**Департамент образования И НАУКИ города Москвы**

**ГБПОУ г. МОСКВЫ МОСКОВСКИЙ АВТОМОБИЛЬНО – ДОРОЖНЫЙ**

**КОЛЛЕДЖ им. А. А. НИКОЛАЕВА**

**КУРСОВОЙ ПРОЕКТ**

**Тема:**  **Проектирование базы данных для ПО «Мотосалон «Temarov»**

Студент\_\_\_\_Шайко Н.И.

Преподаватель\_\_\_\_Каримов М.Т.

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

Дата «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Москва

2022

**Департамент образования И НАУКИ города Москвы**

**ГБПОУ г. МОСКВЫ МОСКОВСКИЙ АВТОМОБИЛЬНО – ДОРОЖНЫЙ**

**КОЛЛЕДЖ им. А. А. НИКОЛАЕВА**

**З А Д А Н И Е**

к курсовому проекту

студента дневного отделения, группы 3/11ИП

специальности 09.02.07 «Информационные системы и программирование»

Фамилия, имя, отчество Шайко Никита Игоревич

Тема курсового проекта Проектирование базы данных для ПО «Мотосалон «Temarov»

1. Постановка задачи:

Основной целью курсового проекта является проектирование базы данных для ПО «Мотосалон “Temari Nara”» который занимается торговлей мотоциклами, их деталями, узлами и принадлежностями, техническим обслуживанием и ремонтом мотоциклов.

Функционал сервера баз данных должен обладать возможностью хранения и обработки данных о технике, пользователях, сервисе, о компании, экипировке, характеристики, марке, тип, модификации, серии, моделей, а также данных справочников, поставщиков, накладных.

2. Исходные данные:

Информация о пользователях, сотрудниках, характеристиках (двигатель, размеры и масса, трансмиссия, динамика и экономичность), значения характеристик, типы транспорта, модификации, марки, модели, серии.

3. Перечень вопросов, подлежащих разработке в курсовом проекте:

1. Постановка задачи и исследование предметной области;
2. Обоснование выбора ОС, СУБД и языка программирования;
3. Проектирование Use-case диаграммы;
4. Проектирование концептуальной модели данных;
5. Проектирование информационно-логической модели данных;
6. Проектирование физической модели данных;
7. Проектирование интерфейса пользователя. ­
8. Тестирование ПО.

Задание принял к исполнению « » \_\_\_\_\_\_ 2022 г.

Студент Н.И.Шайко / И.О. Фамилия подпись

Срок сдачи « » 2022 г.

Руководитель курсового проекта М.Т. Каримов / И.О. Фамилия подпись

Задание рассмотрено на заседании цикловой комиссии « » 2022 г.

Председатель ПЦК / Т.Г. Сянина\_\_\_\_ подпись И.О. Фамилия,

**СОДЕРЖАНИЕ**

[1.ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ 5](#_Toc85012756)

[1.1 Описание предметной области 5](#_Toc85012757)

[1.2 Виды жизненных циклов программного обеспечения 7](#_Toc85012758)

[1.3 Анализ, обоснование и выбор ОС, СУБД и языка программирования 18](#_Toc85012759)

[2. РАЗДЕЛ ПРОЕКТИРОВАНИЯ 46](#_Toc85012760)

[2.1 Use – case диаграмма 46](#_Toc85012761)

[2.2 Концептуальная модель БД 49](#_Toc85012762)

[2.3 Инфологическая модель БД и словарь данных 50](#_Toc85012763)

[2.4 Физическая модель 54](#_Toc85012764)

[2.5 Распределение прав пользователей на сервере БД 54](#_Toc85012765)

[2.6 Проектирование интерфейса мобильного приложения 55](#_Toc85012766)

[2.7 Анализ, обоснование и выбор метода тестирования ПО 60](#_Toc85012767)

[ЗАКЛЮЧЕНИЕ 62](#_Toc85012768)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 64](#_Toc85012769)

[ПРИЛОЖЕНИЕ 1 66](#_Toc85012770)

[ПРИЛОЖЕНИЕ 2 89](#_Toc85012773)

# **1.ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ**

## **1.1 Описание предметной области**

Основной целью курсового проекта является проектирование базы данных для ПО «Мотосалон «Temarov», которое автоматизирует процесс торговли мотоциклами и предоставления услуги ТО.

Актуальность темы заключается в том, что сфера мотоциклов в наше время является, как никогда востребованной, люди нуждаются в мобильном транспорте, который может проехать везде. Учитывая нестабильную ситуацию в стране, где поставки осуществляются только через параллельный импорт, а также максимально увеличиваются их сроки и цены на модели, самое время начать конкурировать с другими компаниями и заявить о себе, как востребованная организация, которая предлагает свои услуги качественно, быстро и, что самое главное, дешевле, чем у остальных. Степень важности исследования в данной ситуации и данный отрезок времени для разработки ПО является высокой, так как для увеличения производительности, то есть КПД, предприятия следует автоматизировать его, тогда и только тогда, обороты компании заметно возрастут.

«Temarov» оказывает услуги по продаже мотоциклов, исходя из характеристик модели: марки, года выпуска, типа и категории ТС, идентификационного номера, номера двигателя, шасси(рамы) и кузова, цвета, мощности двигателя, данные о бывших владельцах, если на то, поступит запрос, составляется договор купли-продажи лично менеджером по продажам мотоциклов, а также покупки б/у моделей, путем запроса у владельца необходимого перечня документов: паспорт гражданина, паспорт транспортного средства (ПТС), свидетельство, подтверждающее регистрацию, с указанием даты и места, постановка ТС на учет (СТС), а также документ, подтверждающий право собственности, чтобы снять его с учета, выполнять данную работу придется отдельному человеку, занимающему должность менеджера по выкупу мотоциклов.

Сотрудниками компании являются:

* Менеджер по выкупу. Выезжает к клиенту на зараннее обговоренное место, проводит осмотр, договаривается по цене, покупает и занимается снятием мотоцикла с учета, затем отправляет мотоцикл в салон на микро-ремонт(при необходимости).
* Помощник Менеджера по выкупу. Выезжает на место встречи вместе с Менеджером по выкупу, прямая обязанность помогать при транспортировке мотоцикла, осуществлять помощь при необходимости.
* Бухгалтер. Обязан вести внутренний контроль за учетом и отчетностью, вести налоговый учет, формировать отчеты по налогообложению, сообщать об ошибках, проводить финанализ, вести бюджетирование и управление денежными потоками, вовремя архивировать документы.
* Оператор. Принимает заказы, обрабатывает запросы. Звонит клиенту и обсуждает условия, договаривается о встречи, направляет на вызов Менеджера по выкупу или переводит звонок на Менеджера по продажам для дальнейшего консультирования клиента.
* Менеджер по продажам. Консультирует клиентов по моделям, обрабатывает запросы на покупку, составляет договоры. Выполнение ежедневных, еженедельных, месячных отчетов о проделанной работе.
* Администратор. Занимается проектированием и налаживанием работы базы данных, следит за корректностью ии безопаснотью её эксплуатации пользователями. Он должен обеспечить бесперебойную защиту сервера и доступ пользователей к нужной информации. Администратор отвечает за защиту данных и их хранение. При сбоях в работе сервера или самой базы администратор восстанавливает утраченные данные.

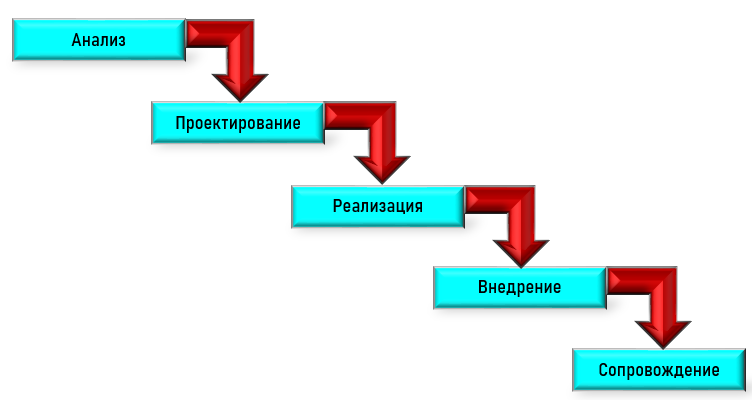
Оказание услуги купли-продажи мотоцикла, деталей, узлов и принадлежностями, технического обслуживания и ремонта мотоциклов осуществляется после оформления заявки через сайт(приложения) компании или напрямую через оператора (указанный контакт на сайте), затем реализуется консультация, после принятия решения клиента о чем-либо, выполняется обработка запроса и после направления его в определенную сферу, допустим покупки мотоцикла, происходит составление необходимых бумаг, оплата и, если то необходимо, доставка транспортного средства на базу клиента.

Вывод об организации можно сделать следующий. Компания «Temarov» осуществляет продажу мотоциклов, покупку б/у моделей, занимается продажей запчастей и производит ремонт техники. Границы корпорации расширены, следует провести автоматизацию отделов учета, продажи и покупки, а также финансовую составляющую. Следует провести огромную работу для упрощения и улучшения системы в целом. Вторым шагом для осуществления задуманного будет анализ и выбор жизненного цикла программного обеспечения.

## **1.2 Виды жизненных циклов программного обеспечения**

Жизненный цикл программного обеспечения – ряд событий, происходящих с системой в процессе её создания и дальнейшего использования. Говоря другими словами, это время от начального момента создания какого-либо программного продукта, до конца его разработки и внедрения. Жизненный цикл программного обеспечения можно представить в виде моделей.

Рассмотрим каскадную модель ЖЦ. Данная модель предусматривает последовательную организацию работ. Демонстрируется классический подход к разработке различных систем из любой ПО. В раннее существовавших ИС каждое приложение представляет собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков. На рисунке 1.1 представлена касакадная модель жизненного цикла.



**Рисунок 1.1** – Каскадная модель жизненного цикла

Первый этап разработки каскадной модели предполагает анализ – исследование проблемы, четкая формулировка требований заказчика. Результат – техническое задание (задание на разработку), согласованное со всеми заинтересованными сторонами.

Второй этап, проектирование – разработка проектных решений, удовлетворяющих всем требованиям технического задания. Результат – комплект проектной документации, содержащей все необходимые данные для реализации проекта.

Третий этап включает в себя реализацию –программирование в соответствии с проектными решениями, полученными на предыдущем этапе. Методы, используемые для реализации, не имеют принципиального значения. На этом этапе также производится проверка ПО на предмет соответствия требованиям технического задания, выявление скрытых недостатков, проявляющихся в реальных условиях работы ИС. Результат – готовый программный продукт.

Четвёртый этап учитывает внедрение – сдача готового проекта. Главная задача – убедить заказчика, что все его требования реализованы в полной мере.

Пятый этап основывается на сопровождении – в ходе сопровождения в программу вносятся изменения с тем, чтобы исправить обнаруженные в процессе использования дефекты и недоработки (первичное сопровождение), а также для добавления новой функциональности, с целью повысить удобство использования и применимость ПО (зрелое сопровождение).

Каскадная модель имеет следующие преимущества: Проста и понятна заказчикам, так как часто используется другими организациями для отслеживания проектов, не связанных с разработкой ПО Проста и удобна в применении:

* процесс разработки выполняется поэтапно;
* ее структурой может руководствоваться даже слабо подготовленный в техническом плане или - неопытный персонал;
* она способствует осуществлению строгого контроля менеджмента проекта;
* каждую стадию могут выполнять независимые команды (все документировано);
* позволяет достаточно точно планировать сроки и затраты.

Положительные стороны каскадного подхода заключаются в следующем:

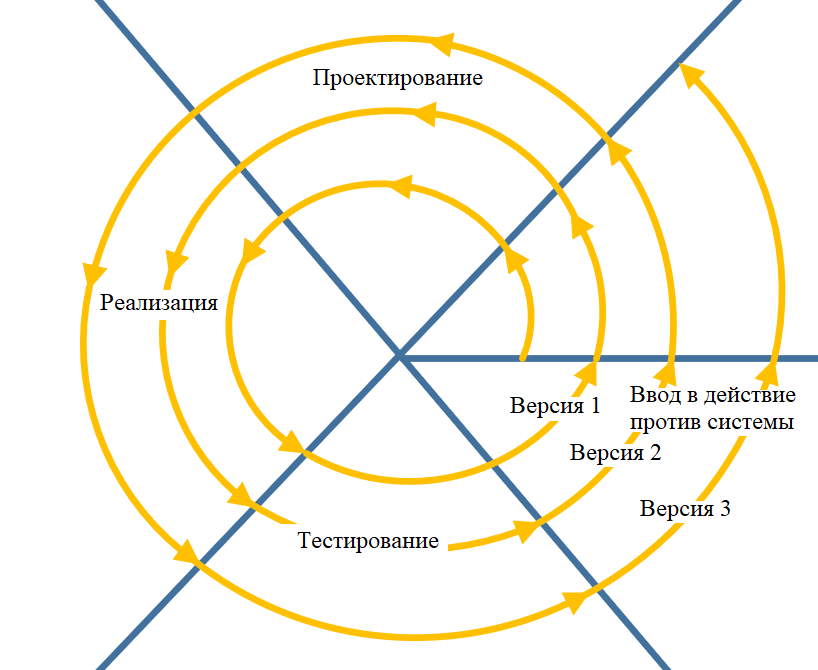
* на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
* выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты;
* в самом начале разработки достаточно точно и полно сформулировать все требования с тем, чтобы предоставить разработчикам свободу в выборе реализации, наилучшей с технической точки зрения.

Недостатки каскадной модели связаны с тем, что реальный процесс создания ИС никогда не укладывается в такую жесткую схему. Недостатки каскадной модели особо остро проявляются в случае, когда трудно (или невозможно) сформулировать требования или требования могут меняться в процессе выполнения проекта. В этом случае разработка ПО имеет принципиально циклический характер. Основные недостатки:

* существенная задержка получения результатов;
* ошибки и недоработки на любом этапе выясняются, как правило, на последующих этапах работы, что приводит к необходимости возврата на предыдущие стадии;
* сложность распараллеливания работ по проекту;
* чрезмерная информационная перенасыщенность каждого из этапов;
* сложность управления проектом;
* высокий уровень риска и ненадежность инвестиций.

