st.departure_time_relative IS NULL
JOIN station last_st ON last_st.id = last_tr_st.station_id

JOIN train_station from_tr_st ON from_tr_st.train_id = train.id
```

```
            AND from_tr_st.departure_time_relative IS NOT NULL
JOIN station from_st ON from_st.id = from_tr_st.station_id
            AND from_st.name = '{from}'

JOIN train_station to_tr_st ON to_tr_st.train_id = train.id
            AND to_tr_st.arrival_time_relative IS NOT NULL
            AND to_tr_st.arrival_time_relative >
                  from_tr_st.departure_time_relative
JOIN station to_st ON to_st.id = to_tr_st.station_id
      AND to_st.name = '{to}'

ORDER BY depart_time, arrival_time;
```

2.4 Программная реализация и документирование базы данных

Для наглядного представления проделанной работы над базой данных была выбрана программа *MySQL Workbench*, являющаяся инструментом для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию базы данных в единое бесшовное окружение.

База данных содержит 11 таблиц:

- *train_type* – типы поездов (например, городских, межрегиональных, региональных линий и т. д.);
- *train* – номера поездов;
- *station* – названия станций;
- *carriage_type* – типы вагонов (купе, СВ, плацкарт и т. д.);
- *carriage* – вагоны, количество мест в них;
- *passenger* – личные данные пассажиров (ФИО, номер паспорта);
- *user* – электронные адреса и хэши паролей пользователей;
- *carriage_station* – состояние мест в вагонах (занято/свободно);
- *voyage* – рейсы поездов, дата и время отправления;
- *ticket* – проездные документы;
- *train_station* – маршруты движения поездов.

Подробное описание стобцов каждой из представленных таблиц см. в приложении А к пояснительной записке.

2.5 Проектирование разрабатываемого приложения

В качестве языка программирования для написания сереверной части веб-приложения был выбран *Rust*. Он гарантирует безопасную работу с памятью с проверкой на этапе компиляции и без использования сборщика

мусора. Применены фреймворк *actix-web*, шаблонизатор *yarte* и *ORM diesel*. Для клиентской части приложения были использованы языки разметки *HTML* и *CSS.* Некоторые функции требуют поддержки в браузере *JavaScript.*

С целью обеспечения безопасности пользовательских данных на этапе передачи используется шифрование по протоколу *TLS.* Пароли пользователей в базе данных хранятся в виде хешей и соли. Хеширование выполняется по алгоритму *Argon2*. Данный алгоритм был признан лучшим алгоритмом шифрования паролей в 2015 году.

При проектировании веб-приложения использована архитектура, близкая по своей сути к широко применяемому шаблону проектирования *MVC (Model – View – Controller).*

3 ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ БАЗЫ ДАННЫХ

3.1 Руководство пользователя

