**Министерство цифрового развития,   
связи и массовых коммуникаций Российской Федерации  
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технический университет связи и информатики»**

Кафедра Интеллектуальных систем в управлении и автоматизации

**Курсовая работа**

По дисциплине «Технологии баз данных»

По теме «Интернет-магазин электронной техники «Светлячок»»

Выполнил: студент группы БВТ1801

Задоркин Максим Александрович

Руководитель:

Воронов Вячеслав Игоревич

Москва 2021

**Оглавление**

[Введение 3](#_Toc73322909)

[Задачи курсового проекта 3](#_Toc73322910)

[Обоснование выбора темы проекта 3](#_Toc73322911)

[Глава 1. Анализ предметной области АСУ «Светлячок» 4](#_Toc73322912)

[Общий анализ продаж магазина «Светлячок». 5](#_Toc73322913)

[Анализ текущих решений. 6](#_Toc73322914)

[Технологии для реализации. 8](#_Toc73322915)

[Требования к разрабатываемой базе данных магазина «Светлячок» 9](#_Toc73322916)

[Выводы 10](#_Toc73322917)

[Глава 2. Проектирование базы данных для магазина продажи электронной техники «Светлячок» 11](#_Toc73322918)

[Инфологическое моделирование 11](#_Toc73322919)

[Обоснование выбора модели данных 13](#_Toc73322920)

[Сетевая модель 13](#_Toc73322921)

[Иерархическая модель 14](#_Toc73322922)

[Объектно-ориентированная модель 15](#_Toc73322923)

[Реляционная модель 16](#_Toc73322924)

[Многомерные структуры 17](#_Toc73322925)

[Даталогическое проектирование 17](#_Toc73322926)

[Нормализация, схема БД 20](#_Toc73322927)

[Выводы 22](#_Toc73322928)

[Глава 3. Программная реализация 23](#_Toc73322929)

[Анализ и выбор СУБД 23](#_Toc73322931)

[Физическое проектирование базы данных в СУБД 23](#_Toc73322932)

[Разработка форм 24](#_Toc73322933)

[Автоматизация обработки данных в БД 31](#_Toc73322934)

[Безопасность и контроль 31](#_Toc73322935)

[Выводы 32](#_Toc73322936)

[Заключение 33](#_Toc73322937)

[Источники информации 34](#_Toc73322938)

[Приложение №1. Программный код модуля для создания таблиц 35](#_Toc73322939)

[Приложение №2. Программный код триггеров 43](#_Toc73322940)

# Введение

Целью курсового проекта является разработка базы данных для интернет-магазина электронной техники.

Задачи курсового проекта:

1. Анализ системной области;
2. Обзор информационных технологий, подходящих для разработки;
3. Анализ аналогичных информационных систем;
4. Описание требований, предъявляемых к разработке баз данных;
5. Разработка модели баз данных;
6. Обоснование выбора модели данных и осуществление проектирования;
7. Составление схемы базы данных;
8. Осуществление реализации.

Обоснование выбора темы проекта.

За последнее десятилетие активно развиваются интернет-технологии. Большинство средних, крупных бизнесов начали заниматься интернет-продажами. Для доказательства актуальности работы можно привести мировую статистику продаж электронной коммерции от компании Statista, на котором так же приведены предположительные данные на несколько лет вперёд.

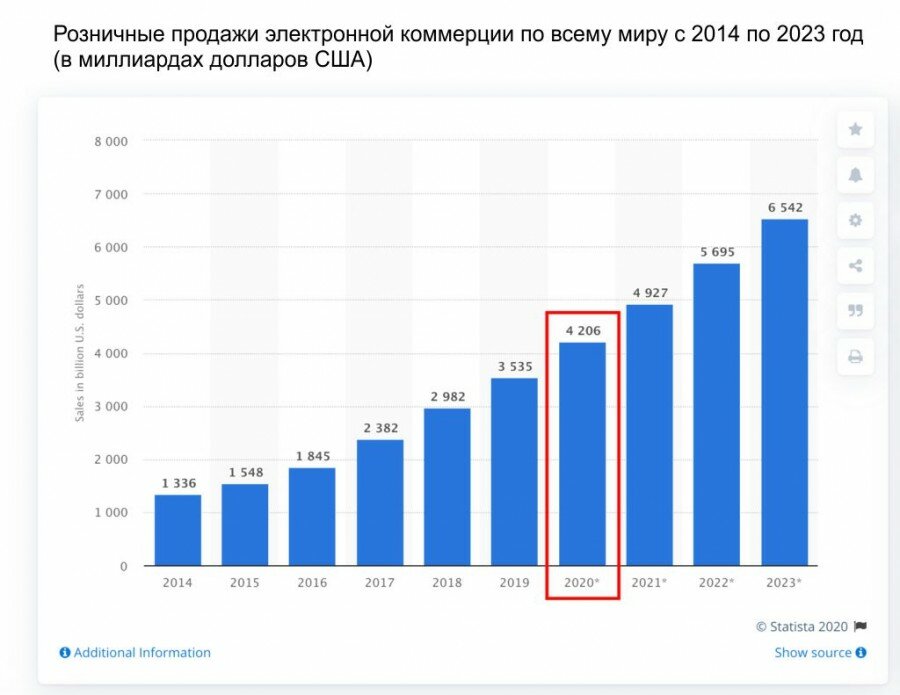


Рисунок 1 – статистика электронной коммерции

Из чего следует, что цифровая коммерция с каждым годом становится все более обширной и тема разработки базы данных для интернет-магазина является актуальной.

# Глава 1. Анализ предметной области АСУ «Светлячок»

Магазин “Светлячок” — это молодой развивающийся бизнес, история которого составляет отрезок времени, длиною в 6 лет. Основным направлением деятельности является продажа электронной техники, включая следующие категории товаров:

1. Смартфоны;
2. Компьютеры;
3. Ноутбуки;
4. Планшеты;
5. Умные часы;
6. Аксессуары;
7. Наушники.

На данный момент все предприятие включает в себя 20 человек. На рисунке 2 представлена организационная структура магазина бытовой техники «Светлячок», которая состоит из руководителя, бухгалтерии, менеджера, администратора, кассы, продавца и курьера.

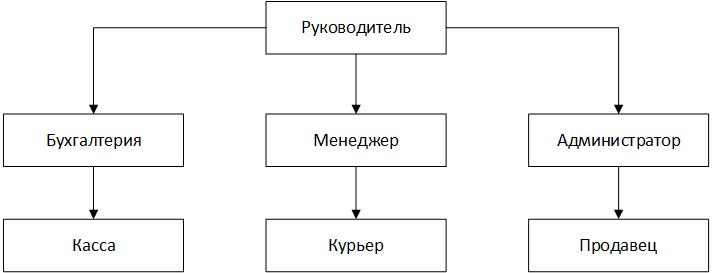


Рисунок 2 – Организационная структура магазина электронной техники «Светлячок»

Чтобы расширить свою сферу влияния, директор магазина хочет создать интернет-магазин. Далее по тексту рассмотрим устройство магазина и предполагаемое решение более подробно.

## Общий анализ продаж магазина «Светлячок».

В начале анализа определимся с теми рабочими местами, которые есть в магазине в данный момент и их ключевыми функциональными особенностями для магазина:

1. Директор предприятия;
   1. Ключевая отчётность.
2. Администратор торгового зала;
   1. Подбор персонала.
3. Товаровед торгового зала;
   1. Контроль остатков товара;
   2. Контроль цен;
   3. Составление отчётов для организации поставок.
4. Оператор;
   1. Распечатка этикеток, ценников, списков.
5. Консультант;
   1. Консультация покупателей.
6. Кассир предприятия;
   1. Регулирование покупок на кассовом аппарате.

Составим схему бизнес-процесса продажи товара.