Применимость между тем, каскадная модель не утратила своей актуальности при решении следующих типов задач: Требования и их реализация максимально четко определены и понятны; используется неизменяемое определение продукта и вполне понятные технические методики. Это задачи типа:

* научно-вычислительного характера (пакеты и библиотеки научных программ типа расчета несущих конструкций зданий, мостов, …),
* операционные системы и компиляторы,
* системы реального времени управления конкретными объектами. Кроме того, каскадная модель применима в условиях:
* повторная разработка типового продукта (автоматизированного бухгалтерского учета, начисления зарплаты, …);
* выпуск новой версии уже существующего продукта, если вносимые изменения вполне определены и управляемы (перенос уже существующего продукта на новую платформу). Принципы каскадной модели находят применение как элементы моделей других типов, о чем речь пойдет ниже.
* Рассмотрим спиральную модель ЖЦ предлагает итерационный процесс разработки ИС. Основной упор делается на начальные этапы ЖЦ – анализ и проектирование, так как именно здесь проверяется и обосновывается реализуемость технических решений путем создания прототипов.



**Рисунок 1.2** – Спиральная модель жизненного цикла

Схема работы спиральной модели выглядит следующим образом. Разработка вариантов продукта представляется как набор циклов раскручивающейся спирали. Каждому циклу спирали соответствует такое же 16 количество стадий, как и в модели каскадного процесса. При этом начальные стадии, связанные с анализом и планированием, представлены более подробно с добавлением новых элементов. В каждом цикле выделяются четыре базовые фазы:

* определение целей, альтернативных вариантов и ограничений,
* оценка альтернативных вариантов, идентификация и разрешение рисков,
* разработка продукта следующего уровня,
* планирование следующей фазы.

«Раскручивание» проекта начинается с анализа общей постановки задачи на разработку ПО. Здесь на первой фазе определяются общие цели, устанавливаются предварительные ограничения, определяются возможные альтернативы подходов к решению задачи. Далее проводится оценка подходов, устанавливаются их риски. На шаге разработки создается концепция (видение) продукта и путей его создания. Следующий цикл – разработка проекта – начинается с планирования разработки. На фазе определения целей устанавливаются ограничения проекта (по срокам, объему финансирования, ресурсам и т.д.), определяются альтернативы проектирования, связанные с альтернативами требований, применяемыми технологиями проектирования, привлечением субподрядчиков. На фазе оценки альтернатив устанавливаются риски вариантов и делается выбор варианта для дальнейшей реализации. На фазе разработки выполняется проектирование и создается демо-версия, отражающая основные проектные решения. Следующий цикл –реализация ПО – также начинается с планирования. Альтернативными вариантами реализации могут быть применяемые технологии реализации, привлекаемые ресурсы. Оценка альтернатив и связанных с ними рисков на этом цикле определяется степенью «отработанности» технологий и «качеством» имеющихся ресурсов. Фаза разработки выполняется по каскадной модели с выходом – действующим вариантом (прототипом) продукта.

Отметим некоторые особенности спиральной модели: до начала разработки ПО есть несколько полных циклов анализа требований и проектирования. Количество циклов модели (как в части анализа и проектирования, так и в части реализации) не ограничено и определяется сложностью и объемом задачи В модели предполагаются возвраты на оставленные варианты при изменении стоимости рисков. Каждая итерация представляет собой законченный цикл разработки, приводящий к выпуску внутренней или внешней версии, и которая совершенствуется от итерации к итерации, чтобы стать законченной системой. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его 17 качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации. Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований.

Основные преимущества спиральной модели:

* Существенно упрощается внесение изменений в проект при изменении требований заказчика;
* Отдельные элементы ИС интегрируются в единое целое постепенно, причем, процесс интеграции производится фактически непрерывно;
* Уменьшение уровня рисков, т.к. риски обнаруживаются именно в процессе интеграции. Риск максимален в начале разработки проекта, по мере продвижения он уменьшается;
* Гибкость в управлении проектом. Например, можно сократить сроки разработки за счет уменьшения функциональности системы или использовать в качестве составных частей вместо своих разработок продукцию других фирм;
* Итерационный подход упрощает повторное использование компонентов. Анализ проекта после проведения нескольких начальных итераций позволяет выявить общие, многократно используемые компоненты, которые на последующих итерациях будут совершенствоваться;
* Спиральная модель позволяет получить более надежную и устойчивую систему, т.к. ошибки и слабые места обнаруживаются и исправляются на каждой итерации.

Основные недостатки спиральной модели связаны с ее сложностью:

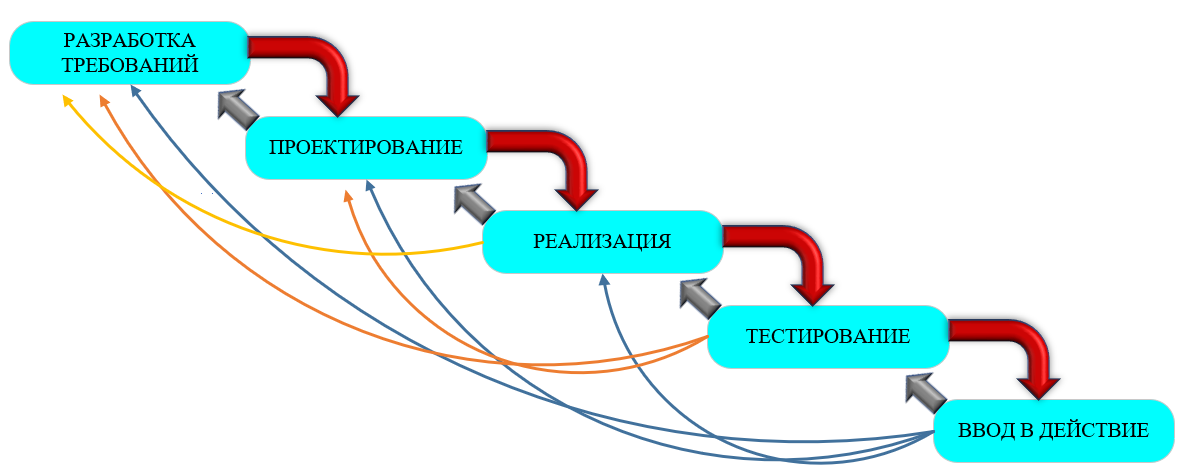
* слобжность анализа и оценки рисков при выборе вариантов;
* сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий);
* сложность оценки точки перехода на следующий цикл;
* бесконечность модели – на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки.

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Применимость. Спиральную модель целесообразно применять при следующих условиях:

* когда пользователи не уверены в своих потребностях или когда требования слишком сложны и могут меняться в процессе выполнения проекта и необходимо прототипирование для анализа и оценки требований;
* когда достижение успеха не гарантировано и необходима оценка рисков продолжения проекта;
* когда проект является сложным, дорогостоящим и обоснование его финансирования возможно только в процессе его выполнения;
* когда речь идет о применении новых технологий, что связано с риском их освоения и достижения ожидаемого результата;
* при выполнении очень больших проектов, которые в силу ограниченности ресурсов можно делать только по частям.

Рассмотрим Итерационную модель(70-80-ых годов XX в.) которая соответствует технологии проектирования «снизу — вверх». Допускает итерационные возвраты на предыдущие этапы после выполнения очередного этапа. На рисунке 1.3 представлена итерационная модель жизненного цикла.



**Рисунок 1.3** – Итерационная модель жизненного цикла

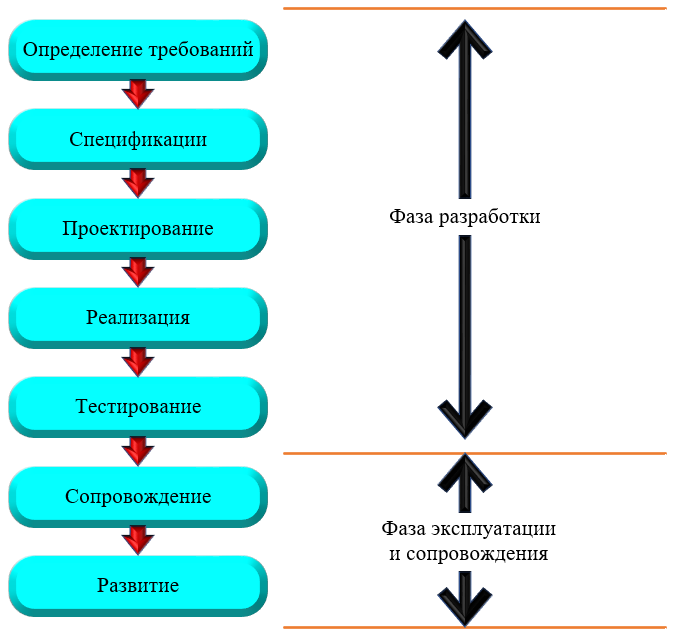
Модель предусматривает обобщение полученных проектных решений отдельных задач в общесистемные решения. При этом возникает потребность в пересмотре ранее сформулированных требований.

Непосредственным достоинством выступает - возможность оперативно вносить коррективы в проект.

Важным недостатоком является – время. При большом числе итераций растет время проектирования, возникают расхождения в проектных решениях и документации, запутывается функциональная и системная архитектура созданной ПО. Необходимость в перепроектировании старой или создании новой системы может возникнуть сразу после этапа внедрения или эксплуатации.

Рассмотрим общепринятую модель ЖЦ,вероятно, самым распространенным мотивом обращения к понятию жизненного цикла является потребность в систематизации работ в соответствии с технологическим процессом. Этому назначению хорошо соответствует так называемая общепринятая модель жизненного цикла программного обеспечения, согласно которой программные системы проходят в своем развитии две фазы:

* разработка;
* сопровождение.



**Рисунок 1.4** – Общепринятая модель жизненного цикла

Разработка начинается с идентификации потребности в новом приложении, а заканчивается передачей продукта разработки в эксплуатацию.

Первым этапом фазы разработки является постановка задачи и определение требований. Определение требований включает описание общего контекста задачи, ожидаемых функций системы и ее ограничений. На этом этапе заказчик совместно с разработчиками принимают решение о создании системы. Особенно существен этот этап для нетрадиционных приложений.

Прежде чем приступать к созданию проекта по спецификациям, они должны быть тщательно проверены на соответствие исходным целям, полноту, совместимость (непротиворечивость) и однозначность.

Разработка проектных решений, отвечающих на вопрос, как должна быть реализована система, чтобы она могла удовлетворять специфицированным требованиям, выполняется на этапе проектирования. Поскольку сложность системы в целом может быть очень большой, главной задачей этого этапа является последовательная декомпозиция системы до уровня очевидно реализуемых модулей или процедур.

На следующем этапе реализации, или кодирования каждый из этих модулей программируется на наиболее подходящем для данного приложения языке. С точки зрения автоматизации этот этап традиционно является наиболее развитым.

В рассматриваемой модели фаза разработки заканчивается этапом тестирования (автономного и комплексного) и передачей системы в эксплуатацию.

Фаза эксплуатации и сопровождения включает в себя всю деятельность по обеспечению нормального функционирования программных систем, в том числе фиксирование вскрытых во время исполнения программ ошибок, поиск их причин и исправление, повышение эксплуатационных характеристик системы, адаптацию системы к окружающей среде, а также, при необходимости, и более существенные работы по совершенствованию системы. Все это дает право говорить об эволюции системы. В связи с этим, фаза эксплуатации и сопровождения разбивается на два этапа: собственно сопровождение и развитие. В ряде случаев на данную фазу приходится большая часть средств, расходуемых в процессе жизненного цикла программного обеспечения.

Для проектирования разработки ПО, исходя из вышесказанного автором, был сделан выбор в пользу каскадной модели жизненного цикла, потому что классический метод подходит мне больше всего. Последовательное выполнение всех этапов проекта в строго фиксированном порядке. Такой подход позволит двигаться к реализации продукта эффективно, убирается гибкость и бесконечные замечания по поводу конечности продукта от заказчика, и, соответственно, модель является более чем альтернативной всем остальным, ведь она допускает делать упор в этап разработки и полностью погрузиться в проект с головой.

Для дальнейшего проектирования программного обеспечения следует провести анализированный выбор операционной системы, системы управления базой данных и языка программирования, что я и собираюсь сделать в следующей главе.

## **1.3 Анализ, обоснование и выбор ОС, СУБД и языка программирования**

Бурное развитие информационных технологий обусловило необходимость выработки новых подходов к решению проблем автоматизации деятельности различных организаций, служб, предприятий. Работа таких организаций связана с накоплением большого количества информации.

В недавнем прошлом, информацию такого рода хранили в картотеках, используя «бумажные» технологии. И, разумеется, они занимали довольно большие объёмы. Вся информация заносилась вручную на бумажные карточки, что создавало большие неудобства, затрачивалось большое количество времени и, естественно, работа такого вида сама по себе была очень кропотливой. При необходимости узнать информацию требовалась затратить большое количество времени. Сама картотека «изнашивалась» по ходу работы, то есть через некоторое время бумага могла утратить своё былое качество.

Для создания базы данных необходимо выбрать СУБД, а прикладную программу можно создать на языке программирования и манипулирования данными, а также с помощью средств быстрой разработки приложений баз данных, встроенных в СУБД, или с помощью других инструментальных средств. При выборе СУБД нужно принимать во внимание десятки факторов.

* тип модели данных, которую поддерживает данная СУБД, её адекватность потребностям рассматриваемой предметной области;
* характеристики производительности системы;
* запас функциональных возможностей для дальнейшего развития системы;
* степень оснащённости системы инструментарием для персонала администрирования данными.

К вашему вниманию на рисунке 1.5 изображен график Доли рынка операционных систем, предоставленный сервисом StatCounter Global Stats[[1]](#footnote-1).На графике представлен анализ рынка ОС, основанный на актуальности использования десктопных ОС по всему миру за 2022 год.

**Рисунок 1.5** – Доля рынка операционных систем за 2022 год

Делая заключение по данному графику, становится понятно, что наиболее используемой и популярной ОС является Microsoft Windows у которой в среднем 77% популярности, второе место занимает OS X от компании Apple у которой в среднем 17%, далее расположились операционные системы Linux и Chrome OS у которых значение популярности приблизительно по 2,4% и другие ОС занимают остаточные проценты.