В разработанном веб-приложении посетитель сразу же попадает на страницу выбора маршрута (рисунок 3.1).

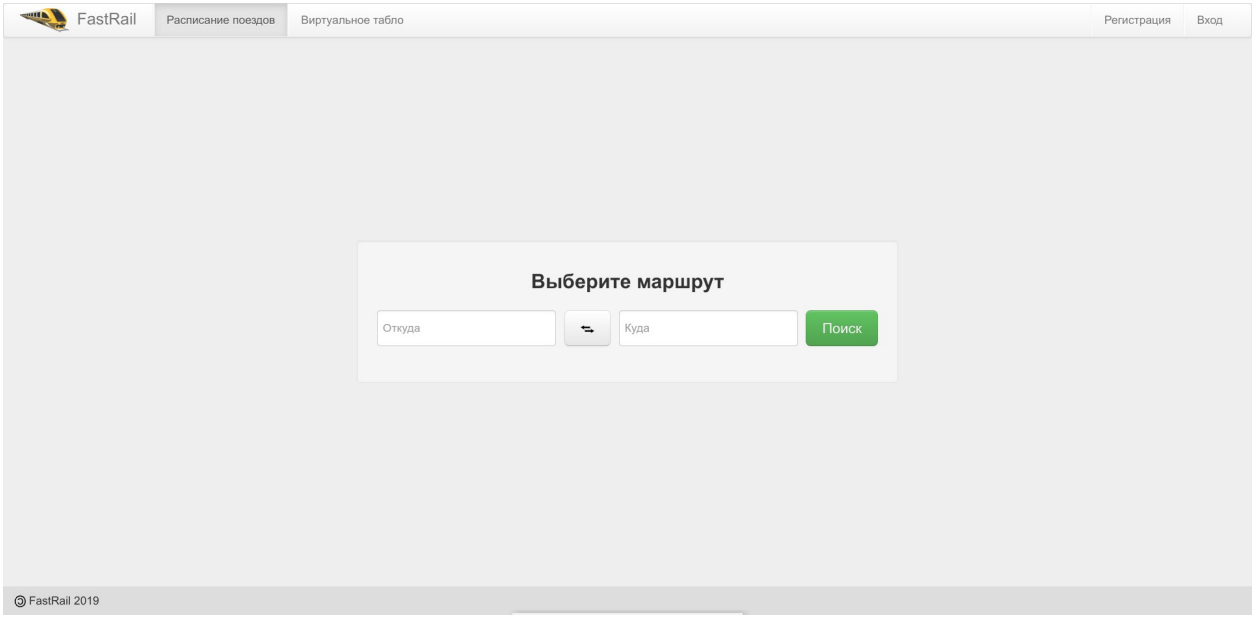


Рисунок 3.1 – Выбор маршрута

В случае выбора одной станции и не выбора другой будет выдана ошибка (рисунок 3.2)

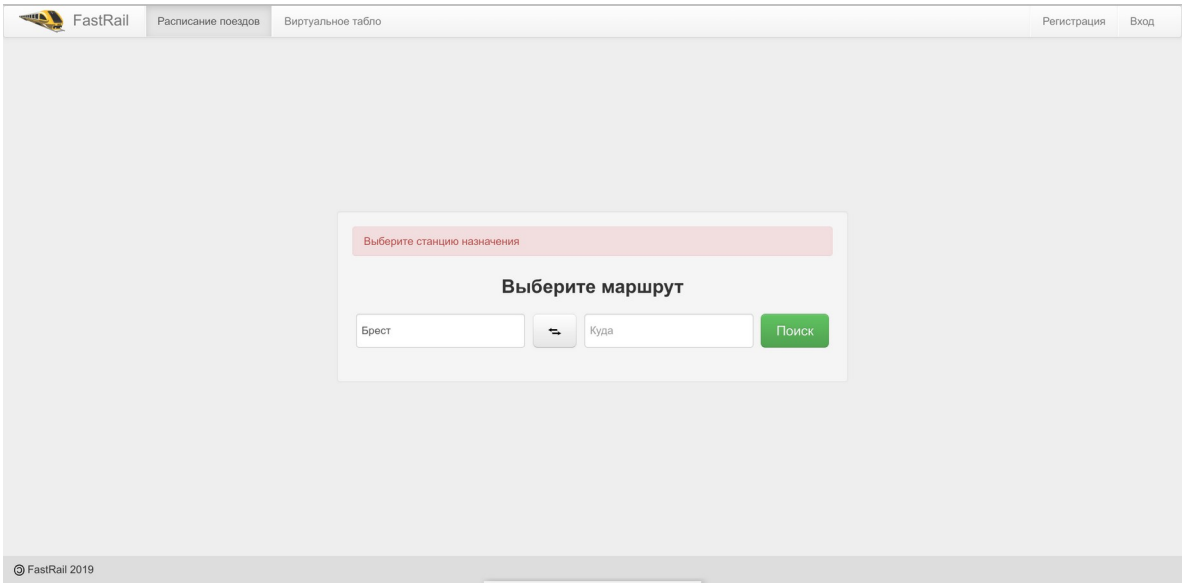
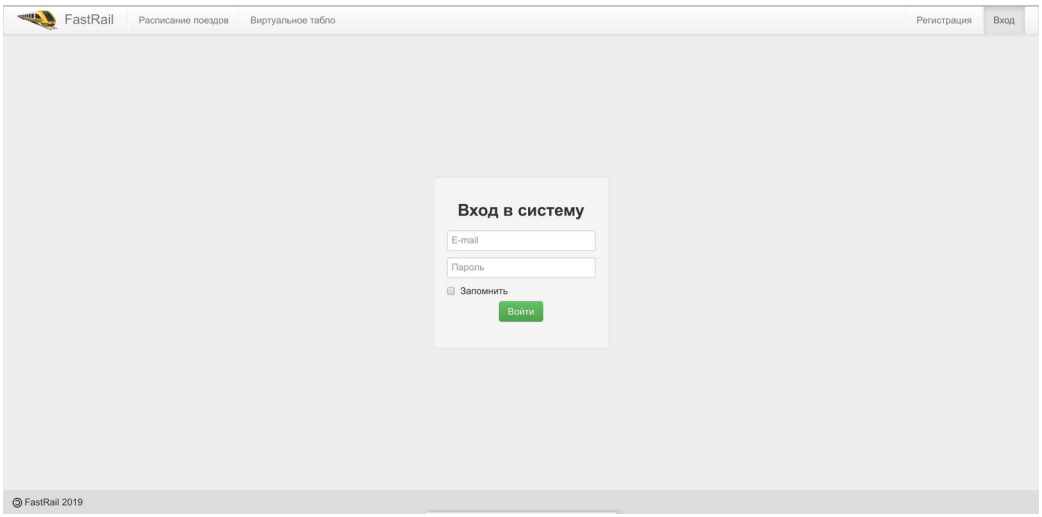


Рисунок 3.2 – Ошибка выбора станции

Для входа в систему необходимо ввести *E-mail* и пароль (рис. 3. 3)



3.2 Администрирование базы данных

Администрирование базы данных осуществляется администраторами сайта. В его основные обязанности входит заполнение базы данных актуальной информацией, обновление, редактирование и удаление устаревшей информации.

Также администратор отвечает за работоспособность базы данных и веб-приложения, а также исправляет ошибки, которые могут возникнуть во время использования веб-приложения пользователями.

3.3 Реализация клиентских запросов

Запросы к базе данных генерируются *ORM* (англ. *Object Relational Mapping* – представление реляционных БД средствами ООП). В данном проекте было принято решения использовать *ORM diesel*. Это самая популярная система подобного рода для языка *Rust*. Кроме популярности, к её преимуществам можно отнести наличие хорошей документации.

Для сложных запросов использован язык *SQL*.

3.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных

Одной из наиболее опасных угроз ИТ-безопасности является несанкционированный доступ к конфиденциальной информации. По уровню наносимого ущерба этот вид угроз ненамного уступает только вирусным

инфекциям. В базах данных несанкционированный доступ может быть реализован на двух этапах – при входе в базу данных (путем фальсификации процедур идентификации/аутентификации) и на стадии авторизации.

По мере того, как деятельность организаций всё больше зависит от компьютерных информационных технологий, проблемы защиты баз данных становятся всё более актуальными. Угрозы потери конфиденциальной информации стали обычным явлением в современном компьютерном мире. Каждый сбой работы базы данных может парализовать работу целых корпораций, банков, что приведет к ощутимым материальным потерям. Защита данных становится одной из самых актуальных проблем в современных компьютерных технологиях.

Один из самых эффективных и простых методов защиты информации пользователей – это защита паролем. Этого недостаточно, так как пароли могут быть украдены или сфальсифицированы, поэтому необходимо шифровать пароли.

Как уже было упомянуто выше, в проекте используется шифрование паролей по алгоритму *Argon2*. Для этого служит библиотека *argonautica.*

ЗАКЛЮЧЕНИЕ

При разработке курсового проекта были выполнены следующие задачи:

– обоснован выбор средств разработки базы данных: системы управления базами данных и языка программирования;