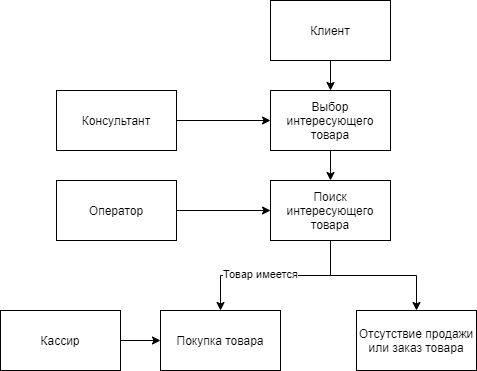


Рисунок 3 – схема процесса продажи товара

Основными проблемами такого подхода являются большое количество работников на различных точках и неэффективная тактика продаж.

## Анализ текущих решений.

**1С: CRM Управление взаимоотношениями с клиентами** – это программная система, которая содержит в себе инновационные методы и функции по эффективному и автоматизированному управлению взаимоотношениями с клиентами. Использование программного комплекса позволяет формировать пользователям самые действенные сценарии взаимодействия с контрагентами, что обеспечивает возможность не только их удерживать, но и привлекать новых. Таким образом, создаются условия для увеличения количества успешно закрытых сделок, уменьшения потерь, а также оптимизации клиент-сервиса и прироста доходов фирмы в целом.

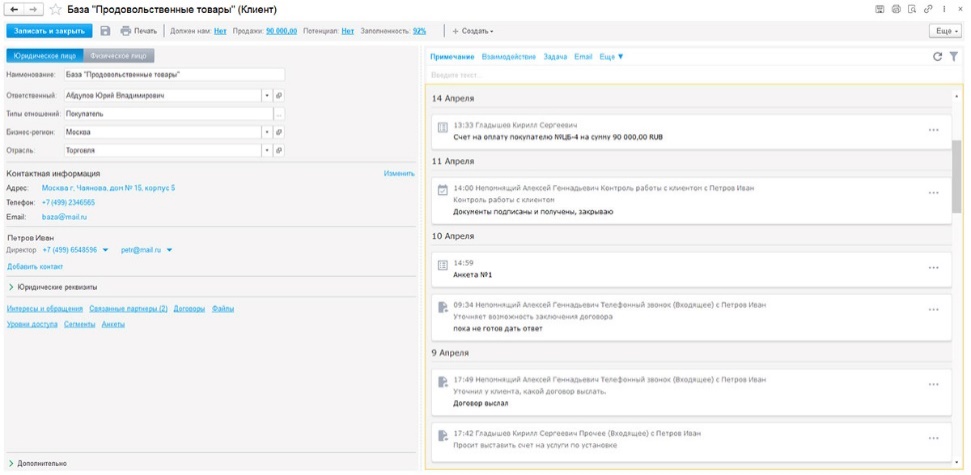


Рисунок 4 – 1C: CRM Управление взаимоотношениями с клиентами

**amoCRM** — это система управления клиентами и сделками, которая помогает контролировать и может способствовать увеличению продаж. Программный продукт amoCRM— это комплексная система управления взаимоотношениями с клиентами (CRM) позволяет сократить время вашей команды продаж за счёт автоматизации рутинной работы. Управление задачами и исполнителями, создание почтовых рассылок и общение с коллегами в одном приложении, а для удобства «полевых работ» представлено мобильное приложение для iOS и Android.

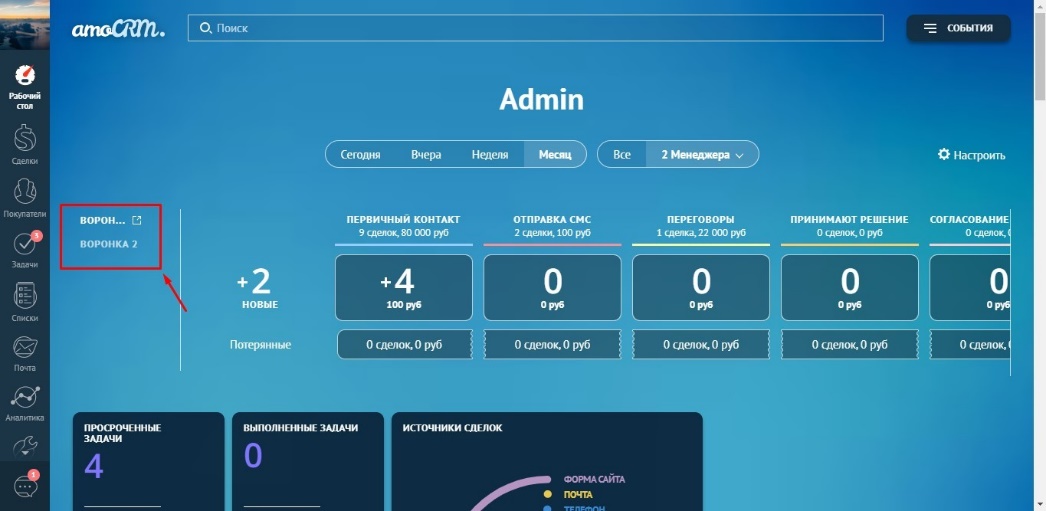


Рисунок 5 – amoCRM

**Битрикс24** - российский сервис для управления бизнесом. Разработчик и провайдер — российская компания «1С-Битрикс». Сервис позволяет управлять бизнесом в режиме «одного окна». Включает в себя:

* CRM
* менеджер задач
* корпоративную социальную сеть
* чаты
* конструктор сайтов
* конструктор интернет-магазинов
* облачное хранилище и онлайн-документы
* календарь
* бизнес-процессы
* учёт рабочего времени
* сквозную аналитику
* мобильное приложение
* аудио/видеозвонки и видеоконференции в HD
* генератор документов и отчётов
* структуру компании

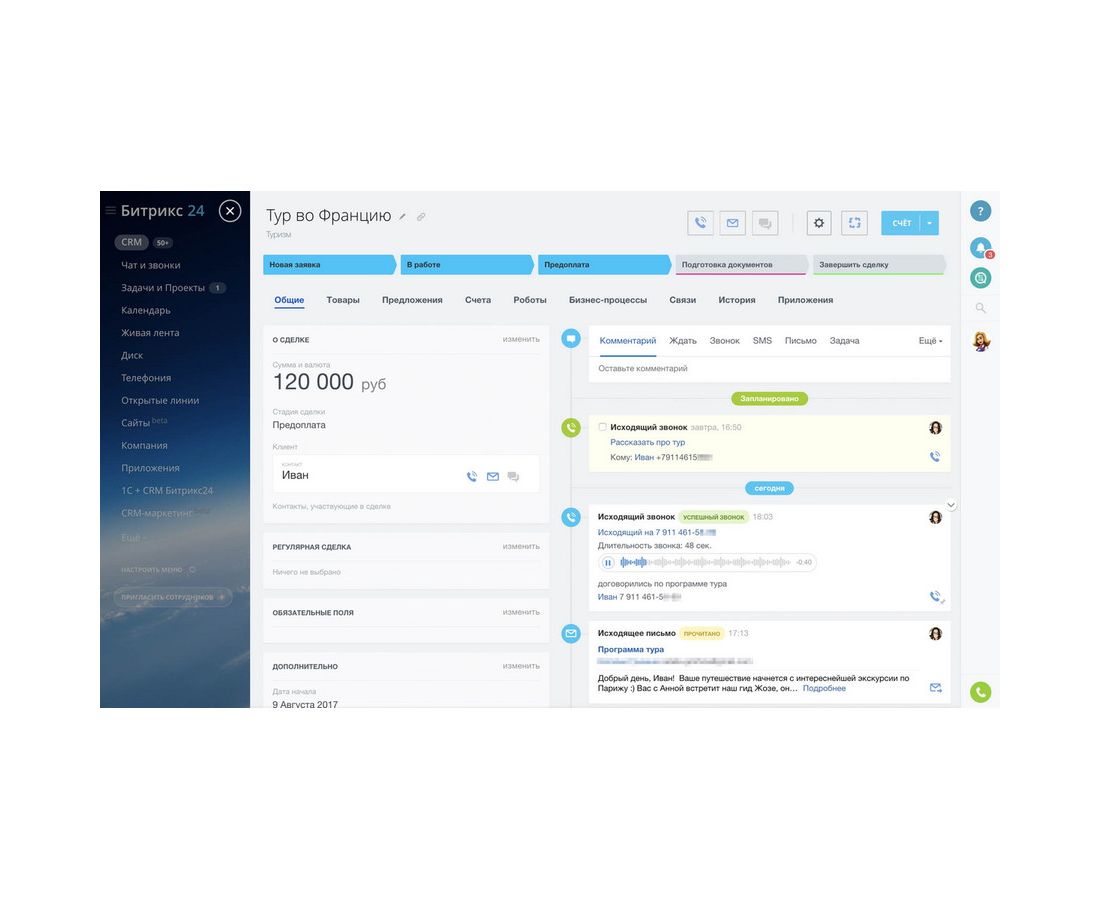


Рисунок 6 – Битрикс24

При анализе вышеперечисленных программных решений можно вывести ключевые компоненты любой системы интернет-магазина:

