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 технических наук, ассистент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Д. Сыс |
|  | \_\_\_.\_\_\_\_.2021 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«БАЗА ДАННЫХ ДЛЯ ПОДДЕРЖКИ РАБОТЫ автомагазина»**

БГУИР КП 1-39 03 02 004 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 913801  Гончар Виталий Витальевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2021  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2021

**РЕФЕРАТ**

БГУИР КП 1-39 03 02 004 ПЗ

**Гончар, В.В.** База данных для поддержки работы автомагазина: пояснительная записка к курсовому проекту / В.В. Гончар. – Минск: БГУИР, 2021. – 52 с.

Пояснительная записка 52 с., 15 рисунков, 16 источников, 4 приложения.

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЁ ФОРМАЛИЗАЦИЯ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ, ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ДЛЯ ОСНОВНОГО ВИДА ДЕЯТЕЛЬНОСТИ РАССМАТРИВАЕМОЙ ОБЛАСТИ, ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ БАЗЫ ДАННЫХ

*Цель* *проектирования*: разработка веб-приложения, проектирование эффективной и безопасной базы данных для поддержания работы автомагазина.

*Методология проведения работы*: в процессе решения поставленных задач использованы теория проектирования базы данных, методы проектирования веб-приложений.

*Результаты работы*: изучены способы хранения информации в базе данных, проведен анализ реляционной модели данных, разработано веб-приложение для работы автомагазина.

*Область применения результатов*: разработанная база данных и приложение в будущем, возможно применить для создания полноценного коммерческого приложения, которое может быть использовано реальным автомагазином.

СОДЕРЖАНИЕ

[Введение 7](#_Toc90225439)

[1 Анализ предметной области и ее формализация для проектирования базы данных 8](#_Toc90225440)

[1.1 Описание предметной области 8](#_Toc90225441)

[1.2 Анализ информационных потребностей пользователей и предварительное описание запросов 9](#_Toc90225442)

[1.3 Определение требований и ограничений к базе данных с точки зрения предметной области 10](#_Toc90225443)

[1.4 Постановка решаемой задачи 11](#_Toc90225444)

[2 Проектирование базы данных для основного вида деятельности рассматриваемой предметной области 12](#_Toc90225445)

[2.1 Разработка инфологической модели предметной области базы данных 12](#_Toc90225446)

[2.2 Выбор и обоснование используемых типов данных и ограничений (доменов) 14](#_Toc90225447)

[2.3 Проектирование запросов к базе данных 17](#_Toc90225448)

[2.4 Программная реализация и документирование базы данных 20](#_Toc90225449)

[3 Применение разработанной базы данных 23](#_Toc90225450)

[3.1 Руководство пользователя 23](#_Toc90225451)

[3.2 Администрирование базы данных 26](#_Toc90225452)

[3.3 Реализация клиентских запросов 27](#_Toc90225453)

[3.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных 29](#_Toc90225454)

[Заключение 31](#_Toc90225455)

[Список используемых источников 32](#_Toc90225456)

[Приложение А (обязательное) Проверка на оригинальность 34](#_Toc90225457)

[Приложение Б (обязательное) Скрипт генерации БД 35](#_Toc90225460)

[Приложение В (обязательное) Листинг программного кода 48](#_Toc90225463)

[Приложение Г (обязательное) Ведомость документов 51](#_Toc90225466)

# ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

|  |  |  |
| --- | --- | --- |
| БД | – | База данных |
| *VIN* | – | Идентификационный номер транспортного средства |
| СУБД | – | Система управления базами данных |
| *PK* | *–* | *Primary Key* |
| *API* | *–* | *Application Programming Interface* |
| *ER* | *–* | *Entity-Relationship* |
| *CASE* | – | Набор инструментов для проектирования программного обеспечения |
| *ID* | *–* | *Identifier* |
| НФ | – | Нормальная форма |
| *DOM* | *–* | *Document Object Model* |
| *HTTP* | – | Протокол передачи гипертекста |
| *JS* | *–* | *JavaScript* |
| *VS* | *–* | *Visual Studio* |
| *GIT* | *–* | Распределённая система управления версиями |
| *SSL* | *–* | Криптографический протокол |
|  |  |  |
|  |  |  |

# ВВЕДЕНИЕ

На сегодняшний день жизнь каждого человека сложно представить без автомобиля. Конечно, можно поспорить, сказав, что в крупных городах (где население близко к одному миллиону) существует развитая система общественного транспорта, такого как метро, автобусы, трамваи, и с этим сложно не согласиться. Однако, личный автомобиль предоставляет своему водителю такие преимущества, которые ни один общественный транспорт предоставить не может. В такие преимущества входят:

– комфорт поездки;

– свобода передвижения.

В связи с этим порождается высокий спрос на покупку личных автомобилей. Данная сфера остаётся актуальной и по сей день и поэтому важно, чтобы покупка автомобиля для будущего автовладельца проходила быстро и приятно. Для этих целей должна быть разработана как база данных для учета автомобиля и поддержки работы автомагазина в целом, так и веб-приложение для работы с базой данных.

При выполнении курсового проекта должны быть учтены особенности, с которыми должно производиться проектирование базы данных, а именно: проектируемая база данных должна отвечать требованиям надёжности, минимальной избыточности, целостности данных, содержать не менее десяти сущностей, а ее схема должна быть приведена к третьей нормальной форме. База данных должна поддерживать основные современные средства для работы и администрирования. Необходимо реализовать следующие объекты базы данных в количестве не менее трех каждый: индекс, триггер, хранимая процедура. Также требуется реализовать систему разграничения прав доступа к данным минимум для двух ролей пользователей.

Таким образом, целью данного курсового проекта является проектирование базы данных для поддержки работы автомагазина, предметной областью которой является реализация автомобилей, а также разработка программного средства.

# 1 **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЕ ФОРМАЛИЗАЦИЯ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ**

Проектирование базы данных – это набор процессов, которые облегчают проектирование, разработку, внедрение и обслуживание систем управления корпоративными данными. Правильно спроектированная база данных проста в обслуживании, улучшает согласованность данных и экономична с точки зрения дискового пространства для хранения. Разработчик базы данных решает, как элементы данных соотносятся и какие данные должны быть сохранены [1].

Основными задачами проектирования баз данных в СУБД являются создание логических и физических моделей дизайна предлагаемой системы баз данных. Логическая модель концентрируется на требованиях к данным и данных, которые должны храниться независимо от физических соображений. Его не волнует, как данные будут храниться или где они будут храниться физически. Модель проектирования физических данных включает в себя перевод логического проекта базы данных на физический носитель с использованием аппаратных ресурсов и программных систем, таких как системы управления базами данных.

# 1.1 Описание предметной области

Из определения следует, что автомагазин – это бизнес, который продает новые или подержанные автомобили на уровне розничной торговли на основании дилерского договора с автопроизводителем или его дочерней компанией по сбыту. Автомагазин, как правило, имеет в своём штате несколько десятков сотрудников, которые разделены на отделы, а именно: отдел продаж, отдел маркетинга, отдел покупок и так далее. Автомагазин также может предоставлять услуги по техническому обслуживанию автомобилей и нанимать автомобильных специалистов (механиков) для хранения и продажи запасных частей для автомобилей и обработки претензий по гарантии.

Тема продаж и покупок автомобилей очень популярна в наше время, так как сам предмет – автомобиль пользуется большой популярностью у населения. Для облегчения поиска нового автомобиля во многих городах организуются специальные магазины, называемые автомагазином (автосалоном), в которых клиенты могут подобрать себе автомобиль по вкусу. Как известно, новые автомобили поставляются в разных конфигурациях. В таком случае, автомобиль стандартной комплектации отличается от автомобиля максимальной комплектации присутствием той или иной функции или прибора. Поэтому, доукомплектование автомобиля по желанию покупателя будет осуществляться путём выбора наличия той или иной детали автомобиля.

Формализованная организация учета реализации автомобилей в автомагазине предполагает следующие бизнес-процессы:

– поступление заказа на какой-либо автомобиль со списком необходимых к установке деталей;

– заказ автомобиля автомагазином у компании-поставщика;

– покупка деталей у компаний-поставщиков, которые будут устанавливаться в автомобиль согласно заказу;

– прием автомобиля, то есть принятие автомобиля на внутренний учет, проведение подготовки, переоборудования и диагностики автомобиля, оповещение покупателя;

– реализация автомобиля, то есть осмотр автомобиля покупателем, оформление договора купли-продажи.

Таким образом был описан построен процесс работы автомагазина, однако стоит понимать, что это – упрощённый вариант.

# 1.2 Анализ информационных потребностей пользователей и предварительное описание запросов

Приложение разрабатывается для нескольких групп пользователей, у каждой из которых свой уровень доступа. Пользователь «Администратор» может просматривать все данные (таблицы), которые есть в базе данных, а также изменять, добавлять и удалять поля во всех таблицах. Единственное, чего не может администратор – это добавление новых пользователей. Для этих целей служит окно регистрации. Таким образом достигается конфиденциальность данных для каждого пользователя, который хранится в базе данных.

Стандартный пользователь, которым в большинстве случаев будет являться посетитель, имеет ограниченный круг возможностей по взаимодействию с базой данных. Что касается создания и удаления записей, то пользователь может делать вышеперечисленные действия только по отношению к таблице «Заказы». С остальными таблицами пользователь не имеет никаких прямых связей. Просмотр списка товаров (автомобилей, в данном случае) происходит через операцию просмотра представления, которое представительно настроено в базе данных.

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

В соответствии с предметной областью система строится с учётом

следующих особенностей:

– в автомагазине может быть любое количество автомобилей;

– каждый автомобиль имеет уникальный 17-значный номер, который называется *VIN*;

– каждый автомобиль имеет свою характеристику;

– стоимость автомобиля и стоимость деталей – положительные числа, с не более чем двумя значениями после запятой;

– покупатель может выбрать несколько автомобилей, но на каждый автомобиль оформляется отдельный заказ.

– при создании заказа автомобиль и работник магазина, указанный в заказе, приобретают статус «занят»;

– при подтверждении заказа автомобиль приобретает статус «продан» и больше не показывается в списке, предоставляемом потенциальным покупателям, однако всё еще числится в базе, а работник становится свободным для следующих заказов;

– при удалении заказа автомобиль и работник, указанные в заказе, приобретают статус «свободный».