**Рисунок 1.6** – Популярность версий Windows

Windows 10[[2]](#footnote-2) — операционная система для персональных компьютеров и рабочих станций, разработанная корпорацией Microsoft в рамках семейства Windows NT.

Преимущества. Windows 10, бесспорно, является инновационной системой. Но только среди ОС от Microsoft. Большинство ее «новых» функций заимствовано у систем, использующих их уже давно (Linux, MacOS X). К плюсам десятки можно отнести идеальную оптимизацию, скорость работы, время работы от аккумулятора ноутбука, наличие драйверов «из коробки» и оптимизацию работы с игровым видео.

Недостатки. Пользователи переносных компьютеров жалуются на то, что новая ОС устанавливается лишь со 2-3 попытки. Жалоб среди обладателей стационарных ПК не было.

* На Windows 10 нельзя внести в настройках свой регион. Система автоматически устанавливает США.
* Отсутствие приложения для воспроизведения DVD-дисков. Это объясняется тем, что в Америке и европейских странах необходимость в этомвиде переносных устройств отпала. Но в России и странах СНГ люди все еще пользуются компакт-дисками.
* Компания Microsoft[[3]](#footnote-3)теперь без проблем может собирать вашу личную информацию: местоположение, историю браузера и т.д. Операционная система автоматически отсылает нужную информацию и статистику пользователя прямиком в корпорацию. В Microsoft с неохотой комментируют такого рода отзывы, успокаивая людей, что все данные нужны для дальнейшего усовершенствования продукции компании. Но многие сомневаются в правдивости слов софт-гиганта.

В целом, Windows 10 является качественным продуктом. Были учтены пожелания пользователей, убраны недостатки предыдущей версии, сделаны интересные нововведения. Опытные пользователи, разбирающиеся в «компьютерной кухне» положительно оценили «десятку» и полностью на нее перешли.

Встречаются и недостатки, но их можно простить, списав на «сырость» версии. Специалисты Майкрософт обещают, что с выходом обновлений, все баги и другие проблемы будут устраняться. В будущем расширят функционал Windows 10.

Windows 11 — проприетарная операционная система для персональных компьютеров, разработанная компанией Microsoft в рамках семейства Windows NT, преемница Windows 10.

К преимуществам новой операционной системы мы отнесем:

Бесплатная активация. Подойдет ключ от Windows 8.1, 10 и выше. Те пользователи, которые в 2021 г. обновятся до Windows 11, не будут платить за лицензию. Стоимость же чистой ОС будет стартовать от 200 долларов. Поэтому бесплатное обновление к новинке – плюс к карме Майкрософт.

«Бесшовное» обновление. Установка новых пакетов обновлений не будет требовать перезагрузки системы для активации изменений. Это удобно, ведь в Windows 10 во время работы с каким-либо приложением ПК мог самостоятельно уйти в перезагрузку.

Возможен переход на другую сборку ОС без её переустановки. К примеру, у вас установлена Домашняя сборка, а при обновлении до Windows 11 вы хотите выбрать Профессиональную версию. Ввел ключ лицензии, система подгрузит файлы для Профессиональной версии и далее продолжит установку.

Удобная защита[[4]](#footnote-4) данных пользователя с помощью OneDrive. После установки ОС, на начальных этапах настройки Windows 11, можно указать создание резервной копии данных с сохранением их на виртуальный диск.

Поддержка рукописного ввода. Однако, реализация ОС для мобильных устройств и десктопных не самая лучшая идея, так как задачи для смартфонов, планшетов отличаются от тех, которые предназначены для ПК и ноутбуков. А еще нужно учитывать мощность железа. Это понимают разработчики Apple, которые для ноутбуков и смартфонов создали две разные системы. Поэтому поддержку рукописного ввода и прочих опций, которые доступны для мобильных гаджетов, отнесем для плюсов. Однако это субъективное мнение.

Интеграция с Android. Совместимых функций маловато: просмотр звонков¸ сообщений, фото, доступ к телефонной книге и файлам, их передача. Важно отметить, что Windows 10 уже имела данную функцию, а поэтому это не новинка.

Недостатки Windows 11. Учет анализа будет делаться со ставкой на рядового пользователя со средним по мощности ПК.

Прожорливость. Минимальные системные требования для установки Windows 11 подняты, хотя для офисного использования системы достаточно 4 Гб ОЗУ и 2 ядер процессора. Однако вы не сможете установить ОС, если ваш ПК имеет технические характеристики ниже, чем заявленные Майкрософт.

Несовместимость со старым железом. Многие пользователи ПК с лицензионными версиями той же Семерки, не смогут обновиться к Windows 11 даже при наличии 8 Гб ОЗУ и мощного SSD диска. Наличие TPM решает все. Windows 11 будет работать на устройствах с процессорами Intel старше 2008 г. выпуска, а вот с AMD дела обстоят иначе. Даже с процессорами, построенными на архитектуре Zen 2.0 и выше, могут возникнуть проблемы. Поэтому перед инсталляцией Windows 11 нужно будет проверить устройство на совместимость с новинкой, так как реализация TPM 2.0 осуществлена двумя способами: отдельным блоком и частью системной логики, а это нужно узнать.

Из контекстного меню правой кнопки мыши пропали функции «Копировать» и «Отправить». Из панели инструментом Проводника также пропали кнопки «Копировать», «Вставить». Пока что комбинации клавиши Win+C и т д работают. Поэтому учите сочетание клавиш.

Ранее, чтобы выполнить какое-то действие, тонкую настройку системы, достаточно было обратиться к «Панели управления» или «Параметрам» системы. В Windows 11 нужно обратиться к «Параметрам», а далее «Панели управления», пройти по нескольким уровням данных разделов. Прятать настройки – не самая лучшая идея. Однако Майкрософт решилась на этот шаг.

Компания Майкрософт[[5]](#footnote-5) уже объявила, что Windows 11 в отличие от Windows 10 не будет получать мелких обновлений, а только одно крупное обновление раз в год. Вот, когда вы привыкните к расположению определенных опций в «Параметрах» или «Панели управления», они могут переехать в другой пункт. Такая миграция предусмотрена сменой версии в рамках одной сборки Windows 11.

Телеметрия в Windows 10 считалась самой жесткой. Однако после выпуска Windows 11 ситуация изменилась. Компания Майкрософт и Google упорно работают над искусственным интеллектом и собирают данные о своих пользователях. Если данные пользователя попадут в руки данным компаниям, то они могут быть использованы не в пользу юзеров Windows 11. Телеметрия в Windows 10 со сложностью, но отключается или контролируется. В Windows 11 изменение некоторых настроек невозможно.

Многими программами Windows 11 распознается как Windows 10, поэтому создаётся впечатление, что это Windows 10, получившая крупный апдейт.

Меню Пуск. Вы уже могли привыкнуть к меню Windows 10. Однако в Windows 11 оно отличается тем, что некоторые функции на начальном экране зафиксированы, а смысл представлять новую ОС со старыми функциям и тем более, лишать пользователя возможности добавлять или убирать элементы с меню Пуск, поскольку количество виджетов фиксировано.

Из всего вышеперечисленного можно сделать вывод о том, что самоый удобной, простой, а самое главное популярной ОС для разработки ПО является Windows 10, поэтому следует сделать выбор в пользу данной операционной системы.

Следующим этапом разработки является анализ и выбор СУБД.

Система управления базами данных (СУБД)[[6]](#footnote-6) — это программное обеспечение, которое взаимодействует с конечными пользователями, приложениями и самой базой данных для сбора и анализа данных.

Есть несколько СУБД на выбор. Нужно разобрать их достоинства и недостатки, чтобы выбрать всего одну.

В ходе анализа приведем подробную статистику популярнейших СУБД среди разработчиков. На рисунке 1.7 представлена подробная диаграмма статистики актуальности использования СУБД предосатвленная сервисом Stack Overflow[[7]](#footnote-7).

**Рисунок 1.7** – Диаграмма актуальных СУБД

Самой популярной и, одновременно, актуальной является СУБД Oracle MySQL - свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB.

К достоинствам СУБД относятся:

* полностью бесплатная СУБД
* поддерживается большинством CMS
* неограниченный многопользовательский режим
* множество плагинов, облегчающих работу с данной СУБД
* поддерживает различные типы таблиц (MyISAM, InnoDB, HEAP, MERGE)
* позволяет добавлять до 50 миллионов строк в таблицы

К недостаткам причисляется:

* Недостаточная надежность.
* В вопросах надежности некоторых процессов по работе с данными (например, связь, транзакции, аудит) MySQL уступает некоторым другим СУБД.
* Низкая скорость разработки.

На следующей строчке расположилась СУБД PostgreSQL 15.0. PostgreSQL 15.0, также известная как Postgres, представляет собой бесплатную систему управления реляционными базами данных с открытым исходным кодом (СУБД), подчеркивающую расширяемость и соответствие SQL. Первоначально он был назван POSTGRES, ссылаясь на свое происхождение как преемника базы данных Ingres, разработанной в Калифорнийском университете в Беркли. В 1996 году проект был переименован в PostgreSQL 15.0, чтобы отразить его поддержку SQL. После проверки в 2007 году команда разработчиков решила сохранить название PostgreSQL 15.0 и псевдоним Postgres.

Веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД PostgreSQL 15.0. phpPgAdmin позволяет через браузер осуществлять администрирование сервера PostgreSQL 15.0 запускать команды SQL и просматривать содержимое таблиц и баз данных.

Рассмотрим следующие преимущества PostgreSQL 15.0:

* PostgreSQL 15.0 — объектно-реляционная СУБД. Это значит, что она поддерживает и объектный, и реляционный подход;
* Еще одна особенность PostgreSQL 15.0 — поддержка большого количества типов записи информации. Это не только стандартные целочисленные значения, числа с плавающей точкой, строки и булевы значения, но и денежный, геометрический, перечисляемый, бинарный и другие типы. PostgreSQL «из коробки» поддерживает битовые строки и сетевые адреса, массивы данных, в том числе многомерные, композитные типы и другие сложные структуры. В ней есть поддержка XML, JSON и NoSQL-баз;
* Работа с большими объемами. В большинстве СУБД, рассчитанных на средние и небольшие проекты, есть ограничения по объему базы и количеству записей в ней. В PostgreSQL 15.0 ограничений нет;
* PostgreSQL 15.0 работает со сложными, составными запросами. Система справляется с задачами разбора и выполнения трудоемких операций, которые подразумевают и чтение, и запись, и валидацию одновременно. Она медленнее аналогов, если речь заходит только о чтении, но в других аспектах превосходит конкурентов;
* Важная особенность PostgreSQL 15.0 — возможность одновременного доступа к базе с нескольких устройств. В СУБД реализована клиент-серверная архитектура, когда база данных хранится на сервере, а доступ к ней осуществляется с клиентских компьютеров. Так, например, реализуются разнообразные сайты. Одна из возможных сложностей — ситуация, когда несколько человек одновременно модифицируют базу и нужно избежать конфликтов;
* PostgreSQL 15.0 соблюдает требования ACID благодаря технологии MVCC. Это делает систему надежной и безопасной в использовании, а данные — защищенными от возможных сбоев, ошибок и потерь;
* Разработчик может написать для СУБД собственные типы и их преобразования, операции и функции, ограничения и индексы, собственный процедурный язык для запросов. PostgreSQL 15.0 можно модифицировать практически под любую нестандартную задачу.
* PostgreSQL 15.0 — возможно, единственная бесплатная СУБД с открытым исходным кодом, которая рассчитана на работу с объемными и сложными проектами. Она мощная, производительная, способна эффективно работать с большими массивами данных.
* PostgreSQL 15.0 — проект, который известен высоким качеством отладки. Каждая версия системы появляется в доступе только после полной проверки, поэтому СУБД очень стабильна. Частая проблема бесплатных проектов — новые версии с большим количеством багов, но в случае с PostgreSQL такой проблемы нет.
* Чаще всего PostgreSQL 15.0 используют на серверах с операционными системами семейства Linux, но СУБД поддерживает и другие ОС. Ее можно установить в системы на базе Windows, BSD, macOS и Solaris. Кроме того, у PostgreSQL 15.0 есть автономный веб-сервер PostgREST, с которым можно работать с помощью REST API. СУБД можно развернуть и в облаке.

Рассмотрим следующие недостатки PostgreSQL 15.0:

* Несмотря на высокую производительность в сравнении с другими ОРСУБД, для определенных задач, где критически важна скорость, иногда лучше выбрать другую СУБД. Так, например, если большая часть операций будет на чтение, то MySQL Database при правильных настройках окажется быстрее.
* По большому счету сложно выискивать недостатки у такой быстрой и эффективной объектно-реляционной СУБД, как PostgreSQL 15.0, которая при этом совершенно бесплатна. Недостатки могут проявиться только в случае необдуманного использования: к примеру, простые задачи можно решить эффективнее с помощью более простых инструментов.

На третьем месте расположилась СУБД Microsoft SQL Server 2019. Система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов - Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Основными достоинствами являются:

* Масштабирование системы. Взаимодействовать с ней можно как на простых ноутбуках, так и на ПК с мощным процессором, который способен обрабатывать большой объем запросов.
* Размер страниц – до 8 Кб. Данные извлекаются быстро, а сложную информацию удобнее хранить. Система обрабатывает транзакции в интерактивном режиме, есть динамическая блокировка;
* Автоматизация рутинных административных задач. Например, управление блокировками и памятью, редактура размеров файлов. В программе продуманы настройки, можно создавать профили пользователей;
* Удобный поиск. Его можно осуществлять по фразам, словам, тексту либо создавать ключевые индексы;
* Поддержка работы с другими решениями Майкрософт, в том числе с Excel, Access.

Основными минусами являются:

* Зависимость от ОС. Система работает только с Windows;
* Высокая стоимость.

На четвертом месте расположилась СУБД MongoDB 6.0. Документоориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров NoSQL-систем, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Применяется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MEAN.