1. Система сделок;
2. Клиенты. Система личного кабинета и оформления заказов;
3. Товары. Деления на категории, ведение и учет ассортимента;
4. Сотрудники.

## Технологии для реализации.

На данный момент рынок представляет различные базы данных. Основное разделение на sql и nosql представителей.  
Самыми популярными sql представителями являются:

1. Oracle
2. MySQL
3. MSSQL
4. PostgreSql

Из nosql языков наиболее популярным является MongoDB, которая использует для хранения популярный в веб разработке формат JSON.

Различия между двумя подходами заключаются в способах хранения данных. Реляционные базы данных хранят структурированные данные, сгруппированные в таблицах, формат которых задан на этапе проектирования хранилища. Не реляционные базы данных могут использовать иерархическую структуру данных. Преимущество их состоит в том, что они могут хранить не структурированную информацию и не накладывают ограничений на типы хранимых данных.

Дадим описание для вышеперечисленных технологий.

**Oracle Database**

Oracle Database - первая в мире база данных, разработанная специально для работы в сетях распределенных вычислений. Oracle Database предназначена для эффективного развертывания на базе различных типов оборудования, от небольших серверов до Oracle Enterprise Grid мощных многопроцессорных серверных систем, от отдельных кластеров до корпоративных распределенных вычислительных систем.

Oracle Database позволяет пользователям виртуализировать использование аппаратного обеспечения - серверов и систем хранения данных. Oracle Database обладает технологиями, которые позволяют администраторам надежно хранить и быстро распределять, и извлекать данные для пользователей и приложений, работающих в сетях Grid. Oracle Database значительно повышает производительность обработки данных и включает в себя удобные средства администрирования.

**MySql**

MySQL является собственностью компании Sun Microsystems, осуществляющей разработку и поддержку приложения. Распространяется под GNU General Public License и под собственной коммерческой лицензией, на выбор. Помимо этого, компания MySQL AB разрабатывает функциональность по заказу лицензионных пользователей, именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в LAMP. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

**MSSQL**

Microsoft SQL Server — 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 (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

**PostgreSql**

PostgreSQL — это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных. PostgreSQL — СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций.

**MongoDB**

MongoBD — это документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, который написана на языке C++ и не требует описание схемы таблиц. MongoDB классифицирована как NoSQL база данных с JSON-подобными документы (точнее BSON — Binary JavaScript Object Notation). То есть, она реализует новый подход к построению базы данных. Используя MongoDB, у вас не будет таблиц, схем, SQL запросов, внешних ключей и много других вещей, которые встречаются в SQL базах.

Для реализации данного проекта будет выбрана реляционная база данных. Из вышеперечисленных представителей выбрана Oracle Database. Из преимуществ данной системы можно выделить распространенность, кроссплатформеность, бесплатное распространение базовой версии, безопасность.

## Требования к разрабатываемой базе данных магазина «Светлячок»

В соответствии с ГОСТ 34.601-90 сформированы следующие требования:

С данной базой данных могут работать следующие группы пользователей:

* администратор;
* продавец;
* курьер;
* клиент.

При работе с базой данных администратор и менеджер может выполнять следующие задачи:

* вносить изменения в личные данные клиентов и работников;
* добавлять или удалять информацию о товарах;
* редактировать или добавлять информацию о заказах;
* посматривать любую информацию.

При работе с базой данных продавец может выполнять следующие задачи:

* просматривать информацию по чекам;
* редактировать или добавлять информацию о заказах;
* посматривать любую информацию.

При работе с базой данных клиент и курьер могут:

* просматривать информацию о заказах;
* просматривать информацию о доставке.

Для данной базы данных требуется предусмотреть следующие ограничения:

* работники не моложе 18 лет;
* у каждого сотрудника должны быть обязательно заполнены все данные;
* при заказе обязательно требуется заполнение полей ФИО, мобильного телефона, адреса доставки.

## Выводы

В первой главе проведен системный анализ предметной области объекта автоматизации «Светлячок», в ходе которого перечислены должности работников и их функции.

Рассмотрены продукты-аналоги на рынке информационных систем (1С, Битрикс24) и даны описания данных систем.

В ходе обзора информационных технологий перечислены классы СУБД, приведены примеры для каждого вида (MySQL, MongoDB).

Указаны требования к разрабатываемой базе данных со стороны каждой из групп пользователей и перечислены выполняемые этими пользователями задачи относительно базы данных. Также описаны ограничения на разрабатываемую БД.

Глава 2. Проектирование базы данных для магазина продажи электронной техники «Светлячок»

В данной главе разработаем инфологическую модель базы данных магазина «Светлячок». Проанализируем существующие даталогические модели данных и обоснуем выбор реляционной модели. На основе построенной инфологической модели проведем логическое проектирование базы данных, опишем каждую сущность и построим реляционную модель базы данных магазина «Светлячок».

## Инфологическое моделирование

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком (последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественного языка). Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

Сущность – любой различимый объект (объект, который мы можем отличить от другого), информацию о котором необходимо хранить в базе данных.

Атрибут – поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей. Атрибуты используются для определения того, какая информация должна быть собрана о сущности.

Связь – ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных – это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи.

Суть инфологического моделирования состоит в выделении сущностей (Информационных объектов предметной области), которые подлежат хранению в базе данных, а также в определении характеристик объектов и взаимосвязей между ними.

Для информационной системы «Магазин Светлячок» на основе проведенного системного анализа предметной области выделены следующие сущности:

1. Категории: содержит информацию о возможных категориях товаров;
2. Товары: содержит информацию о продаваемых товарах;
3. Производитель: содержит производителей товаров;
4. Сделки: содержит информацию о совершенных и совершаемых сделках;
5. Чек: содержит информацию о продаже товара;
6. СтатусыСделки: содержит информацию о возможных статусах сделки;
7. Клиенты: содержит информацию о всех пользователях-покупателях;
8. Должности: содержит информацию о возможных должностях работников;
9. Сотрудники: содержит информацию о всех работниках;
10. Доставки: содержит информацию о доставках;
11. Поставки: содержит информацию о поставках;
12. Поставщик: содержит информацию о поставщиках магазина.

Исходя из приведенных выше сущностей, построена инфологическая модель предметной области, которая представлена на рисунке 7.