– сформулирована концептуальная модель данных;

– разработана база данных вокзала;

– разработан веб-сайт, использующий базу данных.

При выборе системы управления базами данных основными критериями выбора были: скорость работы, наличие полной документации. Для некоторых таблиц разработанной базы можно добавлять, изменять, удалять и просматривать данные.

Программа обеспечивает защиту пользовательских данных благодаря использованию шифрования по протоколу TLS. Была использована реализация протокола в виде библиотеки rustls. Для учебных целей был использован самоподписанный сертификат. В случае использования разработанной системы в реальном мире возможна его замена.

Интерфейс системы был разработан в очень сжатые сроки, поэтому он является довольно простым, но в то же время функциональность осталась на приемлемом. Реализованы следующие функции обработки данных: обновление базы данных, изменение (редактирование и удаление) данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Копылова, Н. Базы данных – революционная эволюция/ Н. Копылова // Лаборатория рекламы, маркетинга и *public relations* [Электронный ресурс]. – 2008. – Режим доступа: http://www.advlab.ru/articles/article378.htm. – Дата доступа: 02.01.2020.

[2] Понятие о данных. [Электронный ресурс] – 2012. – Режим доступа: https://docplayer.ru/ – Дата доступа: 02.01.2020.

[3] Нормализация баз данных // Межрегиональный Открытый Социальный Институт [Электронный ресурс]. – 2013. – Режим доступа: http://studfile.net/preview/5150865/page:14/ – Дата доступа: 03.01.2020.

ПРИЛОЖЕНИЕ А
(обязательное)
Скрипт генерации БД

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE
,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema railway
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema railway
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `railway` DEFAULT CHARACTER SET utf8mb4 ;
USE `railway` ;

-- -----------------------------------------------------
-- Table `railway`.`carriage_type`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`carriage_type` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(32) NOT NULL,
  `seats_count` TINYINT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `name_UNIQUE` (`name` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`passenger`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`passenger` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `passport_num` CHAR(9) CHARACTER SET 'utf8' COLLATE 'utf8_general_ci' NOT
NULL,
  `first_name` VARCHAR(64) CHARACTER SET 'utf8' COLLATE 'utf8_general_ci' NOT
NULL,
  `last_name` VARCHAR(64) CHARACTER SET 'utf8' COLLATE 'utf8_general_ci' NOT
NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `email` (`passport_num` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