В соответствии с вышеперечисленными ограничениями будут разрабатываться как база данных, так и программное средство для работы с базой данных.

Качество проектирования базы данных влияет на работу с этой базой данных в будущем. С хорошо спроектированной базой данных легче работать, легче писать запросы.

Таким образом, были сформированы следующие таблицы:

– таблица «*userrrs*»;

– таблица «*orders*»;

– таблица «*employees*»;

– таблица «*qualifications*»;

– таблица «*payment\_types*»;

– таблица «*order\_types*»;

– таблица «*vehicle\_models*»;

– таблица «*manufacturers*»;

– таблица «*vehicle\_types*»;

– таблица «*engines*»;

– таблица «*turbos*»;

– таблица «*transmissions*»;

– таблица «*wheels*»;

Данного набора сущностей вполне хватит для нормального функционирования автомагазина и взаимодействия пользователей с автомагазином.

# 1.4 Постановка решаемой задачи

Таким образом, в качестве входных данных в данном курсовом проектировании имеются: необходимость в проектировании базы данных, которая будет проста в обслуживании, будет иметь согласованные данные и экономична с точки зрения дискового пространства для хранения. Вместе с этим, база данных должна быть надёжной, обеспечивать целостность данных, содержать не менее 10 сущностей, а также должна быть приведена к третьей нормальной форме. Чтобы обеспечить правильное взаимодействие пользователя с ассортиментом автомагазина, в базе данных будут реализованы так называемые триггеры: например, когда пользователь закажет автомобиль, ему автоматически присвоится статус «занят», поэтому другой пользователь не сможет заказать его до тех пор, пока первый пользователь по какой-либо причине не отменит заказ. Целостность и безопасность данных в базе данных будут реализованы через систему прав доступа, разграничения ролей, а также через использование представлений.

# 2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ДЛЯ ОСНОВНОГО ВИДА ДЕЯТЕЛЬНОСТИ РАССМАТРИВАЕМОЙ ПРЕДМЕТНОЙ ОБЛАСТИ

Правильно спроектированная база данных предоставляет доступ к актуальной и точной информации. Поскольку правильное проектирование важно для достижения целей автомагазина при работе с базой данных, имеет смысл вкладывать время, необходимое для изучения принципов хорошего проектирования.

# 2.1 Разработка инфологической модели предметной области базы данных

Инфологическая модель применяется на втором этапе проектирования базы данных, когда уже сделано словесное описание предметной области, то есть после этапа постановки задачи.

Проблема представления семантики давно интересовала разработчиков, и в семидесятых годах было предложено несколько моделей данных, названных семантическими моделями. К ним можно отнести функциональную модель данных Шипмана, семантическую модель данных, предложенную Хаммером и Мак-Леоном в 1981 году, модель «сущность-связь», предложенную Ченом в 1976 году и ряд других моделей. У всех моделей были свои положительные и отрицательные стороны, но испытание временем выдержала только последняя. И в настоящий момент именно модель Чена «сущность – связь», или «*Entity Relationship*», стала фактическим стандартом при инфологическом моделировании баз данных. Общепринятым стало сокращенное название *ER*-модель, большинство современных *CASE*-средств содержат инструментальные средства для описания данных в формализме этой модели. Кроме того, разработаны методы автоматического преобразования проекта БД из *ER*-модели в реляционную, при этом преобразование выполняется в даталогическую модель, соответствующую конкретной СУБД. Все *CASE*-системы имеют развитые средства документирования процесса разработки БД, автоматические генераторы отчетов позволяют подготовить отчет о текущем состоянии проекта БД с подробным описанием объектов БД и их отношений как в графическом виде, так и в виде готовых стандартных печатных отчетов, что существенно облегчает ведение проекта.

Сущность – это реальный или воображаемый объект, информация о котором представляет интерес. На схемах модели *ER* сущность представлена ​​в виде прямоугольника (рисунок 1), содержащего имя сущности. В этом случае имя сущности – это имя типа, а не конкретного объекта – экземпляра этого типа. Каждый экземпляр объекта должен отличаться от любого другого экземпляра того же объекта.

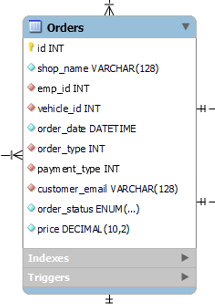


Рисунок 1 – *ER*-сущность «Заказы»

Отношения – это графическая ассоциация, установленная между двумя объектами. Эта ассоциация всегда является двоичной и может существовать между двумя разными объектами или между объектом и самим собой (рекурсивная связь). В любом отношении различаются два конца (в соответствии с парой связанных сущностей), каждый из которых указывает имя конца отношения, степень конца отношения (сколько экземпляров данной сущности связаны), должен ли какой-либо экземпляр этой сущности участвовать в этом соединении).

Инфологическая модель проектируемой базы данных приведена в графических материалах к курсовому проекту.

Стоит также отметить, что важным условием к готовой базе данных является то, что в конце разработки должна получиться база данных в третьей нормальной форме [3].Метод нормальных форм (НФ) состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений.

Отношение находится в первой НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Отношение находится во второй НФ, если оно находится в первой НФ и каждый не ключевой атрибут неприводимо зависит от *PK*. Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Отношение находится в третьей НФ, когда находится во второй НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

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

В *MySQL* три основных типа данных – строчный (*string*), числовой (*numeric*) и тип даты и времени (*date and time*). Каждый из вышеперечисленных типов имеет как минимум ещё 10 подтипов [2]. Естественно, имеет смысл использовать только те типы данных, которые действительно необходимы.

Строчные типы данных, которые были использованы в данном курсовом проекте:

1 *VARCHAR* (*size*) – это строка переменной длины, которая может содержать буквы, числа и специальные символы. Параметр *size* указывает максимальную длину строки – может быть от 0 до 65535. Данный тип используется для хранения строковых значений, например – имён и фамилий сотрудников, названий моделей автомобилей.

2 *TEXT* (*size*) может содержать строку максимальной длиной 65 535 байт. Данный тип используется для хранения описаний автомобилей.

3 *ENUM* (*val1, val2, val3*, ...) – это строковый объект, который может иметь только одно значение, выбранное из списка возможных значений. Можно перечислить до 65535 значений. Если вставлено значение, которого нет в списке, будет вставлено пустое значение. Значения отсортированы в том порядке, в котором они введены. Используется для хранения ролей пользователей, а также для хранения занятости работников и занятости автомобилей.

Числовые типы данных, которые были использованы в данном курсовом проекте:

1 *INT* (*size*) – это целое число. Диапазон включает числа от -2147483648 до 2147483647. Диапазон без знака – от 0 до 4294967295. Используется, в основном, для хранения идентификаторов в сущностях.

2 *DECIMAL* (*size*; *d*) – точное число с фиксированной точкой. Общее количество цифр указано в параметре *size*. Количество цифр после десятичной точки указывается в параметре d. Максимальное число *size* – 65. Максимальное число для *d* – 30. Значение по умолчанию для *size* – 10. Значение по умолчанию для d – 0. Используется для хранения стоимостей автомобилей и деталей, а также для значений коэффициентов у типов заказа.

Что касается типов даты и времени, был выбран только один. *DATETIME* – комбинация даты и времени. Формат: ГГГГ-ММ-ДД чч: мм: сс. Поддерживаемый диапазон: от «1000-01-01 00:00:00» до «9999-12-31 23:59:59». Можно добавлять параметры *DEFAULT* и *ON UPDATE* в определение типа для автоматической инициализации и обновления до текущей даты и времени.

Выше были перечислены типы данных, которые были использованы при создании сущностей для базы данных. В проектируемой базе данных использовались лишь необходимые типы, чтобы сделать понятные и обоснованные ограничения полей. Также можно заметить, что в некоторых сущностях есть поля, который имеют связь с фото. Для этих случаев было решено использовать тип *VARCHAR* и хранить в этих полях путь до директории, в которой хранится необходимое фото.

Что касается ограничений (*CONSTRAINTS*), то они используются для того, чтобы ограничивать значения в таблице тем или иным способом. Это нужно, чтобы обеспечить целостность базы данных. Ограничения могут быть настроены на уровне столбца и на уровне таблицы в целом. Всего таких ограничений 6:

1 *NOT NULL*. В *MySQL* ограничение *NOT NULL* позволяет указать, что столбец не может содержать никаких значений *NULL*(то есть ничего не содержать).

2 *UNIQUE*. Ограничение *UNIQUE* в *MySQL* не позволяет вставлять повторяющееся значение в столбец. Ограничение *UNIQUE* поддерживает уникальность столбца в таблице. В таблице можно использовать более одного столбца *UNIQUE*.

3 *PRIMARY KEY*. Ограничение *PRIMARY KEY* создает уникальный индекс (первичный ключ) для более быстрого доступа к таблице.

4 *FOREIGN KEY*. *FOREIGN KEY* в *MySQL* создает связь между двумя таблицами по одному определенному столбцу обеих таблиц. Указанный столбец в одной таблице должен быть обозначен как *PRIMARY KEY* и на него должен ссылаться столбец другой таблицы – *FOREIGN KEY*.

5 *CHECK* Ограничение *CHECK* используется для ограничения диапазона значений в столбце.

6 *DEFAULT*. Это ограничение означает то, что в таблице *MySQL* каждый столбец должен содержать какое-либо значение (включая *NULL*). При вставке данных в таблицу, если в столбец не задано значение, столбец получает значение, установленное как *DEFAULT*.

В реляционных базах данных практически всегда разные таблицы логически связаны друг с другом. Первичные ключи используются для организации такой связи.

Первичный ключ – это специальный столбец таблицы реляционной базы данных (или комбинация столбцов), предназначенный для уникальной идентификации каждой записи таблицы [4]. Первичный ключ используется как уникальный идентификатор для быстрого анализа данных в таблице. Таблица не может иметь более одного первичного ключа.

Основные характеристики первичного ключа:

– первичный ключ должен содержать уникальное значение для каждой строки данных;

– первичный ключ не может содержать нулевые значения (*null*);

– каждая строка должна иметь значение первичного ключа;

– первичный ключ может использовать одно или несколько полей, уже присутствующих в базовой модели данных, или может быть создано конкретное дополнительное поле в качестве первичного ключа.

Использование первичных ключей также является важным шагом в приведении базы данных к третьей нормальной форме. Ведь для этого нужно, чтобы база данных уже была приведена ко второй нормальной форме, а перед этим к первой нормальной форме, одним из условий которой является наличие первичных ключей.