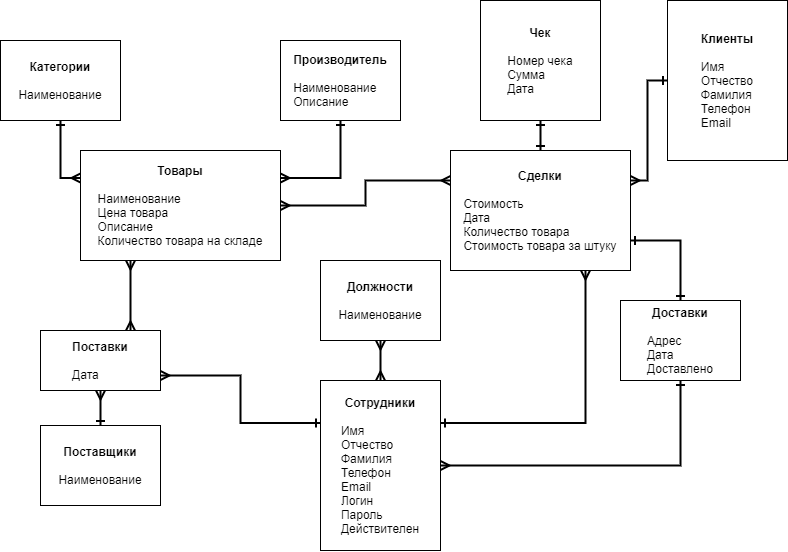


Рисунок 7 – Инфологическая модель базы данных

## Обоснование выбора модели данных

Под даталогической моделью понимается модель, отражающая логические взаимосвязи между элементами данных безотносительно их содержания и физические организации. При этом даталогическая (или просто логическая) модель строится на основе инфологической модели конкретной предметной области, с учётом её особенностей.

Существуют несколько типов даталогических моделей данных:

* сетевая модель;
* иерархическая модель;
* объектно-ориентированная модель;
* реляционная модель;

Необходимо выбрать один из приведённых выше типов и построить на основе инфологической модели, разработанной ранее, даталогическую модель данной ИС. Также необходимо выбрать СУБД, в которой, впоследствии, будет реализована данная БД, т.к. даталогическая модель строится в терминах выбранной СУБД.

Рассмотрим подробнее каждый тип даталогической модели.

## Сетевая модель

Сетевая модель данных — логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.

Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

* каждый экземпляр типа записи P является предком только в одном экземпляре типа связи L;
* каждый экземпляр типа записи C является потомком не более чем в одном экземпляре типа связи L.

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. Недостатком сетевой модели данных являются высокая сложность и жесткость схемы БД, построенной на ее основе.

Пример сетевой модели показан на рисунке 8.

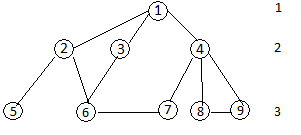


Рисунок 8 - Сетевая модель

## Иерархическая модель

Иерархическая модель данных представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможна ситуация, когда объект-предок не имеет потомков или имеет их несколько, тогда как у объекта-потомка обязательно только один предок.

Как и сетевая, иерархическая модель данных базируется на графовой форме построения данных, и на концептуальном уровне она является просто частным случаем сетевой модели данных. В иерархической модели данных вершине графа соответствует тип сегмента или просто сегмент, а дугам — типы связей предок — потомок. В иерархических структурах сегмент — потомок должен иметь в точности одного предка.

Иерархическая модель представляет собой связный неориентированный граф древовидной структуры, объединяющий сегменты. Иерархическая БД состоит из упорядоченного набора деревьев.

Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути.

К основным недостаткам иерархических моделей следует отнести: неэффективность, медленный доступ к сегментам данных нижних уровней иерархии, четкая ориентация на определенные типы запросов и др. Также недостатком иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями, а также сложность понимания для обычного пользователя. Иерархические модели быстро прошли пик популярности, которая обусловливалась их ранним появлением на рынке. Затем их недостатки сделали их неконкурентоспособными, и в настоящее время иерархическая модель представляет исключительно исторический интерес.

На рисунке 9 представлено графическое изображение иерархической модели.

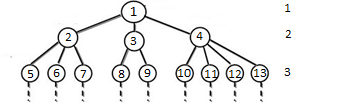


Рисунок 9 - Иерархическая модель

## Объектно-ориентированная модель

В объектно-ориентированной модели при представлении данных имеется возможность идентифицировать отдельные записи базы данных. Между записями и функциями их обработки устанавливаются взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках программирования.

Логическая структура объектно-ориентированной БД внешне похожа на структуру иерархической БД. Основное различие между ними состоит в методах манипулирования данными.

Для выполнения действий над данными в рассматриваемой модели БД применяются логические операции, усиленные объектно-ориентированными механизмами инкапсуляции, наследования и полиморфизма.

Основным достоинством объектно-ориентированной модели данных в сравнении с реляционной является возможность отображения информации о сложных взаимосвязях объектов. Объектно-ориентированная модель данных позволяет идентифицировать отдельную запись базы данных и определять функции их обработки. Недостатками объектно-ориентированной модели являются высокая понятийная сложность, неудобство обработки данных и низкая скорость выполнения запросов.

## Реляционная модель

В реляционной модели достигается гораздо более высокий уровень абстракции данных, чем в иерархической или сетевой. Реляционная модель предоставляет средства описания данных на основе только их естественной структуры, т.е. без потребности введения какой-либо дополнительной структуры для целей машинного представления. Другими словами, представление данных не зависит от способа их физической организации. Это обеспечивается за счет использования математической теории отношений (само название "реляционная" происходит от английского relation - "отношение").

Структура информации дает основание предполагать, что наиболее подходящей для даталогического проектирования будет реляционная модель данных, т.к. она способна обеспечит целостность данных при вставке, удалении и изменении записей, а так же дает возможность организации всех видов связей 1:1, 1:М и М:М (при этом связи М:М раскрываются). К недостаткам традиционных реляционных моделей данных можно отнести является избыточность по полям (из-за создания связей), а также факт того, что в качестве основного и, часто, единственного механизма, обеспечивающего быстрый поиск и выборку отдельных строк таблице (или в связанных через внешние ключи таблицах), обычно используются различные модификации индексов, основанных на B-деревьях. Такое решение оказывается эффективным только при обработке небольших групп записей и высокой интенсивности модификации данных в базах данных.

На рисунке 10 показана реализация реляционной модели данных.

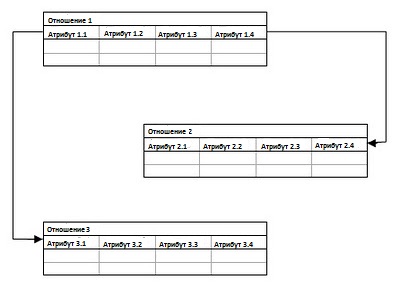


Рисунок 10 - Реляционная модель

## Многомерные структуры

Развивают реляционные модели данных и представляются как гиперкубы данных. Каждая грань куба является размерностью. Основными понятиями, используемыми в многомерных моделях данных, являются «измерение» и «ячейка». Наиболее подходящей моделью данных для этого случая является так называемая многомерная модель, используемая в технологии OLAP (OnLine Analytical Processing – оперативная аналитическая обработка). Отметим, что многомерность модели данных означает здесь многомерное логическое представление структуры информации и, вообще говоря, не связана с многомерностью визуализации.

Такая модель позволяет легко сравнивать данные при разных значениях параметров, строить графики зависимости значений конкретных атрибутов от значений определенных параметров и т.п. Поэтому основное назначение технологии OLAP – обработка информации для проведения анализа и принятия решения.

## Даталогическое проектирование

Для логического проектирования выбрана реляционная модель данных, т.к. она наиболее полно соответствует требованиям, предъявленным к разрабатываемой информационной системе:

* отсутствие дублируемой информации;
* поддержание целостности данных при вставке, удалении или изменении записей;
* возможность организации всех видов связи между отношениями 1:1, 1:M и M: M.

В реляционной базе данных даталогическое проектирование приводит к разработке корректной схемы базы данных, т.е. такой схемы, в которой отсутствуют нежелательные зависимости между атрибутами. При этом можно использовать процесс проектирования с помощью декомпозиции, т.е. последовательно нормализовывать схему отношений, тем самым накладывая ограничения и избавляясь от нежелательных зависимостей между атрибутами.

В реляционных базах данных (РБД) даталогическое проектирование приводит к разработке схемы БД, т.е. совокупности схем отношений, адекватно моделирующих объекты ПО и семантических связей между ними.