Рассмотрим следующие достоинства MongoDB 6.0:

* Динамическая схема: эта СУБД позволяет гибко работать со схемой данных без необходимости изменять сами данные;
* Масштабируемость: MongoDB горизонтально масштабируема, что позволяет легко уменьшить нагрузку на сервера при больших объёмах данных;
* Удобство в управлении: СУБД не нуждается в отдельном администраторе базы данных.

Рассмотрим некоторые недостатки данной СУБД:

* Отсутствует оператор "join". Обычно данные могут быть организованы более денормализованным способом, но на разработчиков ложится дополнительная нагрузка по обеспечению непротиворечивости данных;
* Нет такого понятия, как "транзакция". Атомарность гарантируется только на уровне целого документа, т.е. частичное обновление документа произойти не может;
* Отсутствует понятие "изоляции". Любые данные, которые считываются одним клиентом, могут параллельно изменяться другим клиентом;
* Менее чем более стабильна, не рекомендовано использовать в биллинге;
* Требовательна к ресурсам - память и место на диске.

Проведя анализ популярнейших и актуальных СУБД, наилучшей в выборе осталась MySQL 8.0. Данная система управления базами данных является бесплатной, обладает всеми необходимыми функциями, проста в использовании и рассчитана для дальнейшей работы с .NET фреймворками. Также к СУБД прилагается case-средство, которое применяется для удобного использования самой БД, которое, в свою очередь, обладает удобным интерфейсом для взаимодействия с серверной частью.

Полный разбор показал, что у каждой СУБД есть свои достоинства и недостатки, однако средство для удобной работы показалось более привлекательной для MySQL 8.0.

Для приведенного анализа первой СУБД выступала MySQL 8.0. У данной системы присутствует свое программное обеспечение MySQL 8.0 Workbench. Данное ПО отлично предусмотренно для администрирования и проектирования баз данных. Интерфейс рассчитан на самого базового пользователя, поэтому он является очень удобным и простым в использовании.

Выявленные достоинства проектирования:

* Удобство интерфейса рассчитанная на базового пользователя;
* Простота использования редактора;
* Не требует подключение к серверу и сети.

Также выявленные недостатки проектирования:

* Слабая развитость буфера обмена;
* Нет конструктора запросов, поэтому их приходится писать вручную.

Вторым программным обеспечением для СУБД PostreSQL 15.0 было рассмотрено phpPgAdmin – веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД PostgreSQL 15.0. phpPgAdmin позволяет через браузер осуществлять администрирование сервера PostgreSQL 15.0, запускать команды SQL и просматривать содержимое таблиц и баз данных.

Из достоинств phpPgAdmin следует отметить:

* Особенность запускать непосредственно на сервере, что является очень удобными средством при работе на хостинге.

Из недостатков phpPgAdmin отмечаем:

* прекратил развиваться несколько лет назад;
* об ошибке узнаешь только при попытке сохранить сделанные изменения.

Третьей в нашем описании СУБД была Microsoft SQL Server, у которой также имеется case-средство и им является SQL Server Management Studio. SQL Server Management Studio - утилита из Microsoft SQL Server 2005 и более поздних версий для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server 2019. Утилита включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера.

Рассмотрим следующие преимущества Microsoft SQL Server 2019:

* Тесная интеграция с операционной системой Windows;
* Высокая производительность;
* Отказоустойчивость;
* Поддержка многопользовательской среды;
* Расширенные функции резервирования данных;
* Работа с удаленным подключением;
* Каждый выпуск включает в себя несколько специализированных редакций;
* Это снижает сложность внедрения и затраты на процесс разработки собственных решений, адаптированных для «узких» задач.
* В систему интегрирован сервер интерактивного анализа для принятия решений, создания корпоративных отчетов;
* Есть службы преобразования информации;
* Запросы можно формулировать на английском языке, без программирования;
* СУБД поддерживает работу с другими продуктами Microsoft: Access, MS Excel.

При анализе СУБД были выявлены следующие недостатки:

* Важным недостатком является зависимость от операционной среды: СУБД работает только с системой Windows;
* Высокая цена программы;
* Проблемы работы с интерфейсом (простые опции нужно разрешать в настройках).

Четвертой в рассмотрении СУБД была MongoDB, у которой нет центральной программного обеспечения, а присутствуют только отдельные инструменты.

Из всего вышесказанного, можно сделать вывод о том, что выбором СУБД стала mySQL с case-средством Workbench для неё. Теперь следует выбрать удобный и простой язык программирования, а также среду разработки для него.

На рисунке 1.8 представлена статистика актуальных данных используемых языков программирования. Информация для анализа основных языков JavaScript, Java, C#, Python, PHP и TypeScript предоставлена сайтом habr.com[[8]](#footnote-8).

**Рисунок 1.8** – Гистограмма актуальных языков программирования для разработки мобильных и веб-сайтов приложений

Для начала проведем разбор схожих по названию языков Java и JavaScript. Это просто разные языки, которые используются для разных целей. Похожие названия — просто историческое недоразумение.

Java — это серьёзный объектно ориентированный язык, на котором пишут серверный софт, программы для компьютеров и мобильные приложения. Он интересен тем, что один и тот же код можно скомпилировать под множество разных платформ. Java — один из основных языков для разработки под Android.

Рассмотрим главные преимущества языка Java:

* Кроссплатформенность;
* Написанный вами код будет работать на любой платформе с поддержкой Java;
* Надёжность. Этим качеством Java обязан строгой статической типизации. Любая переменная или выражение имеют определённый тип на момент компиляции, что сводит к минимуму вероятность ошибки.

Рассмотрим главные недостатки Java:

* Более низкая скорость, если сравнивать с C и C++. Как и у других языков высокого уровня, низкая производительность Java связана с компиляцией и абстракцией с помощью виртуальной машины.

JavaScript — это язык для управления веб-страницами в браузере. Он исполняется только в браузере, может красить элементы на странице в разные цвета, рисовать интерфейсы, пуляться данными, но всё — в браузере. Своего рода, JavaScript является начинкой в сфере разработки сайтов, то есть у нас есть статическая основа Html, далее добавляем окраску и дизайн нашему сайту это Css, и вот после этого, мы смело можем наполнить этот сайт разными функциями, тут мы и обращаемся к языку JavaScript.

Рассмотрим главные достоинства:

У джаваскрипта можно выделить три главных достоинства, среди которых, например, доступность (изучив основы JavaScript, вы сможете без труда понять большинство из них и всегда повысить свою квалификацию); повсеместная поддержка (они поддерживаются практически в любых компьютерных средах-любой современный компьютер способен на это), а также полная интеграция с браузером, благодаря которой javascript становится еще более удобным и простым в использовании.

Рассмотрим главные недостатки:

* Отсутствие чтения и загрузки файлов. Это сделано из соображения безопасности для пользователей;
* Доступен для злоумышленников. Весьма легко встроить какой-либо вредоносный код, который может нанести большой урон;
* Динамическая типизация;
* Причина частых проблем при разработке – нет возможности выявить ошибки заранее, только на этапе работы.

Вторым по популярности и значимости стал язык C# — это язык программирования, разработанный компанией Microsoft, чтобы создавать приложения для Windows. Сегодня C# стал кроссплатформенным, на нем можно писать программы как для Windows, так и для iOS и Linux. Язык используется в backend-разработке веб-приложений: в работе с базами данных и сетью, в программировании логики и так далее. Программисты C# востребованы в банках, небольших компаниях и стартапах, мобильной разработке.

У C# выделяют много преимуществ:

* Поддержка подавляющего большинства продуктов Microsoft;
* Бесплатность ряда инструментов для небольших компаний и некоторых индивидуальных разработчиков — Visual Studio, облако Azure, Windows Server, Parallels Desktop для Mac Pro и др.;
* Типы данных имеют фиксированный размер (32-битный int и 64-битный long), что повышает «мобильность» языка и упрощает программирование, так как вы всегда знаете точно, с чем вы имеете дело;
* Автоматическая «сборка мусора» Это значит, что нам в большинстве случаев не придётся заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память;
* Большое количество «синтаксического «сахара» — специальных конструкций, разработанных для понимания и написания кода. Они не имеют значения при компиляции;
* Низкий порог вхождения. Синтаксис C# имеет много схожего с другими языками программирования, благодаря чему облегчается переход для программистов. Язык C# часто признают наиболее понятным и подходящим для новичков;
* С помощью Xamarin на C# можно писать программы и приложения для таких операционных систем, как iOS, Android, MacOS и Linux.

Но есть у C# и некоторые недостатки:

* Приоритетная ориентированность на платформу Windows;
* Язык бесплатен только для небольших фирм, индивидуальных программистов, стартапов и учащихся . Крупной компании покупка лицензионной версии этого языка обойдётся в круглую сумму.

Xamarin — это платформа с открытым исходным кодом, предназначенная для построения современных производительных приложений для iOS, Android и Windows с .NET.

Платформа Xamarin представляет собой уровень абстракции, который обеспечивает управление взаимодействием между общим кодом и кодом базовой платформы. Является фреймворком для разработки мобильного приложения на C#.

Четвертое место в топе языков программирования занимает python -высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов.

Django - Свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Выявлены главные достоинства Django:

* Django как фреймворк задаёт структуру проекта. Она помогает разработчикам понимать, где и как добавлять новую функциональность;
* Благодаря одинаковой для всех проектов структуре гораздо проще найти уже готовые решения или получить помощь от сообщества. Огромное количество увлеченных разработчиков поможет справиться с любой задачей гораздо быстрее;
* Приложения в Django позволяют разделить проект на несколько частей. Приложения устанавливаются путём добавления в settings.INSTALLED\_APPS. Этот подход позволяет легко интегрировать готовые решения;
* Сотни универсальных модулей и приложений очень сильно ускорят разработку;
* Django безопасен из коробки и включает механизмы предотвращения распространенных атак вроде SQL-инъекций (XSS) и подделки межсайтовых запросов (CSRF). Подробнее об этом можно почитать в официальном руководстве по безопасности;
* Django REST Framework, который часто сокращают до «DRF», является библиотекой для построения API. Он имеет модульную и настраиваемую архитектуру, которая хорошо работает для создания как простых, так и сложных API;
* В DRF политики аутентификации и разрешений доступны из коробки. Он поставляется с базовыми классами для CRUD операций и встроенной утилитой для тестирования разрабатываемого API;
* Большие REST API часто требуют большого количества запросов для получения всех необходимых данных. GraphQL — это язык запросов, который позволяет обмениваться связанными данными гораздо проще. Подробнее почитать про основные концепции GraphQL можно в официальной документации;
* Graphene-Django позволит легко добавить соответствующую функциональность в ваш проект. Модели, формы, аутентификация, политики разрешений и другие функциональные возможности Django можно использовать для создания GraphQL API. Библиотека так же поставляется с утилитой для тестирования результата.

Рассмотрим главные недостатки Django:

* Django ORM основан на шаблоне Active Record, который хуже, чем шаблон Unit of Work, используемый в SQLAlchemy. На практике это выражается в том, что в Django модели могут «сохранять» себя по желанию, а транзакции отключены по умолчанию. Подробнее об этом можно почитать в статье «Вещи, которые мне не нравятся в Django»;
* Django является большим и монолитным фреймворком. Это позволяет сообществу разрабатывать сотни универсальных модулей и приложений, но снижает скорость разработки самого Django. Кроме того, фреймворк должен поддерживать обратную совместимость, поэтому он развивается относительно медленно.

Kivy - бесплатный фреймворк Python с открытым исходным кодом для разработки мобильных приложений и другого прикладного программного обеспечения с поддержкой мультитач с естественным пользовательским интерфейсом (NUI). Он распространяется на условиях лицензии MIT и может работать на Android, iOS, Linux, macOS и Windows.

Рассмотрим главные достоинства Kivy:

Во-первых, скорость разработки, лаконичность кода, возможность моментально изменять и отслеживать изменения в приложениях, это возможность просто писать код в то время, когда другие отращивают бороды в безуспешных попытках окончить свое приложение или мастерят очередные сверхмодные костыли для своих проектов.

Во-вторых, это на 99.9% настоящий кроссплатформенный фреймворк, с которым вы можете быть уверенными, что ваш код, единожды написанный, запустится у будет работать на всех доступных платформах. Для справки: Xamarin — это всего лишь 60% кода, который можно переиспользовать, несмотря на заявления разработчиков о 80%. Kivy зрелый фреймворк, который разрабатывается с 2011 года, старше свого собрата React Native (2015 год) и одногодка Xamarin (2011 год).

Рассмотрим главные недостатки Kivy:

* Ненативный набор инструментов создания пользовательского интерфейса;
* Пока недостаточно широко развитый фреймворк для мобильных разработок.

Исходя из всего вышесказанного, следует сделать вывод о том, что выбор языка C# будет наиболее благоприятным. Как уже было сказано раннее, программирование для платформ Андроид осуществляется с помощью платформы Xamarin, которая позволяет делать нативные сборки под целевые платформы и создавать высокопроизводительные приложения с естественным внешним видом. С технической точки зрения, для кроссплатформенной разработки Xamarin использует единый язык C# и библиотеки, обёрнутые в слой .NET. С точки зрения работы и пользовательского опыта, такие приложения можно отнести к нативным, то есть специально разработанным под iOS и Android.

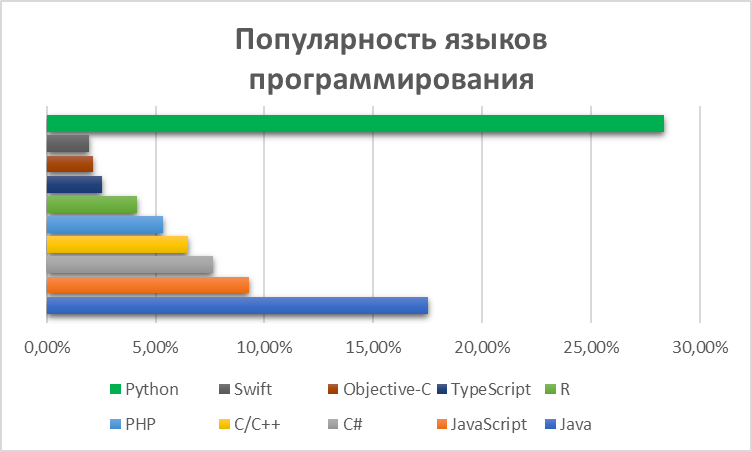
Достоинства платформы Xamarin:

* Единый стек технологий для разработки на всех платформах;
* Производительность близка к нативной;
* Нативный UI;
* Open Source-технологии с корпоративной поддержкой;
* Простая поддержка;
* Полный пакет инструментов разработки.