Внешний ключ – это столбец или группа столбцов в таблице реляционной базы данных, которая обеспечивает связь между данными в двух таблицах. Он действует как перекрестная ссылка между таблицами, поскольку ссылается на первичный ключ другой таблицы, тем самым устанавливая связь между ними. Большинство таблиц в системе реляционных баз данных придерживаются концепции внешнего ключа. В сложных базах данных и хранилищах данных данные в домене должны добавляться в несколько таблиц, таким образом поддерживая связь между ними. Данный принцип работы с данными позволяет сохранить целостность данных, а также минимизирует избыточность.

После настройки первичных и внешних ключей таблицы будут так или иначе связаны, то есть будут находиться в отношениях. Есть определенные правила, которые определяют отношения между таблицами:

– отношение 1:1, когда первичный ключ для одной из таблиц включен в качестве внешнего ключа в другой таблице;

– отношение 1:*n*, когда первичный ключ из одной таблицы добавляется в качестве внешнего ключа в таблицу n;

– отношение *m:n*, при котором создается новая таблица (таблица связи), где первичный ключ состоит из первичных ключей двух оригинальных таблиц.

# 2.3 Проектирование запросов к базе данных

Взаимодействие между базой данных и пользователем будет осуществляться с помощью клиента (разрабатываемого программного средства). Клиент, в свою очередь, обращается к серверу с базой данных с помощью специальных запросов.

Например, та или иная информация о любой сущности может быть получена с помощью *SQL* запросов [5].

Общая структура запроса чтения выглядит следующим образом:

– *SELECT* ('столбцы или \* для выбора всех столбцов; обязательно');

– *FROM* ('таблица; обязательно');

– *WHERE* ('условие/фильтрация; необязательно');

– *GROUP BY* ('столбец, по которому нужно сгруппировать данные; необязательно');

– *HAVING* ('условие/фильтрация на уровне сгруппированных данных; необязательно');

– *ORDER BY* ('столбец, по которому нужно отсортировать вывод; необязательно').

Описания элементов запроса:

1 *SELECT*, *FROM* – обязательные элементы запроса, которые определяют выбранные столбцы, их порядок и источник данных.

2 *WHERE* – необязательный элемент запроса, который используется, когда нужно отфильтровать данные по нужному условию. Очень часто внутри данного элемента используются *IN* / *NOT IN* для фильтрации столбца по нескольким значениям, *AND* / *OR* для фильтрации таблицы по нескольким столбцам.

3 *GROUP BY* – необязательный элемент запроса, с помощью которого можно задать агрегацию по нужному столбцу (например, если нужно узнать какое количество клиентов живет в каждом из городов). При использовании *GROUP BY* обязательно, чтобы перечень столбцов, по которым делается разрез, был одинаковым внутри *SELECT* и внутри *GROUP BY*, агрегатные функции (*SUM*, *AVG*, *COUNT*, *MAX*, *MIN*) должны быть также указаны внутри *SELECT* с указанием столбца, к которому такая функция применяется.

4 *HAVING* – необязательный элемент запроса, который отвечает за фильтрацию на уровне сгруппированных данных (по сути, *WHERE*, но только на уровень выше).

5 *ORDER BY* – необязательный элемент запроса, который отвечает за сортировку таблицы. Осуществлять сортировку можно и по нескольким столбцам, в этом случае сортировка происходит по порядку указанных столбцов. По умолчанию сортировка происходит по возрастанию для чисел и в алфавитном порядке для текстовых значений. Если нужна обратная сортировка, то в конструкции *ORDER BY* после названия столбца надо добавить *DESC*.

*SELECT* относится к стандартным командам, которые позволяют работать непосредственно с данными. К таким командам также относятся команды *CREATE*, *UPDATE* и *DELETE*, которые используются, соответственно, для создания, обновления и удаления таблиц.

Что касается группы пользователей «администратор», то данная группа выполняет запросы непосредственно к таблицам базы данных. Например, запрос выбора всех полей из таблицы с двигателями автомобилей и результат такого запроса представлены на рисунке 2.



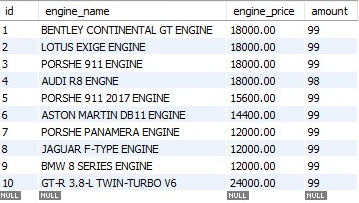
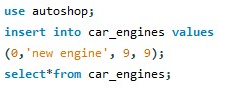


Рисунок 2 – Пример запроса *SELECT*

Подобным образом происходит выбор всех полей для всех таблиц. Администратор должен быть способен также добавлять новые поля, удалять либо изменять уже существующие.

Добавление записи в таблицу с, например, двигателями к автомобилям, осуществляется через оператор *INSERT* и представлено на рисунке 3. Данный оператор используется, например, при регистрации нового пользователя.



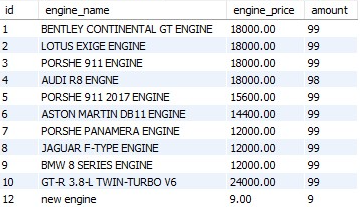
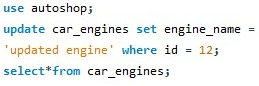


Рисунок 3 – Пример добавления поля

Изменение записей в таблице происходит с помощью оператора *UPDATE* (рисунок 4). При этом важно указать поле, к которому должны быть применены изменения, иначе изменениям подвергнутся все поля в таблице.



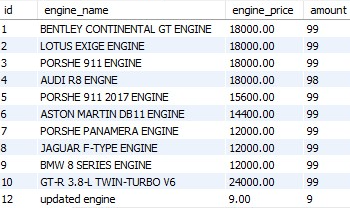
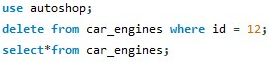


Рисунок 4 – Пример изменения поля

Данный запрос используется, например, для управления ролями администратором, а также для изменения конфигурации автомобиля после создания заказа пользователем.

Запрос на удаление поля можно осуществить с помощью оператора *DELETE* (рисунок 5). Как и в случае с *UPDATE*, лучше использовать условие, по которому удалится то или иное поле. Если условие не указать, то все поля удалятся, то есть таблица станет пустой.



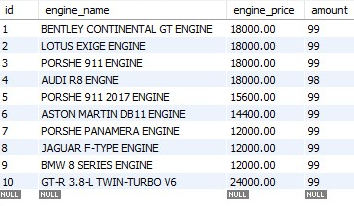


Рисунок 5 – Удаление поля из таблицы

Данный запрос используется, например, пользователем, когда необходимо удалить заказ.

Таким образом была создана связь между сервером с базой данных и веб-приложением, которое разрабатывается в данном курсовом проекте.

# 2.4 Программная реализация и документирование базы данных

Программное средство для поддержки работы автомагазина разрабатывалось целиком при помощи языка программирования *JavaScript*. *JavaScript* – это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц (например, со сложной анимацией, интерактивными кнопками, всплывающими меню и так далее) [6]. Существуют также более продвинутые серверные версии *JavaScript*, такие как *Node.js*, которые позволяют добавлять на веб-сайт больше функций, чем просто загрузка файлов (например, совместная работа между несколькими компьютерами в реальном времени). Внутри веб-браузера *JavaScript* может быть связан с объектами своей среды, чтобы обеспечить программный контроль над ними.

*JavaScript* содержит стандартную библиотеку объектов, таких как *Array*, *Date* и *Math*, а также основной набор языковых элементов, таких как операторы, управляющие структуры и инструкции. Базовый *JavaScript* можно расширить для различных целей, добавив в него дополнительные объекты: клиентский и серверный.

Клиентский *JavaScript* расширяет базовый язык, предоставляя объекты для управления браузером и его объектной моделью документа (*DOM*). Например, клиентские расширения позволяют приложению размещать элементы в *HTML*-форме и реагировать на пользовательские события, такие как щелчки мыши, ввод формы и навигация по страницам.

Серверный *JavaScript* расширяет базовый язык, предоставляя объекты, относящиеся к запуску *JavaScript* на сервере. Например, серверные расширения позволяют приложению обмениваться данными с базой данных, обеспечивать непрерывность потока информации от одной части приложения до другой или выполнять манипуляции с файлами на сервере.

Это означает, что в браузере *JavaScript* может изменить внешний вид веб-страницы (*DOM*). Точно так же *Node.js* *JavaScript* на сервере может отвечать на пользовательские запросы из кода, написанного в браузере.

В качестве среды разработки использовался *Visual Studio Code* – редактор исходного кода, созданный *Microsoft* [7]. *Visual Studio Code* включает встроенную поддержку автозавершения кода *IntelliSense*, глубокого понимания семантического кода и навигации, а также рефакторинга кода. Отладка часто является той функцией, которую разработчики больше всего упускают из виду при более компактном кодировании, поэтому *Visual Studio Code* включает интерактивный отладчик, поэтому можно выполнять пошаговое выполнение исходного кода, проверять переменные, просматривать стеки вызовов и выполнять команды в консоли. *VS Code* также интегрируется со средствами сборки и создания сценариев для выполнения общих задач, ускоряя повседневные рабочие процессы. *VS Code* поддерживает *Git*, поэтому можно работать с системой управления версиями, не выходя из редактора, включая просмотр изменений в ожидающих изменениях.

Что касается документации базы данных, есть несколько подходов [8]. Конечно, в контексте данного курсового проекта, в качестве средства документации можно использовать *ER*-диаграмму, однако в общей практике – это не самый удачный способ, потому что такая диаграмма может быть очень запутанной и мало кто сможет из неё что-либо узнать. Таким образом, хорошая и удобная к чтению человеком документация будет включать в себя для каждой таблицы:

– описание значения таблицы и ее функционального использования (в пользовательском интерфейсе и т. д.);

– описание того, что означает каждый атрибут, если это не очевидно;

– объяснение отношений (внешних ключей) от этой таблицы к другим, и наоборот;

– объяснения дополнительных ограничений и / или триггеров;

– дополнительное объяснение основных представлений и процессов, которые касаются таблицы, если они еще не хорошо документированы.

Учитывая всё вышеперечисленное следует сказать, что не нужно делать документацию ради документации, потому что повторение очевидного будет только сбивать с правильных мыслей. Вместо этого нужно сосредоточиться на том, что может быть неочевидно, и потратить несколько минут на написание действительно четких и кратких объяснений.