```
-- -----------------------------------------------------
-- Table `railway`.`train`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`train` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `num` INT UNSIGNED NOT NULL,
  `train_type_id` TINYINT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `num_UNIQUE` (`num` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`user` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(255) NOT NULL,
  `pass` VARCHAR(255) NOT NULL,
  `is_admin` TINYINT(1) UNSIGNED NOT NULL DEFAULT 0,
  `is_active` TINYINT(1) UNSIGNED NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`station`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`station` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `name_UNIQUE` (`name` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`ticket`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`ticket` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `user_id` INT UNSIGNED NOT NULL,
  `passenger_id` INT UNSIGNED NOT NULL,
  `carriage_id` INT UNSIGNED NOT NULL,
  `from_station_id` INT UNSIGNED NOT NULL,
  `to_station_id` INT UNSIGNED NOT NULL,
  `seat_num` TINYINT UNSIGNED NOT NULL,
  `price` INT UNSIGNED NOT NULL,
  `sell_datetime` DATETIME NOT NULL DEFAULT NOW(),
  `cancel_datetime` DATETIME NULL,
```

```
  PRIMARY KEY (`id`),
  INDEX `fk_user_id_idx` (`user_id` ASC),
  INDEX `fk_ticket_from_station_id_idx` (`from_station_id` ASC),
  INDEX `fk_ticket_to_station_id_idx` (`to_station_id` ASC),
  CONSTRAINT `fk_ticket_user_id`
    FOREIGN KEY (`user_id`)
    REFERENCES `railway`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_ticket_from_station_id`
    FOREIGN KEY (`from_station_id`)
    REFERENCES `railway`.`station` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_ticket_to_station_id`
    FOREIGN KEY (`to_station_id`)
    REFERENCES `railway`.`station` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`carriage`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`carriage` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `voyage_id` INT UNSIGNED NOT NULL,
  `carriage_type_id` INT UNSIGNED NOT NULL,
  `number` TINYINT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`train_type`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`train_type` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `name_UNIQUE` (`name` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`train_station`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`train_station` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
  `train_id` INT UNSIGNED NOT NULL,
  `station_id` INT UNSIGNED NOT NULL,
  `arrival_time_relative` TIME NULL,
  `departure_time_relative` TIME NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_train_id_idx` (`train_id` ASC),
  INDEX `fk_station_id_idx` (`station_id` ASC),
  CONSTRAINT `fk_train_station_train_id`
    FOREIGN KEY (`train_id`)
    REFERENCES `railway`.`train` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_train_station_station_id`
    FOREIGN KEY (`station_id`)
    REFERENCES `railway`.`station` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`voyage`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`voyage` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `train_id` INT UNSIGNED NOT NULL,
  `departure_datetime_absolute` DATETIME NOT NULL,
  `late_by` TIME NOT NULL DEFAULT '00:00',
  `carriage_num_start` ENUM('head', 'tail') NULL,
  `track_num` TINYINT UNSIGNED NULL,
  `platform_num` TINYINT UNSIGNED NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_train_id_idx` (`train_id` ASC),
  CONSTRAINT `fk_voyage_train_id`
    FOREIGN KEY (`train_id`)
    REFERENCES `railway`.`train` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `railway`.`carriage_station`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `railway`.`carriage_station` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `station_id` INT UNSIGNED NOT NULL,
  `carriage_id` INT UNSIGNED NOT NULL,
  `seats_state` BIGINT(8) UNSIGNED NOT NULL,
  `seat_price` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
```