При расмотре были выявлены следующие недостатки платформы Xamarin:

* Задержки с обновлениями платформ;
* Ограниченный доступ к open-source библиотекам;
* Ограниченность экосистемы;
* Требование к базовым знаниям языка программирования;
* Xamarin не подходит для приложений с высокопроизводительной графикой;
* Больший размер приложений;
* Сложности с интеграцией.

Проанализировав языки в общих сферах, следует сделать вывод о том, что наиболее популярным языком из них является Python на 2022 год, а C# занимает лишь четвертое место, что вполне неплохо. Также отметим, что JavaScript и TypeScript держатся в одном диапазоне с языком C#. На рисунке 1.9 представлена сайтом habr.com[[9]](#footnote-9) диаграмма популярности языков программирования.



**Рисунок 1.9** – Популярность языков программирования

Также нам требуется определить интегрированную среду разработки, то есть место, где мы непосредственно будем писать код. И такой основной средой разработки для программирования на C# является Microsoft Visual Studio.

Visual Studio 2022 - это интегрированная среда разработки Microsoft для создания приложений для многочисленных платформ, разбег большой, начиная от Android до iOS и Windows. ASP.NET - это платформа разработки веб-приложений, в состав которой входят: веб-сервисы, программная инфраструктура, модель программирования, от компании Майкрософт. ASP.NET входит в состав платформы .NET Framework и является развитием более старой технологии Microsoft ASP. Можно также разрабатывать службы Azure (облачная платформа компании Microsoft).

Выделим основные достоинства:

* Поддержка нескольких репозиторий Git и поддержка

промежуточного хранения строк;

* Усовершенствования IntelliCode;
* Повышение производительности;
* предоставляется в 64-разрядной версии;
* Более быстрый поиск при использовании возможности "Найти в файлах";
* Средства Git быстрее;
* Создание современных приложений;
* Улучшенные средства разработки для C++ и .NET, а также средство "Горячая перезагрузка";
* Обновления для Blazor & Редакторы Razor + Горячая перезагрузка для ASP.NET.

Выделим основные недостатки:

* невысокое быстродействие из-за концепции «всё в одном»;
* Некоторые плагины (в том числе для разработки приложений для Android) имеют существенные ограничения функциональности;
* Требует много памяти.

Язык Java[[10]](#footnote-10) также имеет средства для разработки мобильных приложений, одним из них является Google Android Studio. Это интегрированная среда разработки (IDE) для работы с платформой Android, написанная на с++, java, kotlin, и анонсированная 16 мая 2013 года на конференции Google I/O. В последней версии Android studio поддерживается Android 4.1 и выше.

Рассмотрим основные достоинства:

* Среда разработки поддерживает работу с несколькими языками программирования, к которым относятся самые популярные – C/C++, Java;
* редактор кода, с которым удобно работать;
* позволяет разрабатывать приложения не только для смартфонов/планшетов, а и для портативных ПК, приставок для телевизоров Android TV, устройств Android Wear, новомодных мобильных устройств с необычным соотношением сторон экрана;
* тестирование корректности работы новых игр, утилит, их производительности на той или иной системе, происходит непосредственно в эмуляторе;
* рефакторинг уже готового кода;
* достаточно большая библиотека с готовыми шаблонами и компонентами для разработки ПО;
* разработка приложения для Android N – самой последней версии операционной системы;
* предварительная проверка уже созданного приложения на предмет ошибок в нем;
* большой набор средств инструментов для тестирования каждого элемента приложения, игры;
* для неопытных/начинающих разработчиков специально создано руководство по использованию Android Studio[[11]](#footnote-11), размещенное на официальном сайте утилиты.

Рассмотрим основные недостатки:

* Несмотря на наличие встроенного Android-эмулятора в самой среде разработки, с тестированием новоразработанного приложения могут возникнуть трудности. Так, для его запуска необходима достаточно внушительная по производительности аппаратная основа ПК, на котором планируется тестирование;
* невозможность написать серверные проекты на языке Java для ПК, Android устройств.

Интегрированной средой разработки Android Studio является JetBrain Intellij IDEA, предназначенную для написания на таких языках как PHP, JavaScript, Sass, Ruby, Python и конечно же Java.

Intellij IDEA[[12]](#footnote-12) - это одна из самых мощных универсальных сред разработки. Она поддерживает такие языки, как PHP, JavaScript, Sass, Ruby, Python и конечно же Java. Добавьте сюда поддержку популярных фреймворков и движков, тестов для кода, адаптацию под мобильные устройства, интеграцию с Git и Mercurial, инструменты для работы с серверами – и вы поймёте, что с IntelliJ IDEA программирование проще и продуктивнее.

Рассмотрим следующие достоинства:

* Мультиязычная. Поддерживает разные языки программирования. Вы сможете писать код на Java, дальше освоить Kotlin и использовать его;
* Интегрирована с Git и Github. Позволяет использовать систему контроля версий Git и отправлять проекты на Github в один клик;
* Позволяет настроить нужное окружение. Чтобы запускать тесты в IDE, достаточно докачать Java Devkit;
* Технология автодополнения IntelliSense;
* Возможность кастомизировать рабочую панель;
* Поддержка разделенного экрана (split screen).

Рассмотрим следующие недостатки:

* Из недостатков можно выделить тяжеловесность этой IDE. Для выполнения даже небольших правок могут потребоваться значительные ресурсы, поэтому если нужно выполнить какую-то простую и быструю задачу, удобнее использовать более легкий редактор.

В заключение данного раздела были проанализированы все популярные и актуальные СУБД, широко описаны их достоинства и недостатки, а также были рассмотренны case-средства к ним. Так, к выбранной СУБД Oracle MySQL[[13]](#footnote-13) было выбрано средство MySQL Workbench соответсвенно. Были исследованы необходимые модели ЖЦ, а также инструменты для разработки необходимого ПО и БД. Каскадная модель жизненного цикла была выбрана за свою простоту структуры, которой может руководствоваться малоопытный сотрудник или, в общем, весь персонал. Язык программирования на котором будет осуществляться реализация необходимого ПО выбран C# и интегрированная среда разработки, созданная практически именно для этого языка Microsoft Visual Studio. Исходя из оценки всех операционных систем, была выбрана OC Microsoft Windows 10. Теперь, закончив теоретическую часть, следует перейти к практической.

# **2. РАЗДЕЛ ПРОЕКТИРОВАНИЯ**

## **2.1 Use – case диаграмма**

Use-case диаграмма[[14]](#footnote-14) - диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Актор[[15]](#footnote-15) — это роль объекта вне системы, который прямо взаимодействует с ее частью — конкретным элементом (элементом Use Case).

Прецедент (элемент Use Case) — это описание последовательности действий (или нескольких последовательностей), которые выполняются системой и производят для отдельного актера видимый результат.

Один актор может использовать несколько элементов прецедентов, и наоборот, один прецедент может иметь несколько актеров, использующих его. Каждый прецедент задает определенный путь использования системы. Набор всех прецедентов определяет полные функциональные возможности системы.

На рисунке 2.1 представлена use-case диаграмма администратора баз данных.



**Рисунок 2.1** – Use-case диаграммма администратора баз данных

Главная цель данной диагрмма состоит в следующем: проектируемая информационная система представляется в форме так называемых вариантов использования, в которых взаимодейтвуют внешние сущности или акторы, при этом, актором или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. Это может быть что-угодно, от человека до техничего устройства или программы, которая служит в роли источника воздействия на моделируемую систему так, как определит разработчик.

Авторизация[[16]](#footnote-16) — это процедура проверки прав пользователя на выполнение определенных действий на веб-ресурсе или в компьютерной системе, результатом которой становится разрешение или отказ в осуществлении запрашиваемых операций, а также предоставление в соответствии с правами пользователю возможностей, которые гарантированы ему веб-ресурсом или компьютерной системой. На рисунке 2.2 показана use-case диаграмма возможности авторизированного и ограничения неавторизированного пользователя.



**Рисунок 2.2** – Use-case диаграмма авторизированного и неавторизированного пользоваталей

На рисунке 2.3 показана use-case диаграмма бухгалтера, выполняющего ведения бухгалтерских учетов и регистров.



**Рисунок 2.3** – Use-case диаграмма бухгалтера

На рисунке 2.4 показана use-case диаграмма Менеджера, ведущего отчет статистики, и занимающегося составлением отчетов продаж.



**Рисунок 2.4** – Use-case диаграммма менеджера

Следующим актором у нас будет изображен на рисунке 2.5 оператор, который занимается консультацией клиента по всем вопросам.



**Рисунок 2.5** – Use-case диаграмма оператора

Таким образом, мы определили все необходимые use-case диаграммы, следующим этапом у нас будет выступать проектирование непосредственно концептуальной модели базы данных.

## **2.2** **Концептуальная модель БД**

Концептуальная (содержательная) модель — это абстрактная модель, определяющая структуру моделируемой системы, свойства её элементов и причинно-следственные связи, присущие системе и существенные для достижения цели моделирования.

Концептуальная модель относится к моделям, которые формируются после процесса концептуализации или обобщения.

Она состоит из концепций, используемых для того, чтобы помочь людям узнать, понять или смоделировать предмет, который представляет модель. Это тоже набор понятий. Напротив, физические модели-это физические объекты, такие как игрушечная модель, которая может быть собрана и заставлена работать подобно объекту, который она представляет. Концептуальная модель представлена в Приложении 2.

Следующим этапом нашего проектирования станет создание инфологической модели.

## **2.3 Инфологическая модель БД и словарь данных**

Инфологическая модель[[17]](#footnote-17) представляет собой описание предметной области, основанное на анализе семантики объектов и явлений, выполненное без ориентации на использование в дальнейшем программных или технических компьютерных средств.

Инфологическое моделирование проводится на начальном этапе проектирования. Его целью является создание описания предметной области и ее однозначное понимание разработчиком и пользователем базы данных. Модель данных в этом случае служит средством документирования и общения.

Инфологическая модель представлена в Приложении 2.

Для более подробного описания БД составим словарь данных (табл. 1-16).

**Таблица 1: Заказ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Order** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdOrder | INT | y |  |
| FK | idStaff | INT | y | foreign key to  Orders from Staff |
| FK | idClient | INT | y | foreign key to  Orders from Client |
| FK | idMoto | INT | y | foreign key to  Orders from Moto |
|  | OrderDate | DATE | y |  |
| FK | idOrderStatus | INT | y | foreign key to  Orders from OrderStatus |
|  | TotalCost | INT | y |  |

**Таблица 2: Сотрудники**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Staff** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdStaff | INT | y |  |
|  | Surname | VARCHAR(45) | y |  |
|  | Name | VARCHAR(45) | y |  |
|  | Patronymic | VARCHAR(45) | n |  |
|  | DateOfBirth | DATE | y |  |
|  | Email | VARCHAR(45) | y |  |
|  | Password | INT | y |  |
|  | Phone | VARCHAR(45) | y |  |
| FK | idPost | INT | y | foreign key to  Staff from ListOfPost |
| FK | idRoles | INT | y | foreign key to  Staff from RolesInTheOrders |
| FK | idUserStatus | INT | y | foreign key to  Staff from UserStatus |

**Таблица 3: Роли в системе**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RolesInTheOrders** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdRoles | INT | y |  |
|  | RoleName | VARCHAR(45) | y |  |

**Таблица 4: Список должностей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ListOfPost** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdPost | INT | y |  |
|  | PostName | VARCHAR(45) | y |  |

**Таблица 5: Клиенты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clients** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdClient | INT | y |  |
|  | Surname | VARCHAR(45) | y |  |
|  | Name | VARCHAR(45) | y |  |
|  | Patronymic | VARCHAR(45) | n |  |
|  | DateOfBirth | DATE | y |  |
|  | Email | VARCHAR(45) | y |  |
|  | Password | INT | y |  |
|  | Phone | VARCHAR(45) |  |  |
| FK | idUserStatus | INT | y | foreign key to  Client from UserStatus |

**Таблица 6: Список услуг в заказе**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ListOfServicesInTheOrder** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdOrder | INT | y |  |
| FK | idServices | INT | y | foreign key to  ListOfServicesInTheOrder from ListOfServices |

**Таблица 7: Список услуг**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ListOfServices** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idServices | INT | y |  |
|  | NameServices | VARCHAR(45) | y |  |
|  | ShortDescription | VARCHAR(45) | y |  |
|  | Cost | FLOAT | y |  |

**Таблица 8: Статус заказа**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderStatus** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdOrderStatus | INT | y |  |
|  | OrderName | VARCHAR(45) | y |  |

**Таблица 9: Статус пользователя**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **UserStatus** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdUserStatus | INT | y |  |
|  | StatusName | VARCHAR(45) | y |  |

**Таблица 10: Марка мото**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MotoBrands** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | IdBrand | INT | y |  |
|  | BrandName | VARCHAR(45) | y |  |

**Таблица 11: Фотографии мото**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MotoPhotos** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK  FK | IdMoto | INT | y | foreign key to MotoPhotos  from Moto |
|  | File | VARCHAR(45) | n |  |

**Таблица 12: Мотоциклы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Moto** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idMoto | INT | y |  |
| FK | TechnicalSpecifications | INT | у | foreign key to Moto  from TechnicalSpecifications |
| FK | idModels | INT | у | foreign key to Moto  from Models |
| FK | idTypeOfMotorcycle | INT | y | foreign key to Moto  from TypeOfMotorcycle |

**Таблица 13: VIN номер**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VINNumber** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idVINNumber | INT | y |  |
|  | Naming | VARCHAR(45) | y |  |

**Таблица 14: Тип двигателя**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EngineType** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idEngineType | INT | y |  |
|  | Naming | VARCHAR(45) | y |  |