Документацию можно проводить как вручную, так и с помощью сторонних программных средств. Инструменты документации баз данных действуют как каталог данных, который фиксирует всю информацию базы данных в упрощенном табличном формате. Чаще всего он используется в качестве словаря данных, который предоставляет подробные сведения о схеме таблицы, или в качестве единственного источника достоверной информации, который обеспечивает обзор всех взаимосвязей внутри базы данных.

# 3 ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ БАЗЫ ДАННЫХ

# 3.1 Руководство пользователя

Руководство пользователя – это технический документ, предназначенный для ознакомления пользователей с определенным продуктом. Выпущенные в печатном или цифровом виде, они могут быть написаны практически о любом продукте, но наиболее распространены в компьютерном программном и аппаратном обеспечении, электронных устройствах и видеоиграх [9].

Взаимодействие пользователя и клиента начинается с окна входа, в котором есть форма как форма для входа, так и форма для регистрации. Форма для входа представлена на рисунке 6.

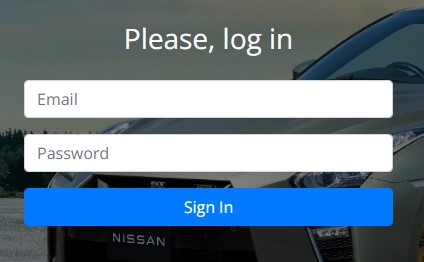


Рисунок 6 – Форма входа в учетную запись

При правильном вводе данных произойдёт перенаправление на домашнюю страницу. Если же ввод почты или пароля произведён неверно, то страница входа просто перезагрузится.

При отсутствии аккаунта предусмотрена форма регистрации, которая представлена на рисунке 7. При успешной регистрации будет выведено информирующее сообщение об успешности операции, после которого будет возможен вход.

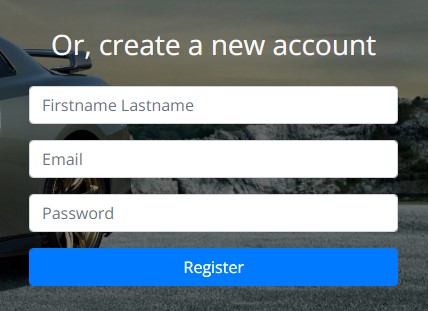


Рисунок 7 – Форма регистрации

Пустыми поля оставить не получится, будет выведено предупреждение о необходимости ввести данные рядом с пустым полем (рисунок 8).

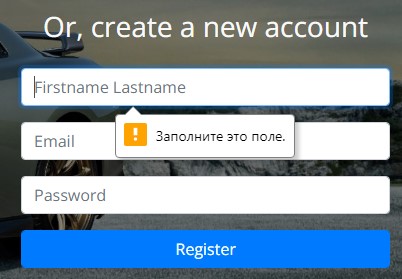


Рисунок 8 – Предупреждение о пустом поле

После успешного входа в свою учетную запись, пользователь будет перенаправлен на страницу, где расположен список доступных автомобилей (рисунок 9).

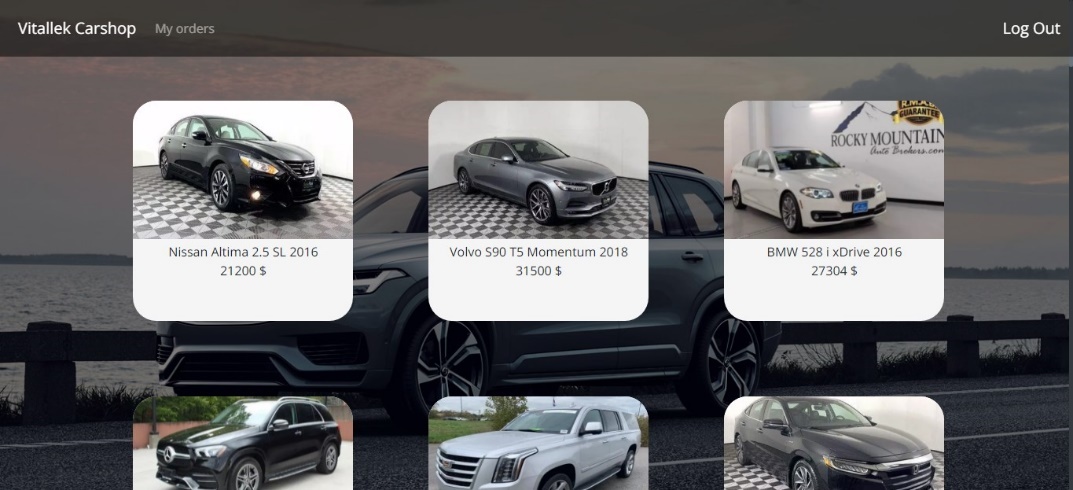


Рисунок 9 – Страница со списком автомобилей

На данной странице пользователь может просматривать автомобили. Также, как можно заметить на верхней навигационной панели, пользователь может просмотреть список своих заказов, а также выйти из своей учетной записи. Чтобы вернуться к списку автомобилей, необходимо нажать на первый элемент панели, который содержит логотип автомагазина.

Чтобы узнать подробную информацию об автомобиле, необходимо нажать на блок, который содержит нужный автомобиль. В данном блоке можно указать детали, которые пользователь хочет поставить, тип заказа и тип оплаты (рисунок 10). После этого заказ может быть создан.

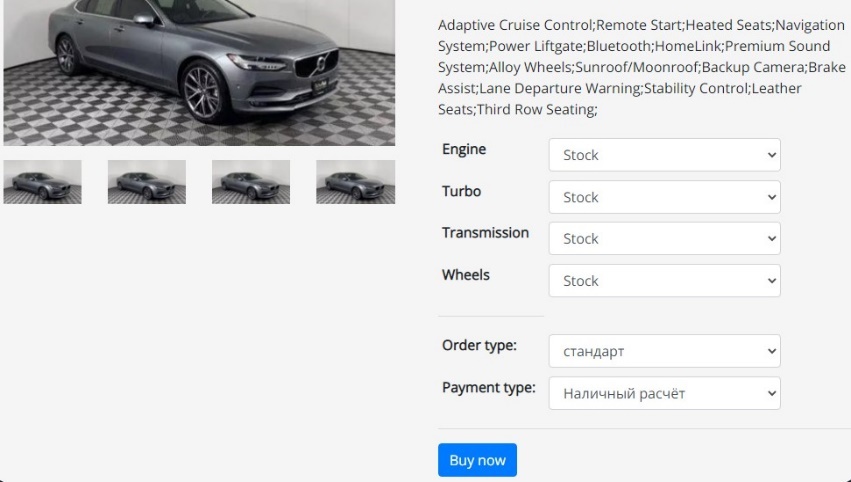


Рисунок 10 – Подробное описание автомобиля

При успешном создании заказа пользователь будет проинформирован об этом через уведомление сверху экрана. Созданный заказ, как уже упоминалось ранее, можно посмотреть путём нажатия на «*My orders*» в верхней панели навигации (Рисунок 11).

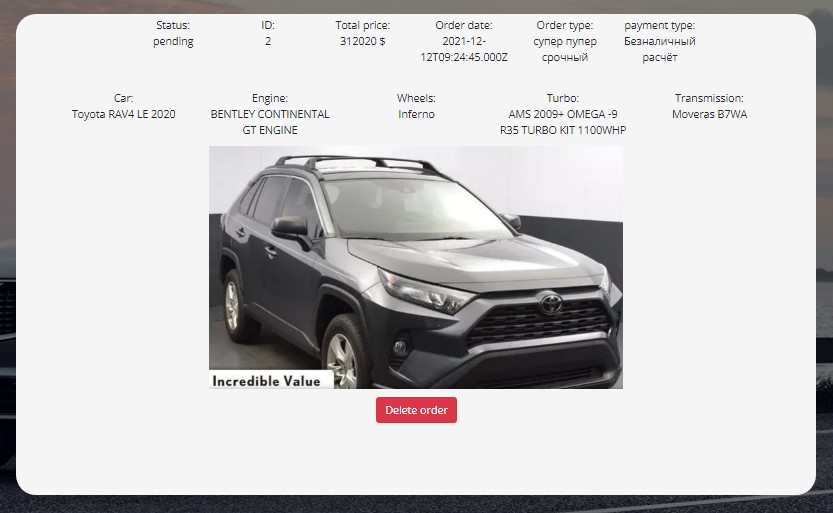


Рисунок 11 – Один заказ из списка

После создания заказа, заказ присваивается случайному свободному работнику для выполнения. А когда заказ выполнен, то есть, когда автомобиль готов, то пользователь, который заказал данный автомобиль, может приехать в автомагазин и забрать его. В случае успешной покупки, автомобилю присваивается статус «продан». Таким образом работает программа, которая позволяет обычному пользователю взаимодействовать с ассортиментом автомагазина.

# 3.2 Администрирование базы данных

Администрирование базы данных – набор действий, который должен выполнять администратор базы данных, чтобы база данных всегда была доступа, обеспечивала безопасность данных. Также администратор должен проводить мониторинг и устранение неполадок базы данных, возможности её безопасного масштабирования, по необходимости [10].

Администратор баз данных обычно также отвечает за другие второстепенные, но все же критически важные задачи и роли. Некоторые из них включают в себя:

1 Безопасность базы данных – обеспечение того, чтобы только авторизованные пользователи имели доступ к базе данных, и защита ее от любого внешнего несанкционированного доступа.

2 Настройка базы данных или настройка любого из нескольких параметров для оптимизации производительности, таких как выделение памяти сервера, фрагментация файлов и использование диска.

3 Резервное копирование и восстановление – здесь роль администратора базы данных состоит в том, чтобы обеспечить наличие в базе данных соответствующих процедур резервного копирования и восстановления для восстановления после любой случайной или преднамеренной потери данных.

4 Создание отчетов из запросов – администраторов баз данных часто вызывают для создания отчетов путем написания запросов, которые затем запускаются в базе данных.

Что касается уровней доступа к базе данных, то было создано три группы пользователей – администратор, работник и пользователь [11], а для каждой группы была создана своя роль на основе прав доступа, которые должна иметь та или иная группа по отношению к любой из таблице в базе данных (рисунок 12).