```
  INDEX `fk_carriage_id_idx` (`carriage_id` ASC),
  INDEX `fk_station_id_idx` (`station_id` ASC),
  CONSTRAINT `fk_carriage_station_station_id`
    FOREIGN KEY (`station_id`)
    REFERENCES `railway`.`station` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_carriage_station_carriage_id`
    FOREIGN KEY (`carriage_id`)
    REFERENCES `railway`.`carriage` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинги программного кода

```
mod controllers;
mod models;
mod responders;
mod schema;
mod templates;

use actix_identity::{CookieIdentityPolicy, IdentityService};
use actix_web::{middleware, web, App, HttpServer};
use diesel::prelude::*;
use diesel::r2d2::{self, ConnectionManager};
use dotenv;
use rustls::internal::pemfile::{certs, pkcs8_private_keys};
use std::fs::File;
use std::io::BufReader;

#[macro_use]
extern crate diesel;

fn setup_rustls() -> rustls::ServerConfig {
    let mut cert_file =
BufReader::new(File::open("tls/cert.pem").unwrap());
    let mut key_file =
BufReader::new(File::open("tls/key.pem").unwrap());
    let cert_chain = certs(&mut cert_file).unwrap();
    let mut keys = pkcs8_private_keys(&mut key_file).unwrap();
    let mut config =
rustls::ServerConfig::new(rustls::NoClientAuth::new());
    config.set_single_cert(cert_chain, keys.remove(0)).unwrap();
    config
}

fn main() -> std::io::Result<()> {
    dotenv::dotenv().ok();
    let sys = actix_rt::System::new("railway");
    let connspec = std::env::var("DATABASE_URL").expect("DATABASE_URL
not found");
    let secret_key = std::env::var("SECRET_KEY").expect("SECRET_KEY not
found");
    let manager = ConnectionManager::<MysqlConnection>::new(connspec);
    let pool = r2d2::Pool::builder()
        .build(manager)
        .expect("Failed to create pool.");
    HttpServer::new(move || {
        App::new()
            .data(pool.clone())
            .wrap(middleware::Compress::default())
            .wrap(IdentityService::new(
```