**Таблица 15: Модель мототоцикла**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MotoModel** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idModels | INT | y |  |
|  | Naming | VARCHAR(45) | y |  |
| FK | idBrand | INT | y | foreign key to MotoModel  from MotoBrands |
| FK | idYear | INT | y | foreign key to Moto  from ListOfReleaseYears |

**Таблица 16: Список годов выпуска**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ListOfReleaseYears** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idYear | INT | y |  |
|  | Meaning | DATE | y |  |

**Таблица 17: Пробег**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mileage** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idMileage | INT | y |  |
|  | Meaning | VARCHAR(45) | y |  |

**Таблица 18: Состояние**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Condition** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idCondition | INT | y |  |
|  | Meaning | VARCHAR(45) | y |  |

**Таблица 19: Расход топлива**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FuelConsumption** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idFuelConsumption | INT | y |  |
|  | Meaning | VARCHAR(45) | y |  |

**Таблица 20: Вид мотоцикла**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TypeOfMotorcycle** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idTypeOfMotorcycle | INT | y |  |
|  | Naming | VARCHAR(45) | y |  |

**Таблица 21: Технические характеристики**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TechnicalSpecifications** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | idTechnicalSpecifications | INT | y |  |
| FK | idTypeOfMotorcycle | INT | y | foreign key to Moto  from TireModels |
| FK | idEngineType | INT | y | foreign key to Moto  from EngineType |
| FK | idMileage | INT | y | foreign key to Moto  from Mileage |
| FK | idVINNumber | INT | y | foreign key to Moto  from VINNumber |
| FK | idCondition | INT | y | foreign key to Moto  from Condition |
| FK | idFuelConsumption | INT | y | foreign key to Moto  from FuelConsumption |

## **2.4 Физическая модель**

Физическая модель данных[[18]](#footnote-18) (или структура базы данных) - это представление структуры данных, реализованной или предназначенной для реализации в системе управления базами данных. В жизненном цикле проекта она обычно является производной от логической модели данных, хотя может быть переработана из данной реализации базы данных. Полная физическая модель данных будет включать все артефакты базы данных, необходимые для создания связей между таблицами или для достижения целей производительности, такие как индексы, определения ограничений, связывающие таблицы, секционированные таблицы или кластеры. Аналитики обычно могут использовать физическую модель данных для расчета оценок объема хранилища; она может включать конкретные сведения о распределении хранилища для данной системы баз данных.

Схема физической модели представлена в Приложении 2, работа выполнена в case-средстве MySQL.

## **2.5 Распределение прав пользователей на сервере БД**

Предусмотрено пять ролей для работы с сервером БД:

Оператор - основная его задача – консультация клиента, как авторизированного, так и неавторизированного, затем в его обязанности входит перевод на другие линии, а конкретно на линии менеджеров, и также, проведение инструктажа по получению отчета.

Бухгалтер[[19]](#footnote-19) – его непосредственной работой представляется ведение бухгалтерских учетов и регистров, то есть, экономический учет.

Пользователь – получает права на просмотр товаров.

Администратор баз данных – обладает особым правом администрирования баз данных, которое включает в себя обязанности: оптимизации производительности базы данных, обеспечение и контроль доступа к базе данных, обеспечение перехода на новую версию СУБД, обеспечение безопасности в базе данных, резервирование и восстановление базы данных, а также обеспечение целостности баз данных.

Менеджер – обладает правами учета статистики, то есть, изменение данных в отчете – добавление данных и удаление данных, и также, составление отчетов в которые входит отчеты по поступлению мотоциклов и их непосредственные продажи.

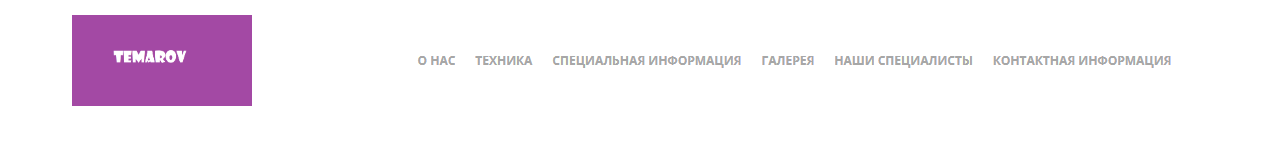
## **2.6 Проектирование интерфейса веб-сайта.**

При входе на веб-сайт вы попадаете на главную страницу. Эта страница необходима для правильной координации по всему сайту и интересующим каталогам. На рисунке 1.1 представлена главная форма страницы с названием компании и задним фоном.



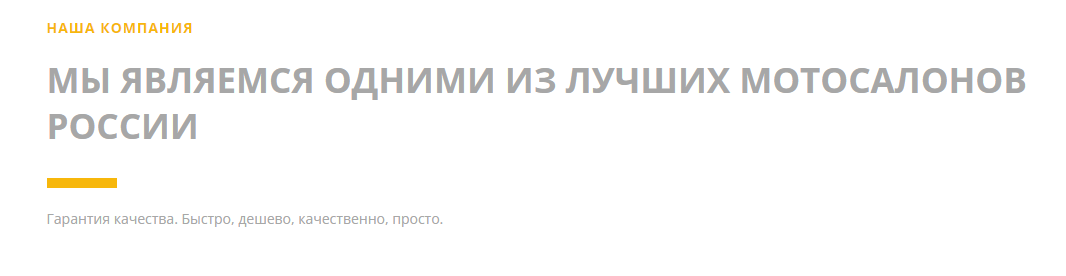
**Рисунок 1.1** – главная форма сайта

Следующим шагом для нашего серфинга по сайту будет одна из важнейших вещей – шапка, сделанная для перемещения на дополнительные страницы сайта. На ней изображены такие переходы как: о нас, техника (выбор товара), специальная информация (конткаты, акции, скидки), галерея (изображения товаров, мотосалона, автозапчастей), наши специалисты(перечень важных работников) Шапка сайта представлена на рисунке 1.2.



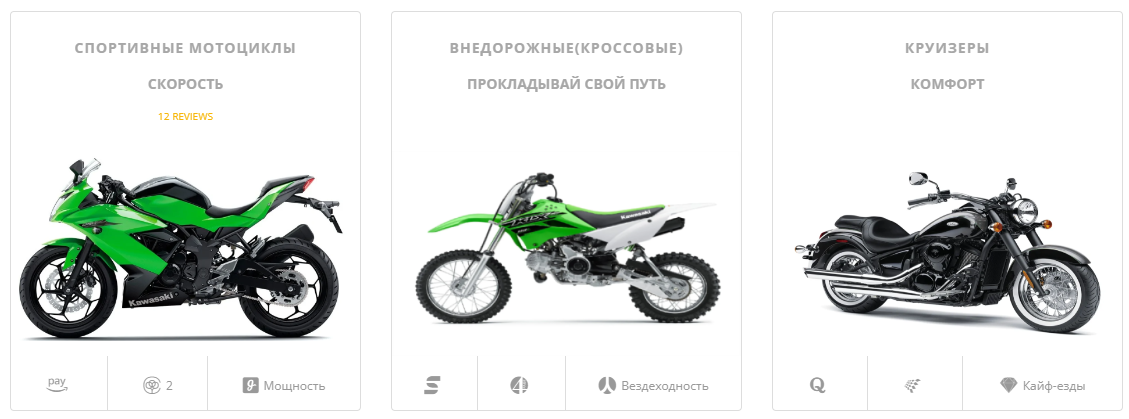
**Рисунок 1.2** – Шапка сайта

Далее мы переходим к небольшому представлению компании и её краткому описанию. Краткое описание представлено на рисунке 1.3.



**Рисунок 1.3** – Краткое описание сайта

Следующей непосредственной частью сайта является представление продаваемой техники разделенной на категории: спортивные мотоциклы, внедорожные, круизеры и их главные достоинства. Категории мотоциклов представлены на рисунке 1.4.



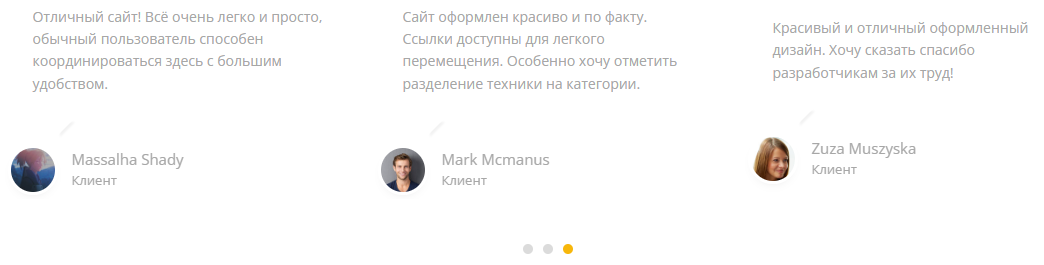
**Рисунок 1.4** – категории мотоциклов

Далее пользователь может перейти и разобрать поэтапную готовность заключения сделки для оформления мотоцикла. Эта услуга заключается, если пользователь намерен продать свою технику в мотосалон компании. Последовательность предоставления услуги представлена на рисунке – 1.5.



**Рисунок 1.5** – последовательность предоставления услуги

Последующей функцией сайта будет предоставления отзывов пользователей(клиентов), имевших место быть сделакам, заключенным с компанией мотосалоном. Примеры отзывов представлены на рисунке 1.6.

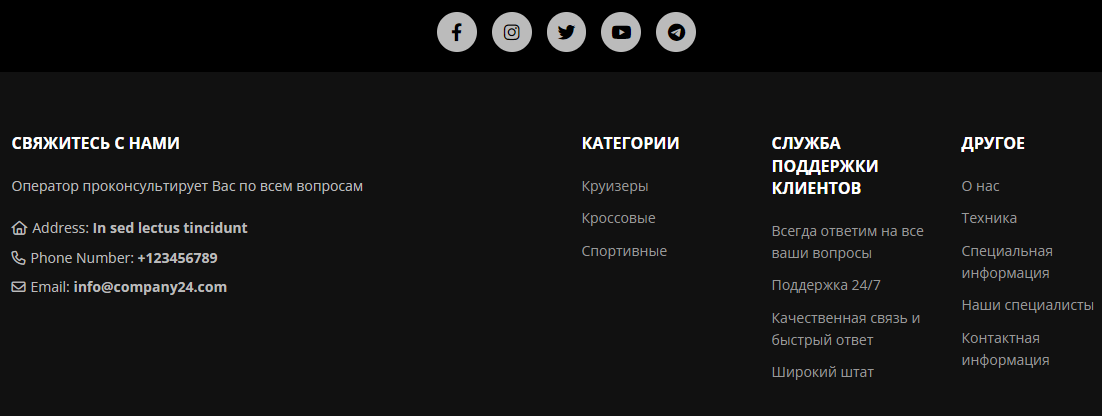


**Рисунок 1.6** – примеры отзывов

Внизу главной страницы изображен подвал. В котором присутствует четыре списка:

* Свяжитесь с нами. В списке присутствуют данные, необходимые для установления связи между клиентом и оператором. Перечислены адрес главного офиса компании, номер телефона и почта;
* Категории. В списке повторно перечислены категории мототехники представленной в мотосалоне.
* Служба поддержки клиентов. Она включает в себя онлайн-консультацию, описание поддержки, которая обеспечивает фидбек клиенту, качественная связь и быстрый ответ, а также широкий штат для возможности ответить каждому пользователю.

Подвал с представленной информацией представлен на рисунке 1.7.



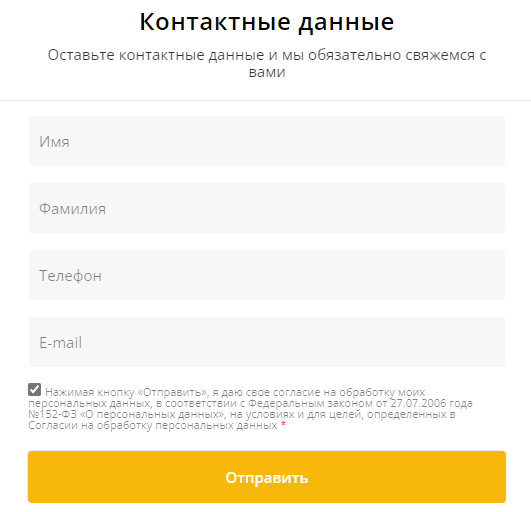
**Рисунок 1.7** – подвал

Также на сайте представлен отдельный блок в виде, переходной ссылки для прохождения регистрации, после прохождения которой, пользователю перезванивает оператор. Данная форма представлена на рисунке 1.8.



**Рисунок 1.8** – ссылочная форма для регистрации

После прохождения по данной ссылочной форме, пользователю откроется данном окне форма регистрации пользователя. Форма представлена на рисунке 1.9.



**Рисунок 1.9** – форма авторизации пользователя

## **2.7 Анализ, обоснование и выбор метода тестирования ПО**

Тестирование программного обеспечения (Software Testing)[[20]](#footnote-20) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом. Цель тестирования — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

Существует несколько методов тестирования программного обеспечения:

* Тестирование производительности (performance testing) — определение стабильности и потребления ресурсов в условиях различных сценариев использования и нагрузок.
* Нагрузочное тестирование (load testing) — определение или сбор показателей производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству).
* Тестирование масштабируемости (scalability testing) — тестирование, которое измеряет производительность сети или системы, когда количество пользовательских запросов увеличивается или уменьшается.
* Объёмное тестирование (volume testing) — это тип тестирования программного обеспечения, которое проводится для тестирования программного приложения с определенным объемом данных.
* Стрессовое тестирование (stress testing) — тип тестирования направленный для проверки, как система обращается с нарастающей нагрузкой (количеством одновременных пользователей).
* Инсталляционное тестирование (installation testing) — тестирование, направленное на проверку успешной установки и настройки, обновления или удаления приложения.
* Тестирование интерфейса (GUI/UI testing) — проверка требований к пользовательскому интерфейсу.
* Тестирование удобства использования (usability testing) — это метод тестирования, направленный на установление степени удобства использования, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.
* Тестирование локализации (localization testing) — проверка адаптации программного обеспечения для определенной аудитории в соответствии с ее культурными особенностями.
* Тестирование безопасности (security testing) — это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.
* Тестирование надёжности (reliability testing) — один из способов нефункционального тестирования ПО, целью которого является проверка работоспособности приложения при длительном тестировании с ожидаемым уровнем нагрузки.
* Регрессионное тестирование (regression testing) — тестирование уже проверенной ранее функциональности после внесения изменений в код приложения, для уверенности в том, что эти изменения не внесли ошибки в областях, которые не подверглись изменениям.
* Повторное/подтверждающее тестирование (re-testing/confirmation testing) — тестирование, во время которого исполняются тестовые сценарии, выявившие ошибки во время последнего запуска, для подтверждения успешности исправления этих ошибок.