Основой анализа корректности схемы являются функциональные зависимости между атрибутами БД. Некоторые могут быть нежелательными.

В конце этого этапа должно быть получено описание схемы БД в терминах выбранной СУБД. Целью даталогического проектирования является построение корректной схемы БД, ориентированную на реляционную модель. Корректной называется схема БД, в которой отсутствуют нежелательные зависимости между атрибутами отношений.

Процесс разработки корректной схемы РБД и является даталогическим проектированием. Возможны 2-а способа:

* Декомпозиция (разбиение);
* Синтез.

Для перехода от инфологической модели к реляционной существует специальный алгоритм:

* каждой сущности ставится в соответствие отношение;
* каждому атрибуту сущности ставится в соответствие соответствующий атрибут соответствующего отношения;
* первичный ключ сущности становится PK соответствующего отношения, при этом атрибуты, входящие в PK, обязательны для заполнения (NOT NULL);
* в каждое отношение, соответствующее подчинённой сущности, добавляется набор атрибутов основной сущности, являющийся в ней первичным ключом. В отношении, соответствующее подчинённой сущности эти атрибуты становятся FK (внешним ключом);
* по умолчанию, все атрибуты, не входящие в PK, необязательны;
* для отражения категоризации сущностей возможны несколько вариантов;
* все связи М:М должны быть раскрыты.

Воспользуемся данным алгоритмом и опишем каждую сущность модели:

1. Товары:
   * Код товара – int NOT NULL PK;
   * Наименование – varchar(100) NOT NULL;
   * Цена – decimal(10,2) NOT NULL;
   * Описание – varchar(500);
   * КоличествоНаСкладе – int NOT NULL;
   * Код категории – int NOT NULL FK;
   * Код производителя – int NOT NULL FK.
2. Категории:
   * Код категории – int NOT NULL PK;
   * Наименование – varchar(100) NOT NULL.
3. Производители:
   * Код производителя – int NOT NULL PK;
   * Наименование – varchar(100) NOT NULL;
   * Описание – varchar(500).
4. Поставки:
   * Код поставки – int NOT NULL PK;
   * Дата – datetime NOT NULL;
   * Код поставщика – int NOT NULL FK;
   * Код сотрудника – int NOT NULL FK.
5. Поставщики:
   * Код поставщика – int NOT NULL PK;
   * Наименование – varchar(100) NOT NULL.
6. Поставки\_Товары:
   * Код товара – int NOT NULL PK FK;
   * Код поставки – int NOT NULL PK FK;
   * Стоимость за штуку – int NOT NULL;
   * КоличествоТовара – int NOT NULL.
7. Должности:
   * Код должности – int NOT NULL PK;
   * Наименование – varchar(100) NOT NULL.
8. Должности\_Сотрудники:
   * Код должности – int NOT NULL PK FK;
   * Код сотрудника – int NOT NULL PK FK.
9. Сотрудники:
   * Код сотрудника – int NOT NULL PK;
   * Имя – varchar(100) NOT NULL;
   * Отчество – varchar(100);
   * Фамилия – varchar(100) NOT NULL;
   * Телефон – varchar(12);
   * Email – varchar(100);
   * Логин – varchar(100) NOT NULL UQ;
   * Пароль – varchar(100) NOT NULL
   * Действителен -tinyint NOT NULL.
10. Сделки:
    * Код сделки – int NOT NULL PK;
    * Код клиента – int NOT NULL FK;
    * Код сотрудника – int NOT NULL FK;
    * Стоимость сделки – int NOT NULL;
    * Дата сделки – datetime NOT NULL.
11. Чек:
    * Код чека – int NOT NULL PK;
    * Сумма – decimal(10,2) NOT NULL;
    * Дата – datetime NOT NULL
    * Код сделки – int NOT NULL FK.
12. Клиенты:
    * Код клиента – int NOT NULL PK;
    * Имя – varchar(100) NOT NULL;
    * Отчество – varchar(100);
    * Фамилия – varchar(100) NOT NULL;
    * Телефон – varchar(12);
    * Email – varchar(100).
13. Доставки:
    * Код доставки – int NOT NULL PK;
    * Дата – datetime;
    * Адрес – varchar(500) NOT NULL;
    * Доставлено – tinyint NOT NULL;
    * Код сотрудника – int NOT NULL FK;
    * Код сделки – int NOT NULL FK.
14. Сделки\_Товары:
    * Код сделки – int NOT NULL FK PK;
    * Код товара – int NOT NULL FK PK;
    * Стоимость за штуку – int NOT NULL;
    * КоличествоТовара – int NOT NULL.

При переходе от инфологической модели к реляционной модели была раскрыта связь М:М между отношениями «Сделки»-«Товары», «Поставки»-«Товары» и «Сотрудники»-«Должности» и получены соответственно следующие таблицы «Сделки\_Товары», «Поставки\_Товары» и «Должности\_Сотрудники» .

Исходя из приведённых выше отношений, построим схему получившейся БД (Рисунок 11):