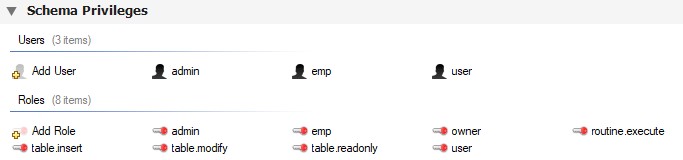


Рисунок 12 – Пользователи и роли

Таким образом реализована система прав доступа в базе данных для поддержки работы автомагазина. Так, администратор владеет самым широким спектром прав и способен изменять любые поля во всех таблицах. Работник автомагазина (группа *emp*) не имеет доступа к таблице с другими работниками и к таблице с пользователями.

Обычный пользователь имеет самый узкий спектр прав. Данный пользователь может только просматривать представления, которые заранее подготовлены для просмотра. Также в возможности данной группы входит создание и удаление заказов.

# 3.3 Реализация клиентских запросов

В процессе работы с программным средством пользователь будет так или иначе обращаться к базе данных с помощью запросов.

Например, когда пользователь открывает список автомобилей, для него выполняется запрос, который представлен на рисунке 13.

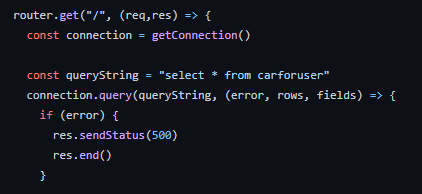


Рисунок 13 – Реализация запроса *SELECT*

В данной функции происходит выполнение запроса, который покажет все данные, которые есть в таблице «*carForUser*», которая в свою очередь является представлением. В случае успешного выполнения запроса данные с таблицы будут переданы в *HTTP*-клиент *Axios*, после которого данные будут отображены на странице.

Аналогично реализован показ запросов для пользователя. Единственным отличием является то, что дополнительно проводится проверка на соответствие адреса электронной почты пользователя, который запрос выполняет и адреса почты, который указан в заказе.

Пользователь также может создавать заказы. Функция создания заказа показана на рисунке 14.

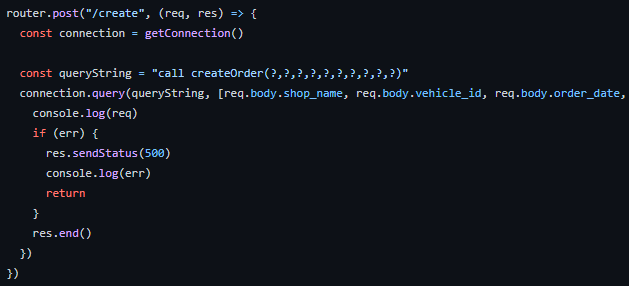


Рисунок 14 – Реализация функции создания заказа

В данном случае для создания заказа используется вызов процедуры, в которую передаются определённые параметры, а именно значения, которые нужно вставить в поле заказа.

# 3.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных

Безопасность базы данных предполагает защиту данных от преднамеренного и непреднамеренного доступа, модификации или разрушения. В основном, применяется запрещение несанкционированного доступа (разграничение прав доступа), защита от копирования и криптографическая защита [12].

Существует несколько рекомендаций по обеспечению высокого уровня защиты базы данных. Одной из таких рекомендаций является обеспечение разных прав доступа для пользователей, как уже было упомянуто ранее. Более того, обычный пользователь работает не с таблицами напрямую, а с представлениями, что представляет собой дополнительный уровень защиты данных, так как пользователь никак не может повлиять на данные в таблицах в базе данных [13].

Отдельное внимание стоит уделить хранению пользователей и данных этих пользователей, а также их передаче.

Когда происходит регистрация пользователя, пароль, который он вводит, не вносится в поле с паролем в таком виде, в каком он введён. Перед этим пароль предварительно шифруется с помощью адаптивной криптографической хеш-функции формирования ключа *bcrypt* [14]. *bcrypt* позволяет создать платформу защиты паролей, которая может развиваться вместе с аппаратными технологиями для защиты от угроз. Таким образом, после создания пользователя, пароль такого пользователя выглядит так, как представлен на рисунке 15.



Рисунок 15 – Пример зашифрованной записи

Как можно увидеть, пароли действительно зашифрованы, и, следовательно, защищены. Когда происходит аутентификация, пароль, введённый пользователем, обрабатывается хеш-функцией и сравнивается с полем пароля в базе данных (при условии соответствия адресов электронных почт). Если хеши паролей совпадают, то аутентификация будет произведена успешно, и пользователь получит так называемый токен, который будет хранится в *cookies* браузера.

Стоит отменить, что *HTTP Cookie* – это текстовые файлы с небольшими фрагментами данных (например, в случае данного курсового проекта – данные пользователя – идентификатор, почта, пароль, имя), которые используются для управления сеансом, персонализации и мониторинга [15]. Таким образом, при использовании *cookie* отпадает необходимость процедуры входа в учетную запись после перезагрузки вкладки или браузера. Как можно заметить, *cookie* свободно передаются между клиентом и сервером, поэтому могут быть перехвачены.

Безопасные (*secure*) *cookie* отсылаются на сервер только если запрос выполняется по протоколу *SSL* и *HTTPS*. Однако важные данные никогда не следует передавать или хранить в *cookies*, поскольку сам их механизм весьма уязвим в отношении безопасности, а флаг *secure* никакого дополнительного шифрования или средств защиты не обеспечивает.

В связи с этим, имеет смысл указывать флаг «*httpOnly*» для *Cookie*. *Cookie* *HTTPonly* не доступны для *JavaScript* что помогает избежать межсайтового скриптинга [16]. Этот флаг используется для тех *cookie*, к которым не требуется обращаться через *JavaScript*. В частности, если *cookies* используются только для поддержки сеанса, то в *JavaScript* они не нужны, так что в этом случае следует устанавливать флаг *HttpOnly*.

# ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта была спроектирована база данных, которая содержит в себе 13 сущностей. Каждая сущность имеет первичный ключ, данные нормализованы, транзитивные отношения отсутствуют, таким образом база данных приведена к третьей нормальной форме. В качестве СУБД был выбран *MySQL*, настройка базы данных проводилась в *MySQL* *Workbench* 8.0. В процессе разработки базы данных были учтены особенности, которые перечислены в задании: база данных должна отвечать требованиям надёжности, минимальной избыточности, целостности данных, содержать не менее десяти сущностей, а ее схема должна быть приведена к третьей нормальной форме. База данных должна поддерживать основные современные средства для работы и администрирования.

Были реализованы следующие объекты базы данных в количестве не менее трех каждый: индекс, триггер, хранимая процедура. Также была реализована система разграничения прав доступа к данным минимум для двух ролей пользователей.

Что касается веб-приложения, то приложение разработано на языке *JavaScript* с использованием фреймворка *Express* для создания *API*, а также с использованием *React* и *Axios* для создания пользовательского интерфейса. В приложении реализована система условного рендеринга, которая основывается на разграничении прав доступа к базе данных. Интерфейс приложения разработан удобным и интуитивно понятным.

Курсовой проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 96,48%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников». Скриншот приведен в приложении А.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Проектирование БД [Электронный ресурс]. – Режим доступа: https://www.lucidchart.com/pages/database-diagram/database-design. – Дата доступа: 15.11.2021.

[2] Типы данных *MySQL* [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/sql/sql\_datatypes.asp. – Дата доступа: 15.11.2021.

[3] Нормализация отношений [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/254773/. – Дата доступа: 21.11.2021.

[4] Первичный ключ [Электронный ресурс]. – Режим доступа: https://www.techopedia.com/definition/5547/primary-key. – Дата доступа: 17.11.2021.

[5] Запросы *MySQL* [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/mysql-common-mysql-queries/. – Дата доступа: 20.11.2021.

[6] *JavaScript* [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. – Дата доступа: 23.11.2021.

[7] *VS Code* [Электронный ресурс]. – Режим доступа: https://code.visualstudio.com/docs/editor/whyvscode. – Дата доступа: 25.11.2021.

[8] *DB documentation* [Электронный ресурс]. – Режим доступа: https://www.holistics.io/blog/top-5-database-documentation-tools-for-any-teams-in-2020/. – Дата доступа: 28.11.2021.

[9] Руководство пользователя [Электронный ресурс]. – Режим доступа: https://zenkit.com/en/blog/how-to-write-a-user-manual/. – Дата доступа: 01.12.2021.

[10] Администрирование [Электронный ресурс]. – Режим доступа: https://www.techopedia.com/definition/24080/database-administration. – Дата доступа: 03.12.2021.

[11] Пользователи и роли [Электронный ресурс]. – Режим доступа: https://satoricyber.com/sql-server-security/sql-server-roles/. – Дата доступа: 07.12.2021.

[12] *MySQL Safety* [Электронный ресурс]. – Режим доступа: https://www.mysql.com/why-mysql/presentations/mysql-security-best-practices/. – Дата доступа: 08.12.2021.

[13] Безопасность БД [Электронный ресурс]. – Режим доступа: https://www.tripwire.com/state-of-security/featured/database-security-best-practices-you-should-know/. – Дата доступа: 09.12.2021.

[14] *Bcrypt* [Электронный ресурс]. – Режим доступа: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/. – Дата доступа: 09.12.2021.

[15] *Cookie* [Электронный ресурс]. – Режим доступа: https://www.kaspersky.com/resource-center/definitions/cookies. – Дата доступа: 10.12.2021.