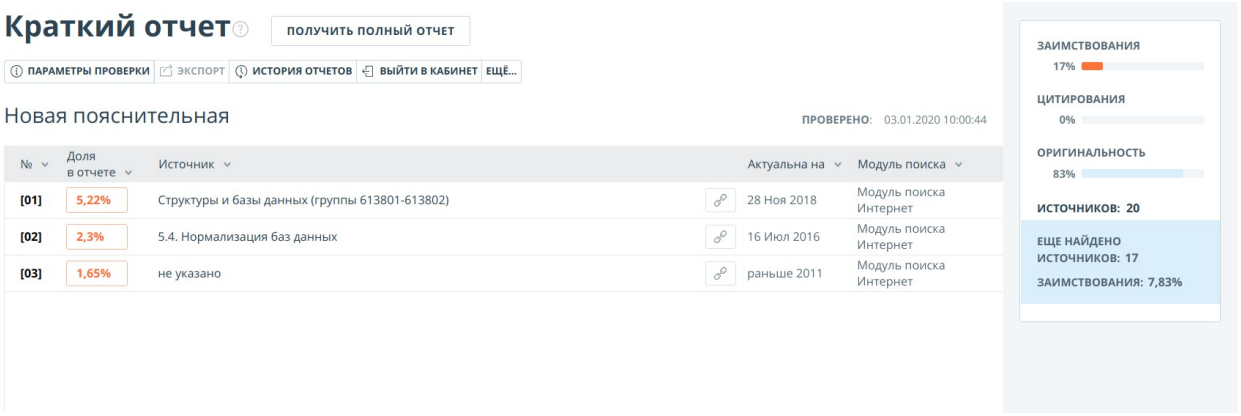
```
                CookieIdentityPolicy::new(secret_key.as_bytes())
                    .name("auth")
                    .path("/")
                    .domain("localhost")
                    .max_age_time(chrono::Duration::days(1))
                    .secure(true),
            ))
            .service(actix_files::Files::new("static", "static"))
            .route("account", web::get().to(responders::account))
            .route("admin", web::get().to(responders::admin))
            .route("board", web::get().to(responders::board))
            .route("buy", web::get().to(responders::buy))
            .route("/", web::get().to(responders::timetable))
            .route("timetable", web::get().to(responders::timetable))
            .route("logout", web::get().to(responders::logout))
            .service(
                web::resource("login")
                    .route(web::get().to(responders::login_get))
                    .route(web::post().to(responders::login)),
            )
            .service(
                web::resource("register")
                    .route(web::get().to(responders::register_get))
                    .route(web::post().to(responders::register)),
            )
            .default_service(
                web::resource("/")
                    .route(web::get().to(responders::error404))
                    .route(web::to(|| "Error 400")),
            )
    })
    .bind_rustls("0.0.0.0:8443", setup_rustls())?
    .start();

    sys.run()
}

table! {
    carriage (id) {
        id -> Unsigned<Integer>,
        voyage_id -> Unsigned<Integer>,
        carriage_type_id -> Unsigned<Integer>,
        number -> Unsigned<Tinyint>,
    }
}

table! {
    carriage_station (id) {
        id -> Unsigned<Integer>,
        station_id -> Unsigned<Integer>,
        carriage_id -> Unsigned<Integer>,
        seats_state -> Unsigned<Bigint>,
        seat_price -> Unsigned<Integer>,
```

```
    }
}

table! {
    carriage_type (id) {
        id -> Unsigned<Tinyint>,
        name -> Varchar,
        seats_count -> Unsigned<Tinyint>,
    }
}

table! {
    passenger (id) {
        id -> Unsigned<Integer>,
        passport_num -> Char,
        first_name -> Varchar,
        last_name -> Varchar,
    }
}

table! {
    station (id) {
        id -> Unsigned<Integer>,
        name -> Varchar,
    }
}

table! {
    ticket (id) {
        id -> Unsigned<Integer>,
        user_id -> Unsigned<Integer>,
        passenger_id -> Unsigned<Integer>,
        carriage_id -> Unsigned<Integer>,
        from_station_id -> Unsigned<Integer>,