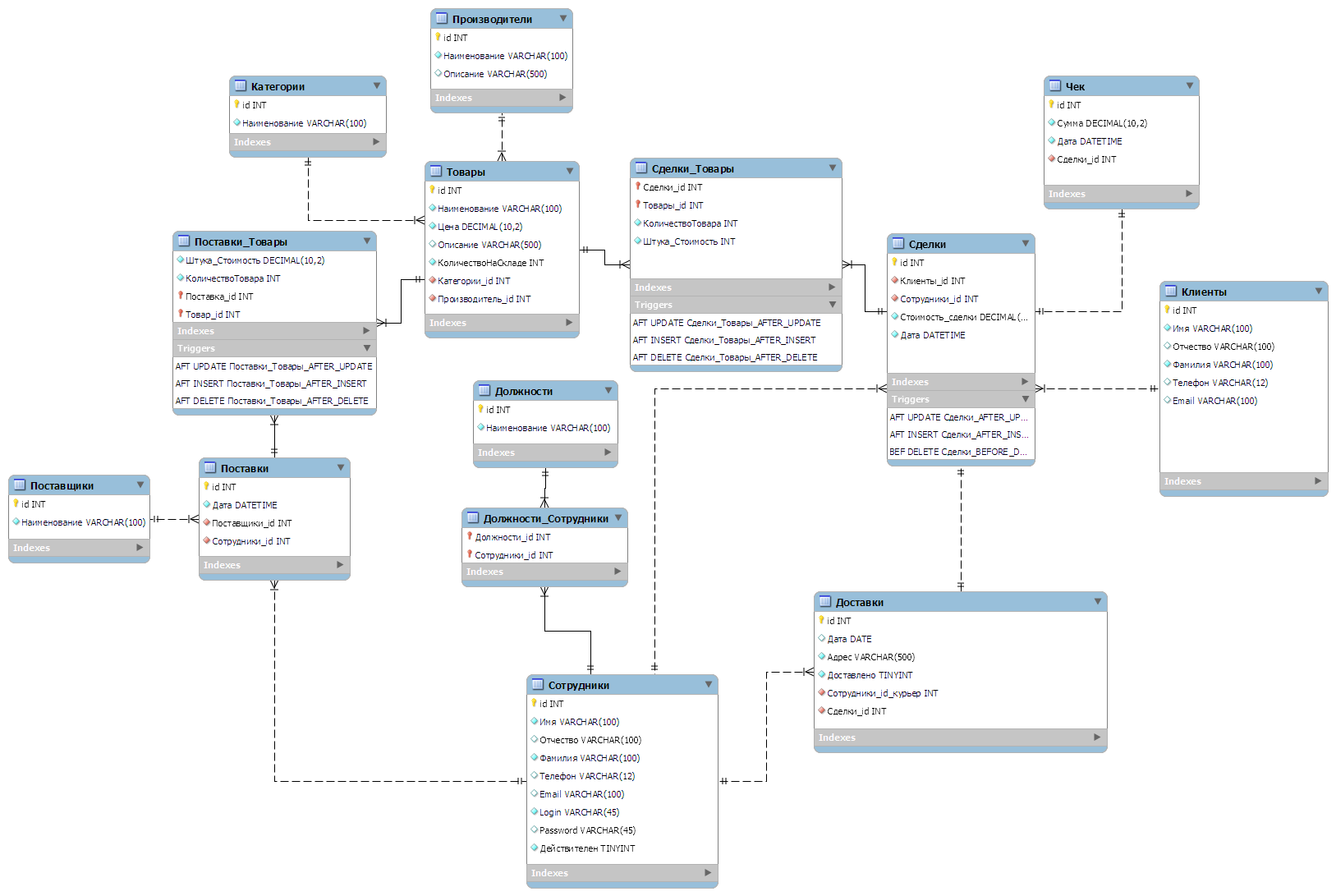


Рисунок 11 – Даталогическая модель базы данных

## Нормализация, схема БД

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, которая потенциально может привести к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Нормализация – это процесс преобразования базы данных к виду, отвечающему нормальным формам. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную избыточность, то есть нормализация не имеет целью уменьшение или увеличение производительности работы, или же уменьшение или увеличение объёма БД. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в БД информации. Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты (то есть факты, не выводимые из других хранимых фактов).

Понятие нормализации тесно связано с понятием функциональной зависимости. Функциональная зависимость (ФЗ) определяет отношения между объектами и их свойствами в рассматриваемой ПО. ФЗ R.AR.B называется полной, если набор атрибутов B ФЗ от A и не зависит функционально от любого подмножества А. ФЗ R.АR.В называется транзитивной, если существует такой набор атрибутов С, который удовлетворяет следующим свойствам:

1. С ¢ А;

2. В ¢ С;

3. существует ФЗ R.А  R.С;

4. не существует ФЗ R.С R.А;

5. не существует ФЗ R.С R.B.

Таблица находится в первой нормальной форме, если каждый её атрибут атомарен, то есть может содержать только одно значение. Таким образом, не существует 1НФ таблицы, в полях которых могут храниться списки значений. Для приведения таблицы к 1НФ обычно требуется разбить таблицу на несколько отдельных таблиц.

Отношение находится во второй нормальной форме, если она находится в первой нормальной форме, и при этом любой её атрибут, не входящий в состав первичного ключа, функционально полно зависит от первичного ключа. Функционально полная зависимость означает, что атрибут функционально зависит от всего первичного составного ключа, но при этом не находится в функциональной зависимости от какой-либо из входящих в него атрибутов (частей). Или другими словами: в 2НФ нет не ключевых атрибутов, зависящих от части составного ключа.

Отношение находится в третьей нормальной форме, если она находится во второй нормальной форме 2НФ и при этом любой ее не ключевой атрибут зависит только от первичного ключа. Таким образом, отношение находится в 3НФ тогда и только тогда, когда оно находится во 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых.

Схема базы данных находится во 2НФ, так как в единственном отношении, имеющим составной первичный ключ (Отношение «Содержание чека»), не ключевой атрибут функционально полно зависит от PK.

3НФ – отношение находится в 3НФ, если оно находится во 2НФ и не содержит транзитивных зависимостей. Все отношения данной модели находятся в 3НФ, т.к. ни в одном из них нет транзитивных зависимостей.

При решении практических задач в большинстве случаев третья нормальная форма является достаточной. Поэтому процесс проектирования базы данных, как правило, заканчивается приведением к ней.

Разрабатываемая база данных уже удовлетворяет требованиям третьей нормальной формы. Следовательно, процесс нормализации проводить не нужно. Схема базы данных показана на рисунке 11.

## Выводы

Во второй главе курсовой работы приведена разработка информационно-логической модели. Выделены сущности, дано их описание и построена инфологическая модель предметной области.

Далее в ходе обоснования выбора модели данных описаны существующие модели данных (иерархическая, сетевая, реляционная, объектно-ориентированная), указаны их достоинства и недостатки, и сделан выбор в пользу реляционной модели.

Затем на основании инфологической модели построена реляционная модель данных, дан список атрибутов ее отношений и проведена нормализация до третьей нормальной формы. Таким образом, завершено проектирование базы данных и получена вся информация, необходимая для реализации проектируемой информационной системы в одной из реляционных СУБД.

# Глава 3. Программная реализация

В данной главе проведем анализ и выбор СУБД. База данных магазина «Светлячок» содержит 14 таблиц, которые будут спроектированы в разделе Физическое проектирование БД. Также будут показаны скриншоты готовых таблиц. В конце главы буду написаны 3 триггера и показана их реализация.



## Анализ и выбор СУБД

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

MySQL Workbench — инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL.

Для реализации форм и отчетов выбран стек технологий JavaScript, главными из которых являются:

* Платформа Nodejs – для реализации языка с серверной стороны приложений
* Electron – фреймворк для создания кроссплатформенных DeskTop приложений
* React-стек для манипуляции и управлением всей основной структурой кода.

## Физическое проектирование базы данных в СУБД

На основе реляционной модели произведена программная реализация. База данных содержит 15 таблиц:

* Категории;
* Производители;
* Товары;
* Поставщики;
* Поставки;
* Поставки\_Товары;
* Сотрудники;
* Должности;
* Должности\_Сотрудники;
* Доставки;
* Сделки;
* Сделки\_Товары;
* Чек;
* Клиенты.

Программный код для создания таблиц базы данных на MySQL представлен в приложение №1.

## Разработка форм

При помощи языка JavaScript было реализовано приложений, позволяющее просматривать, редактировать и добавлять данные в базу данных. Примеры разработанных форм представлены ниже:

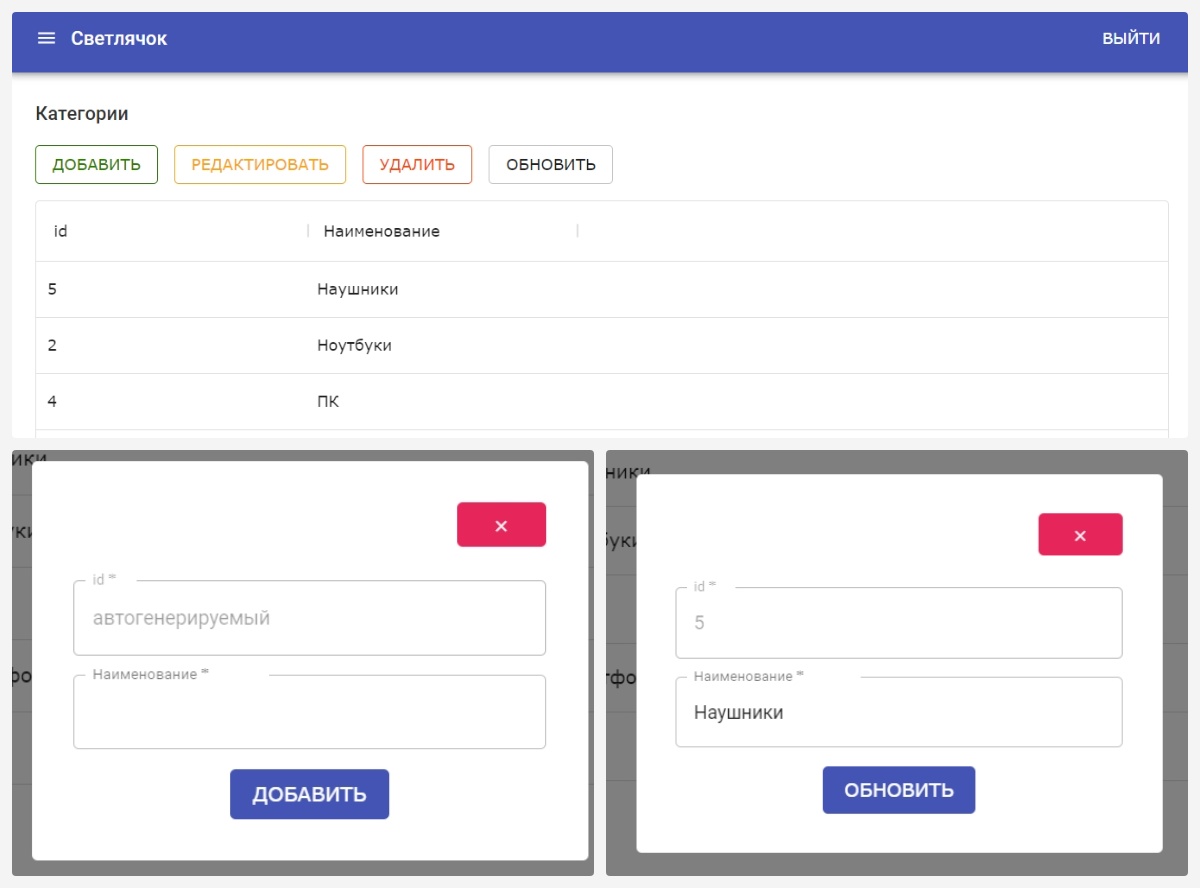
На рисунке 11 представлены формы для просмотра, добавления и редактирования категорий товаров.