[16] *HTTP Cookie* [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTTP/Cookies. – Дата доступа: 11.12.2021.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Проверка на оригинальность**

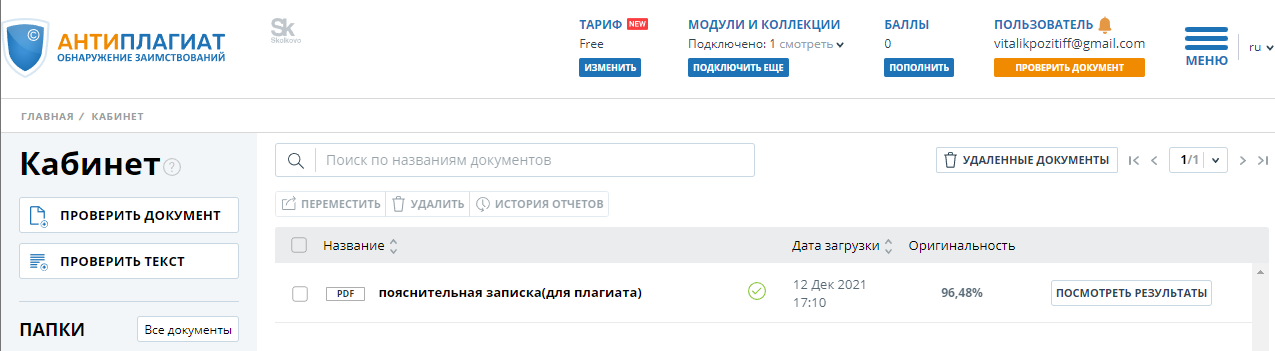


Рисунок А.1 – Проверка на оригинальность

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Скрипт генерации БД**

DROP SCHEMA IF EXISTS `autoshop` ;

CREATE SCHEMA IF NOT EXISTS `autoshop`;

USE `autoshop` ;

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

-- Table `autoshop`.`manufacturers`

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

DROP TABLE IF EXISTS `autoshop`.`manufacturers` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`manufacturers` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `company\_name` VARCHAR(128) NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE INDEX `id\_UNIQUE` (`id` ASC) VISIBLE)

ENGINE = InnoDB

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

-- Table `autoshop`.`userrrs`

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

DROP TABLE IF EXISTS `autoshop`.`userrrs` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`userrrs` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `FnameLname` VARCHAR(128) NOT NULL,

  `email` VARCHAR(128) NOT NULL,

  `password` VARCHAR(255) NOT NULL,

  `role` ENUM('user', 'admin') NOT NULL,

  UNIQUE INDEX `customer\_id` (`id` ASC) VISIBLE,

  UNIQUE INDEX `email\_UNIQUE` (`email` ASC) VISIBLE,

  PRIMARY KEY (`email`))

ENGINE = InnoDB

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

-- Table `autoshop`.`qualifications`

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

DROP TABLE IF EXISTS `autoshop`.`qualifications` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`qualifications` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `qual\_name` VARCHAR(45) NOT NULL,

  `koef` DECIMAL(2,1) NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE INDEX `order\_id\_UNIQUE` (`id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `autoshop`.`employees`

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

DROP TABLE IF EXISTS `autoshop`.`employees` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`employees` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `first\_name` VARCHAR(128) NOT NULL,

  `last\_name` VARCHAR(128) NOT NULL,

  `qualification\_id` INT NOT NULL,

  `isFree` ENUM('free', 'busy') NOT NULL DEFAULT 'free',

  UNIQUE INDEX `emp\_id` (`id` ASC) VISIBLE,

  PRIMARY KEY (`id`),

  INDEX `emp\_kval\_idx` (`qualification\_id` ASC) VISIBLE,

  CONSTRAINT `emp\_kval`

    FOREIGN KEY (`qualification\_id`)

    REFERENCES `autoshop`.`qualifications` (`id`))

ENGINE = InnoDB

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

-- Table `autoshop`.`vehicle\_types`

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

DROP TABLE IF EXISTS `autoshop`.`vehicle\_types` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`vehicle\_types` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `type\_name` VARCHAR(64) NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE INDEX `type\_id\_UNIQUE` (`id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `autoshop`.`car\_engines`

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

DROP TABLE IF EXISTS `autoshop`.`car\_engines` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`car\_engines` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `engine\_name` VARCHAR(128) NOT NULL,

  `engine\_price` DECIMAL(8,2) NOT NULL,

  `amount` INT NOT NULL,

  UNIQUE INDEX `engine\_id\_UNIQUE` (`id` ASC) VISIBLE,

  PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `autoshop`.`transmissions`

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

DROP TABLE IF EXISTS `autoshop`.`transmissions` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`transmissions` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `transmission\_name` VARCHAR(128) NOT NULL,

  `transmission\_price` DECIMAL(8,2) NOT NULL,

  `amount` INT NOT NULL,

  UNIQUE INDEX `engine\_id\_UNIQUE` (`id` ASC) VISIBLE,

  PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `autoshop`.`turbos`

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

DROP TABLE IF EXISTS `autoshop`.`turbos` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`turbos` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `turbo\_name` VARCHAR(128) NOT NULL,

  `turbo\_price` DECIMAL(8,2) NOT NULL,

  `amount` INT NOT NULL,

  UNIQUE INDEX `engine\_id\_UNIQUE` (`id` ASC) VISIBLE,

  PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `autoshop`.`wheels`

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

DROP TABLE IF EXISTS `autoshop`.`wheels` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`wheels` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `wheels\_name` VARCHAR(128) NOT NULL,

  `wheels\_price` DECIMAL(8,2) NOT NULL,

  `amount` INT NOT NULL,

  UNIQUE INDEX `engine\_id\_UNIQUE` (`id` ASC) VISIBLE,

  PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `autoshop`.`vehicle\_models`

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

DROP TABLE IF EXISTS `autoshop`.`vehicle\_models` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`vehicle\_models` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `model\_name` VARCHAR(128) NOT NULL,

  `manufacturer\_id` INT NOT NULL,

  `vin` VARCHAR(255) NOT NULL,

  `type\_id` INT NOT NULL,

  `engine\_id` INT NOT NULL,

  `transmission\_id` INT NOT NULL,

  `turbo\_id` INT NOT NULL,

  `wheels\_id` INT NOT NULL,

  `vehicle\_stock\_price` DECIMAL NOT NULL,

  `car\_description` TEXT NULL,

  `is\_sold` ENUM('available', 'busy', 'sold') NOT NULL DEFAULT 'available',

  `photo\_path` VARCHAR(255) NULL,

  PRIMARY KEY (`id`),

  UNIQUE INDEX `model\_id\_UNIQUE` (`id` ASC) VISIBLE,

  INDEX `clients\_car\_idx` (`manufacturer\_id` ASC) VISIBLE,

  INDEX `cars\_type\_idx` (`type\_id` ASC) VISIBLE,

  INDEX `car\_engine\_idx` (`engine\_id` ASC) VISIBLE,

  INDEX `car\_transmission\_idx` (`transmission\_id` ASC) VISIBLE,

  INDEX `car\_turbo\_idx` (`turbo\_id` ASC) VISIBLE,

  INDEX `car\_wheels\_idx` (`wheels\_id` ASC) VISIBLE,

  CONSTRAINT `clients\_car`

    FOREIGN KEY (`manufacturer\_id`)

    REFERENCES `autoshop`.`manufacturers` (`id`),

  CONSTRAINT `cars\_type`

    FOREIGN KEY (`type\_id`)

    REFERENCES `autoshop`.`vehicle\_types` (`id`),

  CONSTRAINT `car\_engine`

    FOREIGN KEY (`engine\_id`)

    REFERENCES `autoshop`.`car\_engines` (`id`),

  CONSTRAINT `car\_transmission`

    FOREIGN KEY (`transmission\_id`)

    REFERENCES `autoshop`.`transmissions` (`id`),

  CONSTRAINT `car\_turbo`

    FOREIGN KEY (`turbo\_id`)

    REFERENCES `autoshop`.`turbos` (`id`),

  CONSTRAINT `car\_wheels`

    FOREIGN KEY (`wheels\_id`)

    REFERENCES `autoshop`.`wheels` (`id`))

ENGINE = InnoDB;

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

-- Table `autoshop`.`order\_types`

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

DROP TABLE IF EXISTS `autoshop`.`order\_types` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`order\_types` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `type\_name` VARCHAR(45) NOT NULL,

  `koef` DECIMAL(2,1) NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE INDEX `order\_id\_UNIQUE` (`id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `autoshop`.`payment\_types`

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

DROP TABLE IF EXISTS `autoshop`.`payment\_types` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`payment\_types` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `type\_name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE INDEX `order\_id\_UNIQUE` (`id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `autoshop`.`Orders`

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

DROP TABLE IF EXISTS `autoshop`.`Orders` ;

CREATE TABLE IF NOT EXISTS `autoshop`.`Orders` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `shop\_name` VARCHAR(128) NOT NULL,

  `emp\_id` INT NOT NULL,

  `vehicle\_id` INT NOT NULL,

  `order\_date` DATETIME NOT NULL,

  `order\_type` INT NOT NULL DEFAULT 1,

  `payment\_type` INT NOT NULL,

  `customer\_email` VARCHAR(128) NOT NULL,

  `order\_status` ENUM('pending', 'confirmed') NOT NULL,

  `price` DECIMAL(10,2) NOT NULL,

  UNIQUE INDEX `order\_id\_UNIQUE` (`id` ASC) VISIBLE,

  INDEX `order-car\_idx` (`vehicle\_id` ASC) VISIBLE,

  INDEX `order-type\_idx` (`order\_type` ASC) VISIBLE,

  INDEX `oplata-type\_idx` (`payment\_type` ASC) VISIBLE,

  PRIMARY KEY (`id`),

  INDEX `whobuycar\_idx` (`customer\_email` ASC) VISIBLE,

  INDEX `order-emp\_idx` (`emp\_id` ASC) VISIBLE,

  CONSTRAINT `order-emp`

    FOREIGN KEY (`emp\_id`)

    REFERENCES `autoshop`.`employees` (`id`),

  CONSTRAINT `order-car`

    FOREIGN KEY (`vehicle\_id`)

    REFERENCES `autoshop`.`vehicle\_models` (`id`),

  CONSTRAINT `order-type`

    FOREIGN KEY (`order\_type`)

    REFERENCES `autoshop`.`order\_types` (`id`),

  CONSTRAINT `payment-type`

    FOREIGN KEY (`payment\_type`)

    REFERENCES `autoshop`.`payment\_types` (`id`),

  CONSTRAINT `whobuycar`

    FOREIGN KEY (`customer\_email`)

    REFERENCES `autoshop`.`userrrs` (`email`))

ENGINE = InnoDB;

USE `autoshop` ;

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

-- Placeholder table for view `autoshop`.`carForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`carForUser` (`id` INT, `company\_name` INT, `model\_name` INT, `wheels\_name` INT, `engine\_name` INT, `transmission\_name` INT, `turbo\_name` INT, `vin` INT, `vehicle\_stock\_price` INT, `car\_description` INT, `photo\_path` INT);

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

-- Placeholder table for view `autoshop`.`ordersForUsers`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`ordersForUsers` (`order\_status` INT, `order\_id` INT, `shop\_name` INT, `price` INT, `order\_date` INT, `order\_type` INT, `payment\_type` INT, `customer\_email` INT, `emp\_id` INT, `model\_name` INT, `wheels\_name` INT, `engine\_name` INT, `turbo\_name` INT, `transmission\_name` INT, `vehicle\_id` INT);