При проведении тестирования веб приложения было выявлено, что все вкладки, представленные на сайте, работают без ошибок.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта была спроектирована БД для ПО мотосалон «Temarov», в ходе которого были выполнены все поставленные задачи, а именно:

* Сформулирована постановка задачи и исследована предметная область;
* Выбраны ОС, СУБД и язык программирования;
* Спроектирована Use-case диаграмма;
* Спроектирована концептуальная модель данных;
* Спроектирована информационно-логическая модель данных;
* Спроектирована физическая модель данных;
* Спроектирован интерфейс пользователя;
* Выполнено тестирование ПО.

Разработанная БД соответствует 3НФ, способствуя целостному хранению и обработке информации.

Реализация веб приложения была осуществлена при помощи технологии ASP.NET с использованием языковых средств программирования С#. В качестве инструментальных средств разработки была выбрана среда Microsoft Visual Studio 2022. Разработка БД была реализована в системе управления MySQL Workbench 8.0.

Задание курсового проекта выполнено в полном объеме, цель достигнута.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Бек Кент - Экстремальное программирование: разработка через тестирование.: "Издательский дом ""Питер""", 2017 – 224с.
2. Борис Новиков, Екатерина Горшкова, Основы технологий баз данных, Litres - 2019, 240 с.
3. Буч Градди Объектно-ориентированный анализ и проектирование с примерами Приложений, 3-е изд. / Буч Градди, Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер с англ. — М.: ООО «И.Д. Вильямс», 2010.
4. Гради Буч, Джеймс Рамбо, Ивар Якобсон, Язык UML. Руководство пользователя, ДМК-Пресс - 2007 г. – 496 с.
5. Иван Аптекарев, ER-модель репозитория поддержки решения задачи QSAR, LAP Lambert Academic Publishing – 2012 – 108 с.
6. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов.: Пер. с англ. М.: ДМК Пресс, 2002 – 160 с.
7. Сергей Тарасов, СУБД для программиста. Базы данных изнутри., СОЛОН-Пресс – 2015 – 322 с.
8. Эдвард Сьорье, Проектирование и реализация систем управления базами данных, 2020 – 461 с.
9. Эрик Редмонд, Джим Р. Уилсон "Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL" (2015) – 384 с.
10. Эрих Гамма Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб.: Питер, 2009. – 366 с.
11. analytics.usa.gov [Электронный ресурс] – Режим доступа: <https://analytics.usa.gov>
12. Beboss [Электронный ресурс] - Режим доступа: <https://www.beboss.ru/bplans-rent-a-special>
13. Habr [Электронный ресурс] - Режим доступа: <https://habr.com/ru/company/qiwi/blog/346438/>
14. Lucidchart [Электронный ресурс] - Режим доступа: [https://www.lucidchart.com/pages/ru](https://www.lucidchart.com/pages/ru%20)
15. MyFirstComp [Электронный ресурс] - Режим доступа: <https://myfirstcomp.ru/operation_systems/windows-10-minusy-i-plyusy-operacionnoj-sistemy/>
16. Shwanoff [Электронный ресурс] - Режим доступа: <https://shwanoff.ru>
17. Studbooks [Электронный ресурс] – Режим доступа: <https://studbooks.net/2256941/informatika/infologicheskaya_model_dannyh>
18. Systems education [Электронный ресурс] - Режим доступа: <https://systems.education/use-case>
19. Tproger [Электронный ресурс] – Режим доступа: <https://tproger.ru/articles/software-development-life-cycle/>
20. Webonto [Электронный ресурс] - Режим доступа: <https://webonto.ru/kontseptualnaya-model-bazyi-dannyih/>
21. Wikipedia [Электронный ресурс] - Режим доступа: <https://ru.wikipedia.org/wiki/Проектирование_баз_данных>
22. Wikipedia [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Database
23. Habr [Электронный ресурс] – режим доступа:https://habr.com/ru/post/651585/

# **ПРИЛОЖЕНИЕ 1**

## **Листинг пяти запросов**

1. Составной многотабличный запрос с параметром, включающий соединение таблиц и CASE-выражение

replace listofreleaseyears(idYear, Meaning)

values

(1, '2010-01-01'),

(2, '2011-01-01'),

(3, '2012-01-01'),

(4, '2013-01-01'),

(5, '2014-01-01'),

(6, '2015-01-01'),

(7, '2016-01-01'),

(8, '2017-01-01'),

(9, '2018-01-01'),

(10,'2019-01-01'),

(11,'2020-01-01'),

(12,'2021-01-01'),

(13,'2022-01-01');

replace motobrands(idBrand,BrandName)

values

(1, 'Suzuki'),

(2, 'BMW'),

(3, 'Kawasaki'),

(4, 'Yamaha');

replace motomodel(idModels, Naming, idBrand, idYear)

values

(1, "S 1000 R", 2, 13),

(2, "F 900 R", 2, 13),

(3, "R NINE T", 2, 13),

(4, "C 400 GT", 2, 13),

(5, "F 850 GS", 2, 13),

(6, "NINJA 125",3, 10),

(7, "KLX230R", 3, 11),

(8, "W175", 3, 9),

(9, "PULSAR 200NS", 3, 8),

(10, "KX112", 3, 13),

(11, "NINJA Z H2", 3, 13),

(12, "XSR 125", 4, 12),

(13, "CRYPTON", 4, 13),

(14, 'R125', 4, 12),

(15, "R1M", 4, 12),

(16, "NEO 125 UBS", 4, 13),

(17, "MIO GEAR", 4, 13),

(18, "MIO GRAVIS", 4, 13),

(19, "GSX-S950", 1, 13),

(20, "GSX-R150", 1, 13),

(21, "ECSTAR GSX-RR", 1, 12),

(22, "AVENIS", 1, 13),

(23, "INTRUDER 150", 1, 10);

use `motorcycle dealership`;

SELECT motomodel.idModels, Naming, idBrand, idYear,

CASE

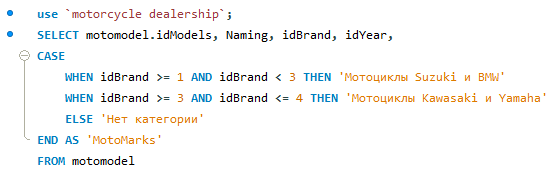
WHEN idBrand >= 1 AND idBrand < 3 THEN 'Мотоциклы Suzuki и BMW'

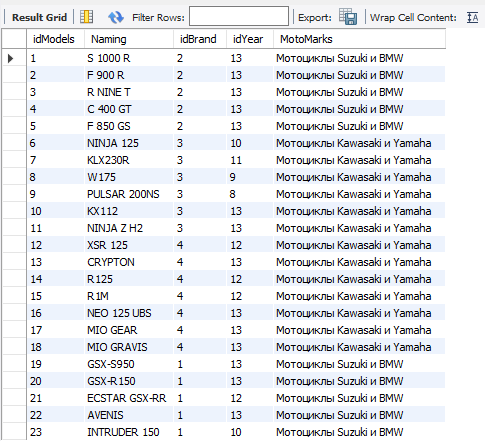
WHEN idBrand >= 3 AND idBrand <= 4 THEN 'Мотоциклы Kawasaki и Yamaha'

ELSE 'Нет категории'

END AS 'MotoMarks'

FROM motomodel

****

****

**Рисунок 1 -** Результат запроса

1. **На основе обновляющего представления (многотабличного VIEW), в котором критерий упорядоченности задает пользователь при выполнении**

create view EmployeePosition as

select name as forename, surname, PostName as Post

from staff, listofpost

where listofpost.IdPost = staff.idPost;





**Рисунок 2** - Результат запроса

1. **Запрос, содержащий коррелированные и некоррелированные подзапросы в разделах SELECT, FROM и WHERE (в каждом хотя бы по одному)**

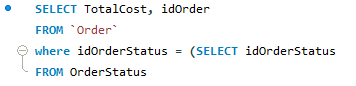
SELECT TotalCost, idOrder

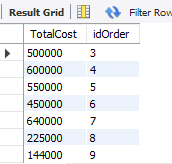
FROM `Order`

where idOrderStatus = (SELECT idOrderStatus

FROM OrderStatus

where OrderName = 'Accept')





**Рисунок 3** - Результат запроса

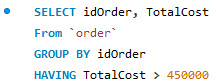
1. **Многотабличный запрос, содержащий группировку записей, агрегативные функции и параметр, используемый в разделе HAVING**

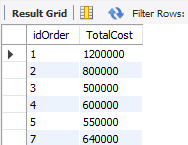
SELECT idOrder, TotalCost

From `order`

GROUP BY idOrder

HAVING TotalCost > 450000





**Рисунок 4** – Результат запроса

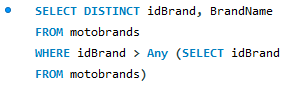
1. **Запрос, содержащий предикат ANY, SOME или ALL**

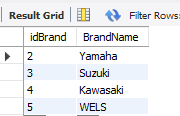
SELECT DISTINCT idBrand, BrandName

FROM motobrands

WHERE idBrand > Any (SELECT idBrand

FROM motobrands)





**Рисунок 5** – результат запроса

## **Листинг SQL-скрипта всех созданных объектов БД**

1. **Создание таблиц**

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`ListOfPost` (

`IdPost` INT NOT NULL AUTO\_INCREMENT,

`PostName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`IdPost`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`RolesInTheOrders`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`RolesInTheOrders` (

`idRoles` INT NOT NULL AUTO\_INCREMENT,

`RoleName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idRoles`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`UserStatus`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`UserStatus` (

`idUserStatus` INT NOT NULL AUTO\_INCREMENT,

`StatusName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idUserStatus`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`Staff`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`Staff` (

`idStaff` INT NOT NULL AUTO\_INCREMENT,

`Surname` VARCHAR(45) NOT NULL,

`Name` VARCHAR(45) NOT NULL,

`Patronymic` VARCHAR(45) NULL,

`DateOfBirth` DATE NOT NULL,

`Email` VARCHAR(45) NOT NULL,

`Password` INT NOT NULL,

`Phone` VARCHAR(45) NOT NULL,

`idPost` INT NOT NULL,

`idRoles` INT NOT NULL,

`idUserStatus` INT NOT NULL,

PRIMARY KEY (`idStaff`),

INDEX `Post\_idx` (`idPost` ASC) VISIBLE,

INDEX `Roles\_idx` (`idRoles` ASC) VISIBLE,

INDEX `UserStatus\_idx` (`idUserStatus` ASC) VISIBLE,

CONSTRAINT `Post`

FOREIGN KEY (`idPost`)

REFERENCES `Motorcycle Dealership`.`ListOfPost` (`IdPost`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Roles`

FOREIGN KEY (`idRoles`)

REFERENCES `Motorcycle Dealership`.`RolesInTheOrders` (`idRoles`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `UserStatus`

FOREIGN KEY (`idUserStatus`)

REFERENCES `Motorcycle Dealership`.`UserStatus` (`idUserStatus`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`Clients`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`Clients` (

`idClient` INT NOT NULL AUTO\_INCREMENT,

`Surname` VARCHAR(45) NOT NULL,

`Name` VARCHAR(45) NOT NULL,

`Patronymic` VARCHAR(45) NULL,

`DateOfBirth` DATE NOT NULL,

`Email` VARCHAR(45) NOT NULL,

`Password` INT NOT NULL,

`Phone` VARCHAR(45) NOT NULL,

`idUserStatus` INT NOT NULL,

PRIMARY KEY (`idClient`),

INDEX `UserStatus\_idx` (`idUserStatus` ASC) VISIBLE,

CONSTRAINT `UserStatus`

FOREIGN KEY (`idUserStatus`)

REFERENCES `Motorcycle Dealership`.`UserStatus` (`idUserStatus`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`TypeOfMotorcycle`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`TypeOfMotorcycle` (

`idTypeOfMotorcycle` INT NOT NULL AUTO\_INCREMENT,

`Naming` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idTypeOfMotorcycle`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`EngineType`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`EngineType` (

`idEngineType` INT NOT NULL AUTO\_INCREMENT,

`Naming` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idEngineType`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`Mileage`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`Mileage` (

`idMileage` INT NOT NULL AUTO\_INCREMENT,

`Meaning` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idMileage`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`VINNumber`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`VINNumber` (

`idVINNumber` INT NOT NULL AUTO\_INCREMENT,

`Naming` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idVINNumber`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`Condition`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`Condition` (

`idCondition` INT NOT NULL AUTO\_INCREMENT,

`Meaning` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idCondition`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`FuelConsumption`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`FuelConsumption` (

`idFuelConsumption` INT NOT NULL AUTO\_INCREMENT,

`Meaning` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idFuelConsumption`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`TechincalSpecifications`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`TechincalSpecifications` (

`idTechincalSpecifications` INT NOT NULL AUTO\_INCREMENT,

`idTypeOfMotorcycle` INT NOT NULL,

`idEngineType` INT NOT NULL,

`idMileage` INT NOT NULL,

`idVINNumber` INT NOT NULL,

`idCondition` INT NOT NULL,

`idFuelConsumption` INT NOT NULL,

PRIMARY KEY (`idTechincalSpecifications`),

INDEX `EngineType\_idx` (`idEngineType` ASC) VISIBLE,

INDEX `TypeOfMotorcycle\_idx` (`idTypeOfMotorcycle` ASC) VISIBLE,

INDEX `Mileage\_idx` (`idMileage` ASC) VISIBLE,

INDEX `VINNumber\_idx` (`idVINNumber` ASC) VISIBLE,

INDEX `Condition\_idx` (`idCondition` ASC) VISIBLE,

INDEX `FuelConsumption\_idx` (`idFuelConsumption` ASC) VISIBLE,

CONSTRAINT `TypeOfMotorcycle`

FOREIGN KEY (`idTypeOfMotorcycle`)