        to_station_id -> Unsigned<Integer>,
        seat_num -> Unsigned<Tinyint>,
        price -> Unsigned<Integer>,
        sell_datetime -> Datetime,
        cancel_datetime -> Nullable<Datetime>,
    }
}

table! {
    train (id) {
        id -> Unsigned<Integer>,
        num -> Unsigned<Integer>,
        train_type_id -> Unsigned<Tinyint>,
    }
}

table! {
    train_station (id) {
        id -> Unsigned<Integer>,
```

```
        train_id -> Unsigned<Integer>,
        station_id -> Unsigned<Integer>,
        arrival_time_relative -> Nullable<Time>,
        departure_time_relative -> Nullable<Time>,
    }
}

table! {
    train_type (id) {
        id -> Unsigned<Tinyint>,
        name -> Varchar,
    }
}

table! {
    user (id) {
        id -> Unsigned<Integer>,
        email -> Varchar,
        pass -> Varchar,
        is_admin -> Bool,
        is_active -> Bool,
    }
}

#[derive(diesel_derive_enum::DbEnum, Debug, Clone, Copy)]
pub enum CarriageNumStart {
    Head,
    Tail,
}

table! {
    use diesel::sql_types::{Tinyint, Unsigned, Integer, Datetime, Time,
Nullable};
    use super::CarriageNumStartMapping;
    voyage (id) {
        id -> Unsigned<Integer>,
        train_id -> Unsigned<Integer>,
        departure_datetime_absolute -> Datetime,
        late_by -> Time,
        carriage_num_start -> Nullable<CarriageNumStartMapping>,
        track_num -> Nullable<Unsigned<Tinyint>>,
        platform_num -> Nullable<Unsigned<Tinyint>>,
    }
}

joinable!(carriage_station -> carriage (carriage_id));
joinable!(carriage_station -> station (station_id));
joinable!(ticket -> user (user_id));
joinable!(train_station -> station (station_id));
joinable!(train_station -> train (train_id));
joinable!(voyage -> train (train_id));

allow_tables_to_appear_in_same_query!(
```

```
    carriage,
    carriage_station,
    carriage_type,
    passenger,
    station,
    ticket,
    train,
    train_station,
    train_type,
    user,
    voyage,
);
```

ПРИЛОЖЕНИЕ В
(обязательное)
Проверка в системе «Антиплагиат»



Краткий отчет

ПОЛУЧИТЬ ПОЛНЫЙ ОТЧЕТ

ПАРАМЕТРЫ ПРОВЕРКИ | ЭКСПОРТ | ИСТОРИЯ ОТЧЕТОВ | ВЫЙТИ В КАБИНЕТ | ЕЩЁ...

Новая пояснительная

ПРОВЕРЕНО: 03.01.2020 10:00:44

№	Доля в отчете	Источник	Актуальна на	Модуль поиска
[01]	5,22%	Структуры и базы данных (группы 613801-613802)	28 Ноя 2018	Модуль поиска Интернет
[02]	2,3%	5.4. Нормализация баз данных	16 Июл 2016	Модуль поиска Интернет
[03]	1,65%	не указано	раньше 2011	Модуль поиска Интернет

ЗАИМСТВОВАНИЯ 17%

ЦИТИРОВАНИЯ 0%

ОРИГИНАЛЬНОСТЬ 83%

ИСТОЧНИКОВ: 20

ЕЩЕ НАЙДЕНО ИСТОЧНИКОВ: 17

ЗАИМСТВОВАНИЯ: 7,83%