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

-- Placeholder table for view `autoshop`.`wheelsForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`wheelsForUser` (`id` INT, `wheels\_name` INT, `wheels\_price` INT);

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

-- Placeholder table for view `autoshop`.`enginesForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`enginesForUser` (`id` INT, `engine\_name` INT, `engine\_price` INT);

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

-- Placeholder table for view `autoshop`.`turbosForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`turbosForUser` (`id` INT, `turbo\_name` INT, `turbo\_price` INT);

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

-- Placeholder table for view `autoshop`.`transmissionsForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`transmissionsForUser` (`id` INT, `transmission\_name` INT, `transmission\_price` INT);

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

-- Placeholder table for view `autoshop`.`paymentTypesForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`paymentTypesForUser` (`id` INT, `type\_name` INT);

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

-- Placeholder table for view `autoshop`.`orderTypesForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`orderTypesForUser` (`id` INT, `type\_name` INT, `koef` INT);

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

-- Placeholder table for view `autoshop`.`qualForUsers`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`qualForUsers` (`id` INT, `qual\_name` INT, `koef` INT);

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

-- Placeholder table for view `autoshop`.`empForUser`

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

CREATE TABLE IF NOT EXISTS `autoshop`.`empForUser` (`id` INT, `first\_name` INT, `qualification\_id` INT);

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

-- procedure showCarsWithFilter

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`showCarsWithFilter`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `showCarsWithFilter` (companyName varchar(20),maxprice decimal(10,2), minprice decimal(10,2))

BEGIN

  select \* from carForUser where company\_name = companyName

    and vehicle\_stock\_price < maxprice and vehicle\_stock\_price > minprice;

END$$

DELIMITER ;

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

-- procedure showCars

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`showCars`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `showCars` ()

BEGIN

  select \* from carForUser;

END$$

DELIMITER ;

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

-- procedure addWheelsToWarehouse

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`addWheelsToWarehouse`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `addWheelsToWarehouse` (howMany int, id int)

BEGIN

  update wheels set amount = amount + howMany where wheels.id = id;

END$$

DELIMITER ;

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

-- procedure addEnginesToWarehouse

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`addEnginesToWarehouse`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `addEnginesToWarehouse` (howMany int, id int)

BEGIN

  update car\_engines set amount = amount + howMany where car\_engines.id=id;

END$$

DELIMITER ;

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

-- procedure addTurbosToWarehouse

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`addTurbosToWarehouse`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `addTurbosToWarehouse` (howMany int, id int)

BEGIN

  update turbos set amount = amount + howMany where turbos.id = id;

END$$

DELIMITER ;

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

-- procedure addTransmissionsToWarehouse

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`addTransmissionsToWarehouse`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `addTransmissionsToWarehouse` (howMany int, id int)

BEGIN

  update transmissions set amount = amount + howMany where transmissions.id = id;

END$$

DELIMITER ;

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

-- procedure createOrder

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

USE `autoshop`;

DROP procedure IF EXISTS `autoshop`.`createOrder`;

DELIMITER $$

USE `autoshop`$$

CREATE PROCEDURE `createOrder` (shop\_name varchar(30), vehicle\_id int, order\_date datetime, order\_type int, payment\_type int, customer\_email varchar(30), engineID int, wheelsID int, transmissionID int, turboID int)

BEGIN

  declare emp\_id int;

  declare total\_price decimal(10,2) default 0;

    declare enginePrice decimal(10,2) default 0;

    declare wheelsPrice decimal(10,2) default 0;

    declare transmissionPrice decimal(10,2) default 0;

    declare turboPrice decimal(10,2) default 0;

    declare orderkoef int default 0;

  update vehicle\_models set engine\_id = engineID, wheels\_id = wheelsID, turbo\_id = turboID, transmission\_id = transmissionID where id = vehicle\_id;

  SELECT id into emp\_id FROM employees where isFree='free'

    ORDER BY RAND()

    LIMIT 1;

    select engine\_price into enginePrice from car\_engines where id = engineID;

  select wheels\_price into wheelsPrice from wheels where id = wheelsID ;

    select turbo\_price into transmissionPrice from turbos where id = turboID;

  select transmission\_price into turboPrice from transmissions where id = transmissionID;

  select koef into orderkoef from order\_types where id = order\_type;

  set total\_price = (enginePrice+wheelsPrice+transmissionPrice+turboPrice)\*orderkoef;

    insert into orders values (0,shop\_name,emp\_id,vehicle\_id,order\_date,order\_type,payment\_type, customer\_email,'pending',total\_price);

END$$

DELIMITER ;

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

-- View `autoshop`.`carForUser`

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

DROP TABLE IF EXISTS `autoshop`.`carForUser`;

DROP VIEW IF EXISTS `autoshop`.`carForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `carForUser` AS

select vehicle\_models.id,

    manufacturers.company\_name,

    vehicle\_models.model\_name,

        wheels.wheels\_name,

        car\_engines.engine\_name,

        transmissions.transmission\_name,

        turbos.turbo\_name,

        vehicle\_models.vin,

        ((select engine\_price from car\_engines where car\_engines.id = vehicle\_models.engine\_id)+

        (select wheels\_price from wheels where wheels.id = vehicle\_models.wheels\_id)+

        (select transmission\_price from transmissions where transmissions.id = vehicle\_models.transmission\_id)+

        (select turbo\_price from turbos where turbos.id = vehicle\_models.turbo\_id)) as vehicle\_stock\_price,

        vehicle\_models.car\_description,

        vehicle\_models.photo\_path

        from vehicle\_models

  LEFT JOIN wheels on wheels.id = vehicle\_models.wheels\_id

    LEFT JOIN car\_engines on car\_engines.id = vehicle\_models.engine\_id

    LEFT JOIN manufacturers on manufacturers.id = vehicle\_models.manufacturer\_id

    LEFT JOIN turbos on turbos.id = vehicle\_models.turbo\_id

    LEFT JOIN transmissions on transmissions.id = vehicle\_models.transmission\_id

    where is\_sold = 'available';

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

-- View `autoshop`.`ordersForUsers`

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

DROP TABLE IF EXISTS `autoshop`.`ordersForUsers`;

DROP VIEW IF EXISTS `autoshop`.`ordersForUsers` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `ordersForUsers` AS

select  Orders.order\_status,

    Orders.id as order\_id,

    Orders.shop\_name,

        price,

    order\_date,

        order\_types.type\_name as order\_type,

        payment\_types.type\_name as payment\_type,

        customer\_email,

    employees.id as emp\_id,

        vehicle\_models.model\_name,

        wheels.wheels\_name,

        car\_engines.engine\_name,

        turbos.turbo\_name,

        transmissions.transmission\_name,

        vehicle\_models.id as vehicle\_id

        from Orders

  LEFT JOIN employees on employees.id = Orders.emp\_id

    LEFT JOIN vehicle\_models on vehicle\_models.id = Orders.vehicle\_id

    LEFT JOIN wheels on wheels.id = vehicle\_models.wheels\_id

    LEFT JOIN car\_engines on car\_engines.id = vehicle\_models.engine\_id

    LEFT JOIN manufacturers on manufacturers.id = vehicle\_models.manufacturer\_id

    LEFT JOIN turbos on turbos.id = vehicle\_models.turbo\_id

    LEFT JOIN transmissions on transmissions.id = vehicle\_models.transmission\_id

    left join order\_types on order\_types.id = orders.order\_type

    left join payment\_types on payment\_types.id = orders.payment\_type;

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

-- View `autoshop`.`wheelsForUser`

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

DROP TABLE IF EXISTS `autoshop`.`wheelsForUser`;

DROP VIEW IF EXISTS `autoshop`.`wheelsForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `wheelsForUser` AS

  select id, wheels\_name, wheels\_price from wheels;

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

-- View `autoshop`.`enginesForUser`

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

DROP TABLE IF EXISTS `autoshop`.`enginesForUser`;

DROP VIEW IF EXISTS `autoshop`.`enginesForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `enginesForUser` AS

  select id, engine\_name, engine\_price from car\_engines;

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

-- View `autoshop`.`turbosForUser`

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

DROP TABLE IF EXISTS `autoshop`.`turbosForUser`;

DROP VIEW IF EXISTS `autoshop`.`turbosForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `turbosForUser` AS

  select id, turbo\_name, turbo\_price from turbos;

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

-- View `autoshop`.`transmissionsForUser`

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

DROP TABLE IF EXISTS `autoshop`.`transmissionsForUser`;

DROP VIEW IF EXISTS `autoshop`.`transmissionsForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `transmissionsForUser` AS

  select id, transmission\_name, transmission\_price from transmissions;

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

-- View `autoshop`.`paymentTypesForUser`

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

DROP TABLE IF EXISTS `autoshop`.`paymentTypesForUser`;

DROP VIEW IF EXISTS `autoshop`.`paymentTypesForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `paymentTypesForUser` AS

  select \* from payment\_types;

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

-- View `autoshop`.`orderTypesForUser`

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

DROP TABLE IF EXISTS `autoshop`.`orderTypesForUser`;

DROP VIEW IF EXISTS `autoshop`.`orderTypesForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `orderTypesForUser` AS

  select \* from order\_types;

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

-- View `autoshop`.`qualForUsers`

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

DROP TABLE IF EXISTS `autoshop`.`qualForUsers`;

DROP VIEW IF EXISTS `autoshop`.`qualForUsers` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `qualForUsers` AS

  select \* from qualifications;

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

-- View `autoshop`.`empForUser`

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

DROP TABLE IF EXISTS `autoshop`.`empForUser`;

DROP VIEW IF EXISTS `autoshop`.`empForUser` ;

USE `autoshop`;

CREATE  OR REPLACE VIEW `empForUser` AS

  select id,first\_name,qualification\_id from employees where isFree = 'free';

USE `autoshop`;

DELIMITER $$

USE `autoshop`$$

DROP TRIGGER IF EXISTS `autoshop`.`Orders\_AFTER\_INSERT` $$

USE `autoshop`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `autoshop`.`Orders\_AFTER\_INSERT` AFTER INSERT ON `Orders` FOR EACH ROW

BEGIN

    DECLARE id\_exists Boolean;