Рисунок 11 – Формы для просмотра, добавления и редактирования категорий товаров.

На рисунке 12 представлены формы для просмотра, добавления и редактирования производителей товаров.

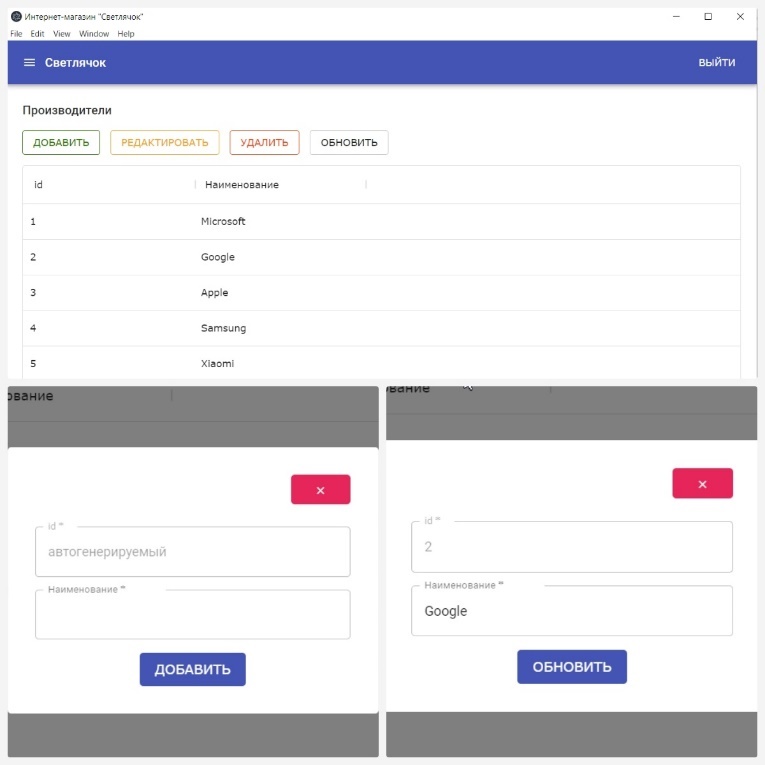


Рисунок 12 – Формы для просмотра, добавления и редактирования производителей товаров.

На рисунке 13 представлены формы для просмотра, добавления и редактирования товаров.

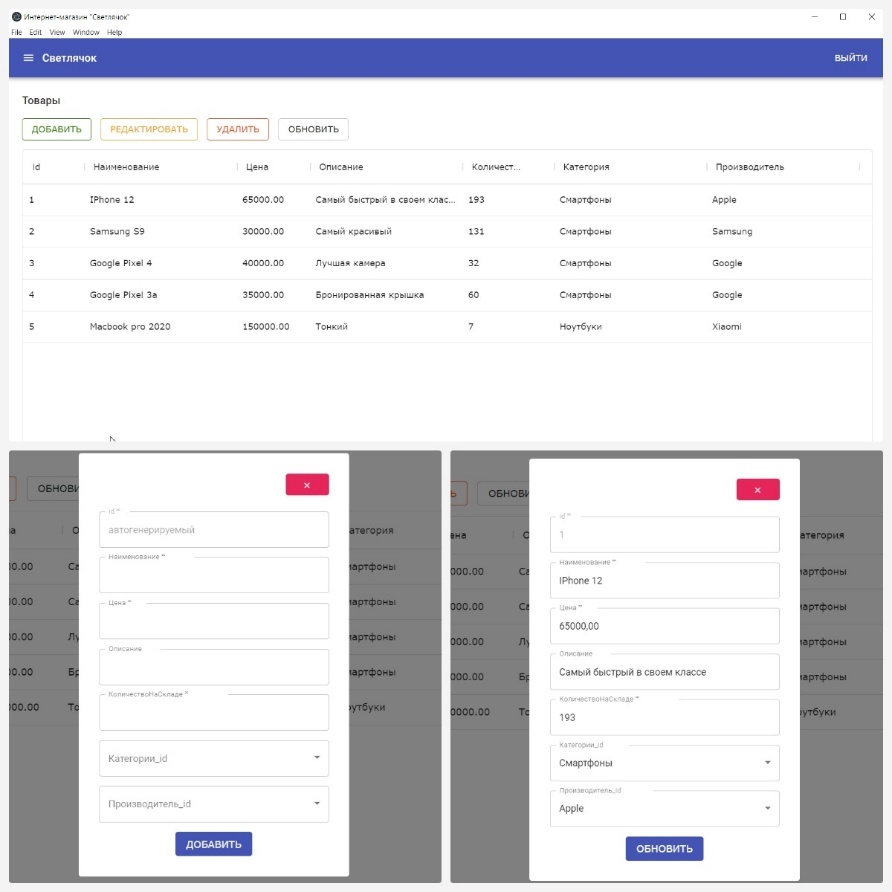


Рисунок 13 – Формы для просмотра, добавления и редактирования товаров.

На рисунке 14 представлены формы для просмотра, добавления и редактирования поставщиков и поставок.

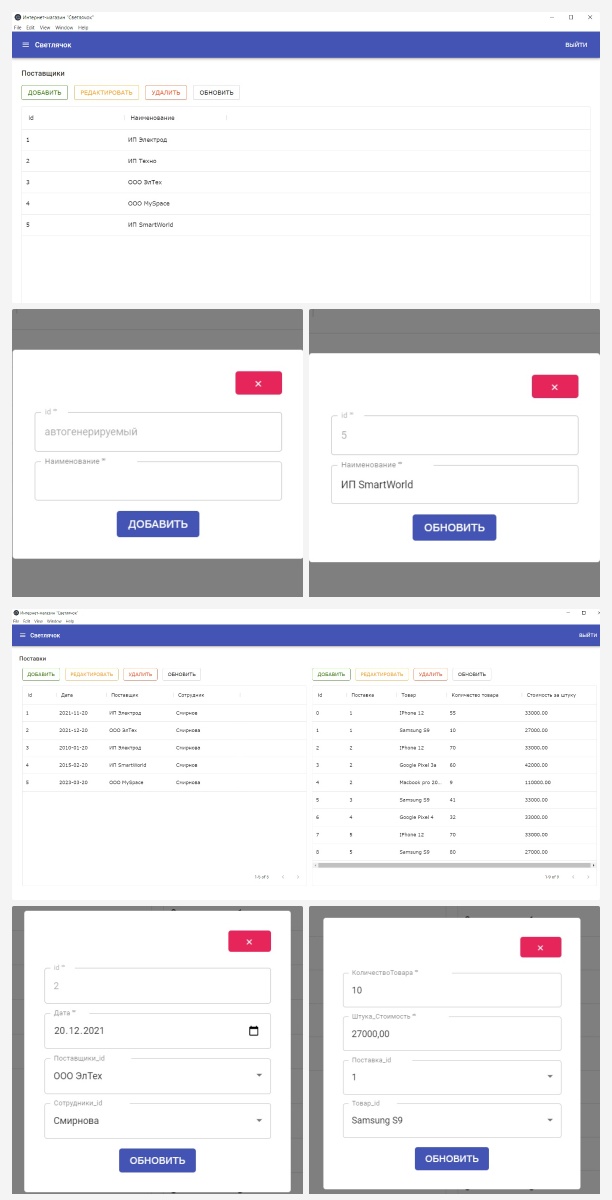


Рисунок 14 – Формы для просмотра, добавления и редактирования поставщиков и поставок.