REFERENCES `Motorcycle Dealership`.`TypeOfMotorcycle` (`idTypeOfMotorcycle`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `EngineType`

FOREIGN KEY (`idEngineType`)

REFERENCES `Motorcycle Dealership`.`EngineType` (`idEngineType`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Mileage`

FOREIGN KEY (`idMileage`)

REFERENCES `Motorcycle Dealership`.`Mileage` (`idMileage`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `VINNumber`

FOREIGN KEY (`idVINNumber`)

REFERENCES `Motorcycle Dealership`.`VINNumber` (`idVINNumber`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Condition`

FOREIGN KEY (`idCondition`)

REFERENCES `Motorcycle Dealership`.`Condition` (`idCondition`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `FuelConsumption`

FOREIGN KEY (`idFuelConsumption`)

REFERENCES `Motorcycle Dealership`.`FuelConsumption` (`idFuelConsumption`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`MotoBrands`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`MotoBrands` (

`idBrand` INT NOT NULL AUTO\_INCREMENT,

`BrandName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idBrand`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`ListOfReleaseYears`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`ListOfReleaseYears` (

`idYear` INT NOT NULL AUTO\_INCREMENT,

`Meaning` DATE NOT NULL,

PRIMARY KEY (`idYear`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`MotoModel`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`MotoModel` (

`idModels` INT NOT NULL AUTO\_INCREMENT,

`Naming` VARCHAR(45) NOT NULL,

`idBrand` INT NOT NULL,

`idYear` INT NOT NULL,

PRIMARY KEY (`idModels`),

INDEX `Brand\_idx` (`idBrand` ASC) VISIBLE,

INDEX `Year\_idx` (`idYear` ASC) VISIBLE,

CONSTRAINT `Brand`

FOREIGN KEY (`idBrand`)

REFERENCES `Motorcycle Dealership`.`MotoBrands` (`idBrand`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Year`

FOREIGN KEY (`idYear`)

REFERENCES `Motorcycle Dealership`.`ListOfReleaseYears` (`idYear`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`Moto`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`Moto` (

`idMoto` INT NOT NULL AUTO\_INCREMENT,

`idTechnicalSpecifications` INT NOT NULL,

`idModels` INT NOT NULL,

`idTypeOfMotorcycle` INT NOT NULL,

PRIMARY KEY (`idMoto`),

INDEX `TechnicalSpecifications\_idx` (`idTechnicalSpecifications` ASC) VISIBLE,

INDEX `Models\_idx` (`idModels` ASC) VISIBLE,

INDEX `TypeOfMotorcycle\_idx` (`idTypeOfMotorcycle` ASC) VISIBLE,

CONSTRAINT `TechnicalSpecifications`

FOREIGN KEY (`idTechnicalSpecifications`)

REFERENCES `Motorcycle Dealership`.`TechincalSpecifications` (`idTechincalSpecifications`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Models`

FOREIGN KEY (`idModels`)

REFERENCES `Motorcycle Dealership`.`MotoModel` (`idModels`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `TypeOfMotorcycle`

FOREIGN KEY (`idTypeOfMotorcycle`)

REFERENCES `Motorcycle Dealership`.`TypeOfMotorcycle` (`idTypeOfMotorcycle`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`OrderStatus`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`OrderStatus` (

`idOrderStatus` INT NOT NULL AUTO\_INCREMENT,

`OrderName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idOrderStatus`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`Order`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`Order` (

`idOrder` INT NOT NULL AUTO\_INCREMENT,

`idStaff` INT NOT NULL,

`idClient` INT NOT NULL,

`idMoto` INT NOT NULL,

`OrderDate` DATE NOT NULL,

`idOrderStatus` INT NOT NULL,

`TotalCost` INT NOT NULL,

PRIMARY KEY (`idOrder`),

INDEX `Staff\_idx` (`idStaff` ASC) VISIBLE,

INDEX `Client\_idx` (`idClient` ASC) VISIBLE,

INDEX `Moto\_idx` (`idMoto` ASC) VISIBLE,

INDEX `OrderStatus\_idx` (`idOrderStatus` ASC) VISIBLE,

CONSTRAINT `Staff`

FOREIGN KEY (`idStaff`)

REFERENCES `Motorcycle Dealership`.`Staff` (`idStaff`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Client`

FOREIGN KEY (`idClient`)

REFERENCES `Motorcycle Dealership`.`Clients` (`idClient`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Moto`

FOREIGN KEY (`idMoto`)

REFERENCES `Motorcycle Dealership`.`Moto` (`idMoto`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `OrderStatus`

FOREIGN KEY (`idOrderStatus`)

REFERENCES `Motorcycle Dealership`.`OrderStatus` (`idOrderStatus`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`ListOfServices`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`ListOfServices` (

`idServices` INT NOT NULL AUTO\_INCREMENT,

`NameServices` VARCHAR(45) NOT NULL,

`ShortDescription` VARCHAR(45) NOT NULL,

`Cost` FLOAT NOT NULL,

PRIMARY KEY (`idServices`))

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`ListOfServisecInTheOrder`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`ListOfServisecInTheOrder` (

`idOrder` INT NOT NULL AUTO\_INCREMENT,

`idServices` INT NOT NULL,

PRIMARY KEY (`idOrder`),

INDEX `Services\_idx` (`idServices` ASC) VISIBLE,

CONSTRAINT `Services`

FOREIGN KEY (`idServices`)

REFERENCES `Motorcycle Dealership`.`ListOfServices` (`idServices`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `Motorcycle Dealership`.`MotoPhotos`

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

CREATE TABLE IF NOT EXISTS `Motorcycle Dealership`.`MotoPhotos` (

`idMoto` INT NOT NULL AUTO\_INCREMENT,

`File` VARCHAR(45) NULL,

PRIMARY KEY (`idMoto`),

CONSTRAINT `Moto`

FOREIGN KEY (`idMoto`)

REFERENCES `Motorcycle Dealership`.`Moto` (`idMoto`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

USE `motorcycle dealership` ;

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

-- Table `motorcycle dealership`.`listofpost`

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

CREATE TABLE IF NOT EXISTS `motorcycle dealership`.`listofpost` (

`IdPost` INT NOT NULL AUTO\_INCREMENT,

`PostName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`IdPost`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

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

-- Table `motorcycle dealership`.`rolesintheorders`

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

CREATE TABLE IF NOT EXISTS `motorcycle dealership`.`rolesintheorders` (

`idRoles` INT NOT NULL AUTO\_INCREMENT,

`RoleName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idRoles`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

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

-- Table `motorcycle dealership`.`userstatus`

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

CREATE TABLE IF NOT EXISTS `motorcycle dealership`.`userstatus` (

`idUserStatus` INT NOT NULL AUTO\_INCREMENT,

`StatusName` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idUserStatus`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

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

-- Table `motorcycle dealership`.`staff`

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

CREATE TABLE IF NOT EXISTS `motorcycle dealership`.`staff` (

`idStaff` INT NOT NULL AUTO\_INCREMENT,

`Surname` VARCHAR(45) NOT NULL,

`Name` VARCHAR(45) NOT NULL,

`Patronymic` VARCHAR(45) NULL DEFAULT NULL,

`DateOfBirth` DATE NOT NULL,

`Email` VARCHAR(45) NOT NULL,

`Password` INT NOT NULL,

`Phone` VARCHAR(45) NOT NULL,

`idPost` INT NOT NULL,

`idRoles` INT NOT NULL,

`idUserStatus` INT NOT NULL,

PRIMARY KEY (`idStaff`),

INDEX `Post\_idx` (`idPost` ASC) VISIBLE,

INDEX `Roles\_idx` (`idRoles` ASC) VISIBLE,

INDEX `UserStatus\_idx` (`idUserStatus` ASC) VISIBLE,

CONSTRAINT `Post`

FOREIGN KEY (`idPost`)

REFERENCES `motorcycle dealership`.`listofpost` (`IdPost`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `Roles`

FOREIGN KEY (`idRoles`)

REFERENCES `motorcycle dealership`.`rolesintheorders` (`idRoles`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `UserStatus`

FOREIGN KEY (`idUserStatus`)

REFERENCES `motorcycle dealership`.`userstatus` (`idUserStatus`)

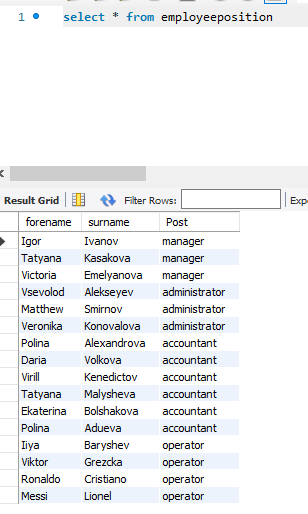
ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

1. **Представление**



**Рисунок 6** – Представление

1. **Исходные тексты всех хранимых процедур**

USE `motorcycle dealership`;

DROP procedure IF EXISTS ` employeeposition `;

DELIMITER $$

USE `motorcycle dealership`$$

create procedure employeeposition (in id int)

begin

select \* from staff

where idStaff = id;

end$$

DELIMITER ;

1. **Триггерры**

DROP TRIGGER IF EXISTS `motorcycle dealership`.`moto\_AFTER\_INSERT`;

DELIMITER $$

USE `motorcycle dealership`$$

CREATE DEFINER=`root`@`localhost` TRIGGER `moto\_AFTER\_INSERT` AFTER INSERT ON `moto` FOR EACH ROW BEGIN

UPDATE `order`

SET TotalCost = TotalCost + idModels

where idMoto = NEW.idMoto

and idOrder = NEW.Order;

END$$

DELIMITER

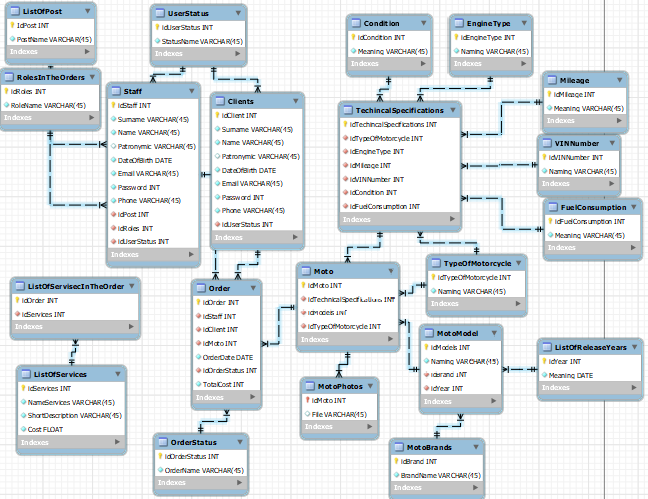
# **Приложение 2**



**Рисунок 1** – Концептуальная модель

# 

**Рисунок 2** - Инфологическая модель



**Рисунок 3** - Физическая модель

1. StatCounter[Электронный ресурс] – Режим доступа: https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202110-202210 [↑](#footnote-ref-1)
2. StatCounter[Электронный ресурс] – Режим доступа: https://gs.statcounter.com/windows-version-market-share/desktop/worldwide/#monthly-202110-202210 [↑](#footnote-ref-2)
3. Столингс Операционные системы: Внутренняя структура и принципы проектирования, 2020, с.112. [↑](#footnote-ref-3)
4. 4Денис Колисниченко Microsoft Windows 11 Первое знакомство, 2022, 42c – 51c. [↑](#footnote-ref-4)
5. Денис Колисниченко Microsoft Windows 11 Первое знакомство, 2022, 51c – 72c. [↑](#footnote-ref-5)
6. Wikipedia [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Database [↑](#footnote-ref-6)
7. StackOverFlow[Электронный ресурс] – Режим доступа: https://insights.stackoverflow.com/survey/2021#demographics-gender-prof [↑](#footnote-ref-7)
8. Harb [Электронный ресурс] – режим доступа: https://habr.com/ru/company/skillfactory/blog/593025/ [↑](#footnote-ref-8)
9. Habr [Электронный ресурс] – режим доступа:]https://habr.com/ru/post/651585/ [↑](#footnote-ref-9)
10. Бек Кент - Экстремальное программирование: разработка через тестирование.: "Издательский дом ""Питер""", 2020 – 224с. [↑](#footnote-ref-10)
11. Сергей Тарасов, СУБД для программиста. Базы данных изнутри., СОЛОН-Пресс – 2021 – 322 с. [↑](#footnote-ref-11)
12. Эдвард Сьорье, Проектирование и реализация систем управления базами данных, 2020 – 461 с. [↑](#footnote-ref-12)
13. Эрих Гамма Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб.: Питер, 2019. – 366 с. [↑](#footnote-ref-13)
14. Гради Буч, Джеймс Рамбо, Ивар Якобсон, Язык UML. Руководство пользователя, ДМК-Пресс - 2020 г. – 496 с. [↑](#footnote-ref-14)
15. Эрик Редмонд, Джим Р. Уилсон "Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL" (2019) – 392 с. [↑](#footnote-ref-15)
16. Иван Аптекарев, ER-модель репозитория поддержки решения задачи QSAR, LAP Lambert Academic Publishing – 2022 – 108 с. [↑](#footnote-ref-16)
17. В.М. Илюшечкин. Основы использования и проектирования баз данных. – М.: Юрайт, Юрайт, 2010. – 224 с. [↑](#footnote-ref-17)
18. М.Ю. Свиридова. Система управления базами данных Access. – М.: Академия, 2021. – 192 с. [↑](#footnote-ref-18)
19. К.Дж. Дейт. Введение в системы баз данных. – М.: Вильямс, 2020. – 1328 с. [↑](#footnote-ref-19)
20. А.В. Благодаров, В.С. Зияутдинов, П.А. Корнев, В.Н. Малыш. Алгоритмы категорирования персональных данных для систем автоматизированного проектирования баз данных информационных систем. – М.: Горячая Линия - Телеком, 2013. – 116 с. [↑](#footnote-ref-20)