       -- Check BookingRequest table

       SELECT 1

       INTO @id\_exists

       FROM orders

       WHERE orders.id= NEW.id;

       IF @id\_exists = 1

       THEN

           UPDATE employees

       SET isFree = 'busy'

       WHERE employees.id = NEW.emp\_id;

       UPDATE vehicle\_models

       SET is\_sold = 'busy'

       WHERE vehicle\_models.id = NEW.vehicle\_id;

       UPDATE wheels

       SET amount = amount - 1

       WHERE (select wheels\_id from vehicle\_models where vehicle\_models.id = NEW.vehicle\_id) = wheels.id;

           UPDATE car\_engines

       SET amount = amount - 1

       WHERE (select engine\_id from vehicle\_models where vehicle\_models.id = NEW.vehicle\_id) = car\_engines.id;

           UPDATE turbos

       SET amount = amount - 1

       WHERE (select turbo\_id from vehicle\_models where vehicle\_models.id = NEW.vehicle\_id) = turbos.id;

           UPDATE transmissions

       SET amount = amount - 1

       WHERE (select transmission\_id from vehicle\_models where vehicle\_models.id = NEW.vehicle\_id) = transmissions.id;

        END IF;

END$$

USE `autoshop`$$

DROP TRIGGER IF EXISTS `autoshop`.`Orders\_AFTER\_UPDATE` $$

USE `autoshop`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `autoshop`.`Orders\_AFTER\_UPDATE` AFTER UPDATE ON `Orders` FOR EACH ROW

BEGIN

     UPDATE employees

     SET isFree = 'free'

     WHERE employees.id = NEW.emp\_id;

     UPDATE vehicle\_models

     SET is\_sold = 'sold'

     WHERE vehicle\_models.id = NEW.vehicle\_id;

END$$

USE `autoshop`$$

DROP TRIGGER IF EXISTS `autoshop`.`Orders\_AFTER\_DELETE` $$

USE `autoshop`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `autoshop`.`Orders\_AFTER\_DELETE` AFTER DELETE ON `Orders` FOR EACH ROW

BEGIN

     UPDATE employees

     SET isFree = 'free'

     WHERE employees.id = OLD.emp\_id;

     UPDATE vehicle\_models

     SET is\_sold = 'available'

     WHERE vehicle\_models.id = OLD.vehicle\_id;

     UPDATE wheels

     SET amount = amount + 1

     WHERE (select wheels\_id from vehicle\_models where vehicle\_models.id = OLD.vehicle\_id) = wheels.id;

     UPDATE car\_engines

     SET amount = amount + 1

     WHERE (select engine\_id from vehicle\_models where vehicle\_models.id = OLD.vehicle\_id) = car\_engines.id;

     UPDATE turbos

     SET amount = amount + 1

     WHERE (select turbo\_id from vehicle\_models where vehicle\_models.id = OLD.vehicle\_id) = turbos.id;

     UPDATE transmissions

     SET amount = amount + 1

     WHERE (select transmission\_id from vehicle\_models where vehicle\_models.id = OLD.vehicle\_id) = transmissions.id;

END$$

DELIMITER ;

SET SQL\_MODE = '';

DROP USER IF EXISTS user;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'user' IDENTIFIED BY 'user';

GRANT SELECT ON TABLE `autoshop`.`manufacturers` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`vehicle\_types` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`vehicle\_models` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`car\_engines` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`transmissions` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`turbos` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`wheels` TO 'user';

GRANT SELECT, DELETE, INSERT ON TABLE `autoshop`.`Orders` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`order\_types` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`payment\_types` TO 'user';

GRANT SELECT ON TABLE `autoshop`.`carForUser` TO 'user';

SET SQL\_MODE = '';

DROP USER IF EXISTS emp;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'emp' IDENTIFIED BY 'emp';

GRANT DELETE, SELECT, UPDATE, INSERT ON TABLE `autoshop`.`manufacturers` TO 'emp';

GRANT SELECT, INSERT, DELETE, UPDATE ON TABLE `autoshop`.`vehicle\_types` TO 'emp';

GRANT SELECT, DELETE, INSERT, UPDATE ON TABLE `autoshop`.`vehicle\_models` TO 'emp';

GRANT SELECT, DELETE, INSERT, UPDATE ON TABLE `autoshop`.`car\_engines` TO 'emp';

GRANT SELECT, DELETE, INSERT, UPDATE ON TABLE `autoshop`.`transmissions` TO 'emp';

GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE `autoshop`.`turbos` TO 'emp';

GRANT UPDATE, SELECT, INSERT, DELETE ON TABLE `autoshop`.`wheels` TO 'emp';

GRANT SELECT, UPDATE ON TABLE `autoshop`.`Orders` TO 'emp';

GRANT SELECT ON TABLE `autoshop`.`qualifications` TO 'emp';

SET SQL\_MODE = '';

DROP USER IF EXISTS admin;

SET SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

CREATE USER 'admin' IDENTIFIED BY 'admin';

GRANT ALL ON autoshop.\* TO 'admin';

GRANT ALL ON TABLE `autoshop`.`carForUser` TO 'admin';

GRANT ALL ON procedure `autoshop`.`showCarsWithFilter` TO 'admin';

-- users

START TRANSACTION;

USE `autoshop`;

INSERT INTO `autoshop`.`userrrs` (`id`, `FnameLname`, `email`, `password`, `role`) VALUES (8, 'Vitaly Gonchar', 'test1@gmail.com', '$2b$10$8ftboVUEB15eyNSaZUHqUO75P/amU2KhW4qX7vu50PhCTRPdKeX22', 'admin');

INSERT INTO `autoshop`.`userrrs` (`id`, `FnameLname`, `email`, `password`, `role`) VALUES (13, 'test user2', 'test2@gmail.com', '$2b$10$sf6EnvoEYY2GOFvl3swOqO35UWyyzjYGubF5GittEEMBgg7iacIhu', 'user');

COMMIT;

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Листинг программного кода**

const express = require('express')

const mysql = require("mysql");

const morgan = require('morgan')

const bodyParser = require('body-parser')

const cookieParser = require("cookie-parser");

const session = require("express-session");

const bcrypt = require("bcrypt");

const saltRounds = 10;

const app = express()

const wheelRoutes = require('./routes/wheels')

const manufacturerRoutes = require('./routes/manufacturers')

const engineRoutes = require('./routes/engines')

const turboRoutes = require('./routes/turbos')

const transmissionRoutes = require('./routes/transmissions')

const vehicle\_typesRoutes = require('./routes/vehicle\_types')

const payment\_typesRoutes = require('./routes/payment\_types')

const order\_typesRoutes = require('./routes/order\_types')

const qualificationRoutes = require('./routes/qualifications')

const employeeRoutes = require('./routes/employees')

const orderRoutes = require('./routes/orders')

const userRoutes = require('./routes/userrrs')

const vehicleModelRoutes = require('./routes/vehicle\_models')

const vehiclesForUser = require('./routes/vehiclesListForUsers')

// nodemon app.js

app.use(morgan('short'))

app.use(bodyParser.urlencoded({extended: false}))

app.use(bodyParser.json())

app.use(express.static('./public'))

app.use(function(req, res, next) {

res.header("Access-Control-Allow-Origin", "http://localhost:8081"); // update to match the domain you will make the request from

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

res.header("Access-Control-Allow-Methods", "POST, PUT, GET, OPTIONS, DELETE");

res.header("Access-Control-Allow-Credentials", "true");

next();

});

app.use(

session({

key: "Token",

secret: "super\_Secret",

resave: false,

saveUninitialized: false,

cookie: {

maxAge: 86400000,

},

})

);

app.use('/wheels', wheelRoutes)

app.use('/manufacturers', manufacturerRoutes)

app.use('/engines', engineRoutes)

app.use('/turbos', turboRoutes)

app.use('/transmissions', transmissionRoutes)

app.use('/vehicle\_types', vehicle\_typesRoutes)

app.use('/payment\_types', payment\_typesRoutes)

app.use('/order\_types', order\_typesRoutes)

app.use('/qualifications', qualificationRoutes)

app.use('/employees', employeeRoutes)

app.use('/orders', orderRoutes)

app.use('/userrrs', userRoutes)

app.use('/vehicle\_models', vehicleModelRoutes)

app.use('/cars',vehiclesForUser)

app.use(cookieParser());

app.use((req, res, next) => {

// This reads the accept-language header

// and returns the language if found or false if not

const lang = req.acceptsLanguages('ru', 'en')

if (lang) { // if found, attach it as property to the request

req.lang = lang

} else { // else set the default language

req.lang = 'en'

}

next()

})

const db = mysql.createConnection({

user: "root",

host: "localhost",

password: "7557475",

database: "autoshop",

});

app.post("/register", (req, res) => {

const FnameLname = req.body.FnameLname;

const email = req.body.email;

const password = req.body.password;

bcrypt.hash(password, saltRounds, (err, hash) => {

if (err) {

console.log(err);

}

db.query(

`INSERT INTO userrrs (fnameLname,email, password,role) VALUES ('${FnameLname}','${email}','${hash}','user')`, (err, res) => {

if (err) {

console.log(err)

return(err)

// return res.status(400).send({

// msg: err

// });

}

}

);

return res.status(201).send({

msg: 'The user has been registerd with us!'

});

});

});

app.get("/logout", (req, res, next) => {

res.clearCookie('Token', { path: '/' })

res.status(200).send({msg: 'Logout complete'})

next()

});

app.get("/login", (req, res) => {

if (req.session.user) {

res.send({ loggedIn: true, user: req.session.user });

} else {

res.send({ loggedIn: false });

}

});

app.post("/login", (req, res) => {

const email = req.body.email;

const password = req.body.password;

db.query(

"SELECT \* FROM userrrs WHERE email = ?;",

email,

(err, result) => {

if (err) {

res.send({ err: err });

}

if (result.length > 0) {

bcrypt.compare(password, result[0].password, (error, response) => {

if (response) {

req.session.user = result;

console.log(req.session.user);

res.send(result);

} else {

res.send({ message: "Wrong email/password combination!" });

}

});

} else {

res.send({ message: "User doesn't exist" });

}

}

);

});

const PORT = process.env.PORT || 5000

// localhost:5000

app.listen(PORT, () => {

console.log("Server is up and listening on: " + PORT)

})

# ПРИЛОЖЕНИЕ Г

**(обязательное)**

**Ведомость документов**