На рисунке 15 представлены формы для просмотра, добавления и редактирования информации о сотрудниках и должностях.

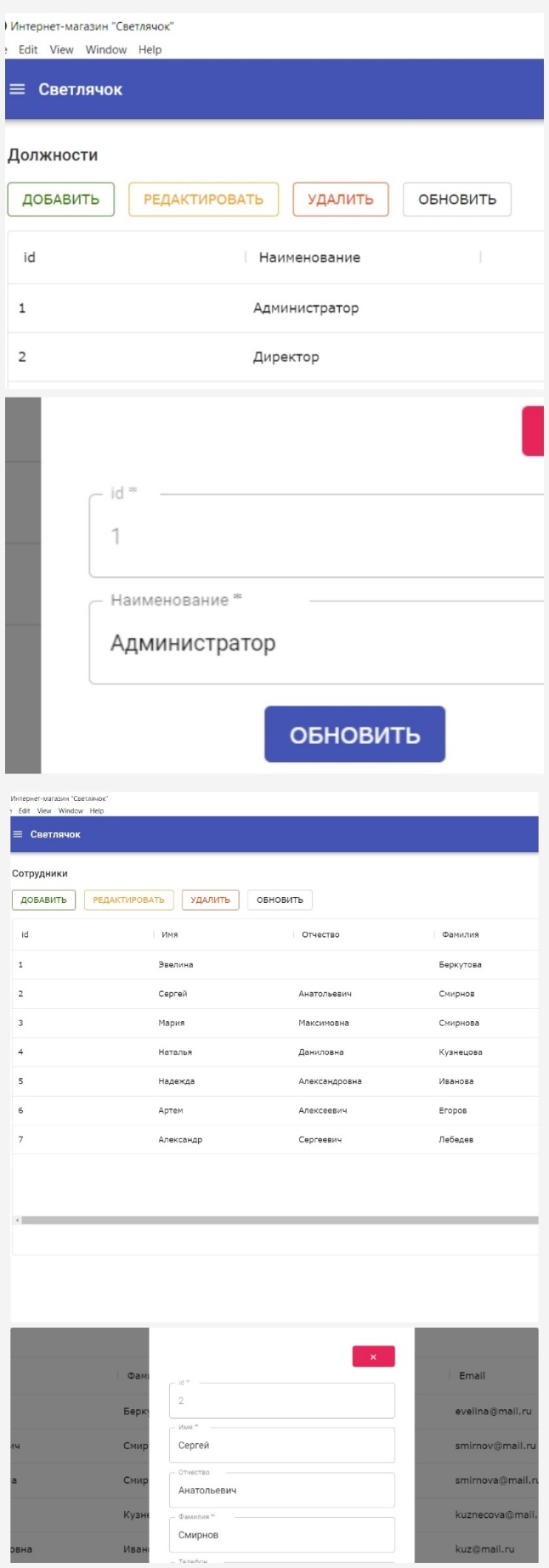


Рисунок 15 – Формы для просмотра, добавления и редактирования информации о сотрудниках и должностях.

На рисунке 16 представлены формы для просмотра, добавления и редактирования информации о клиентах.

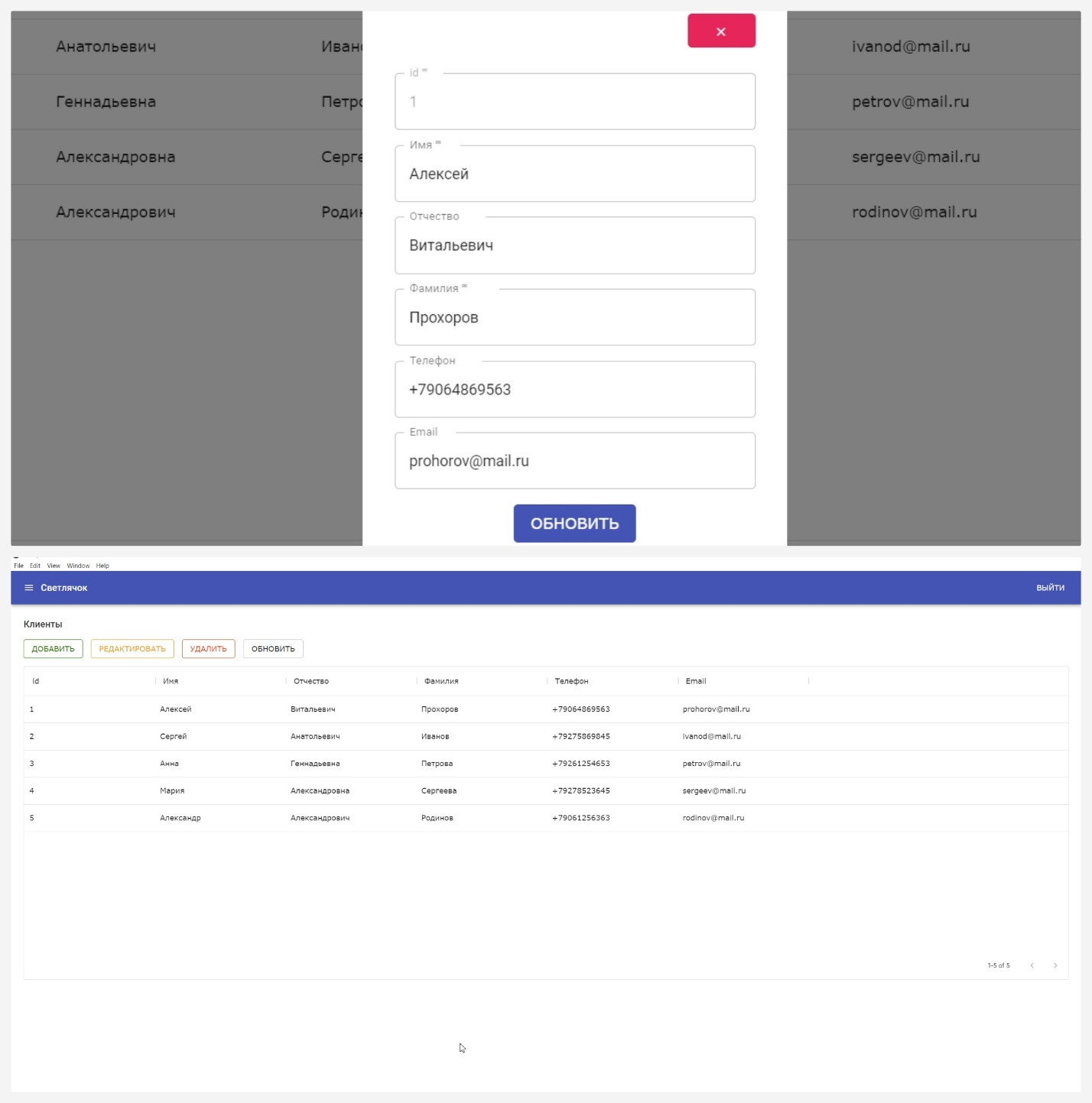


Рисунок 16 – Формы для просмотра, добавления и редактирования информации о клиентах.

На рисунке 17 представлены формы для просмотра, добавления и редактирования информации о сделках.

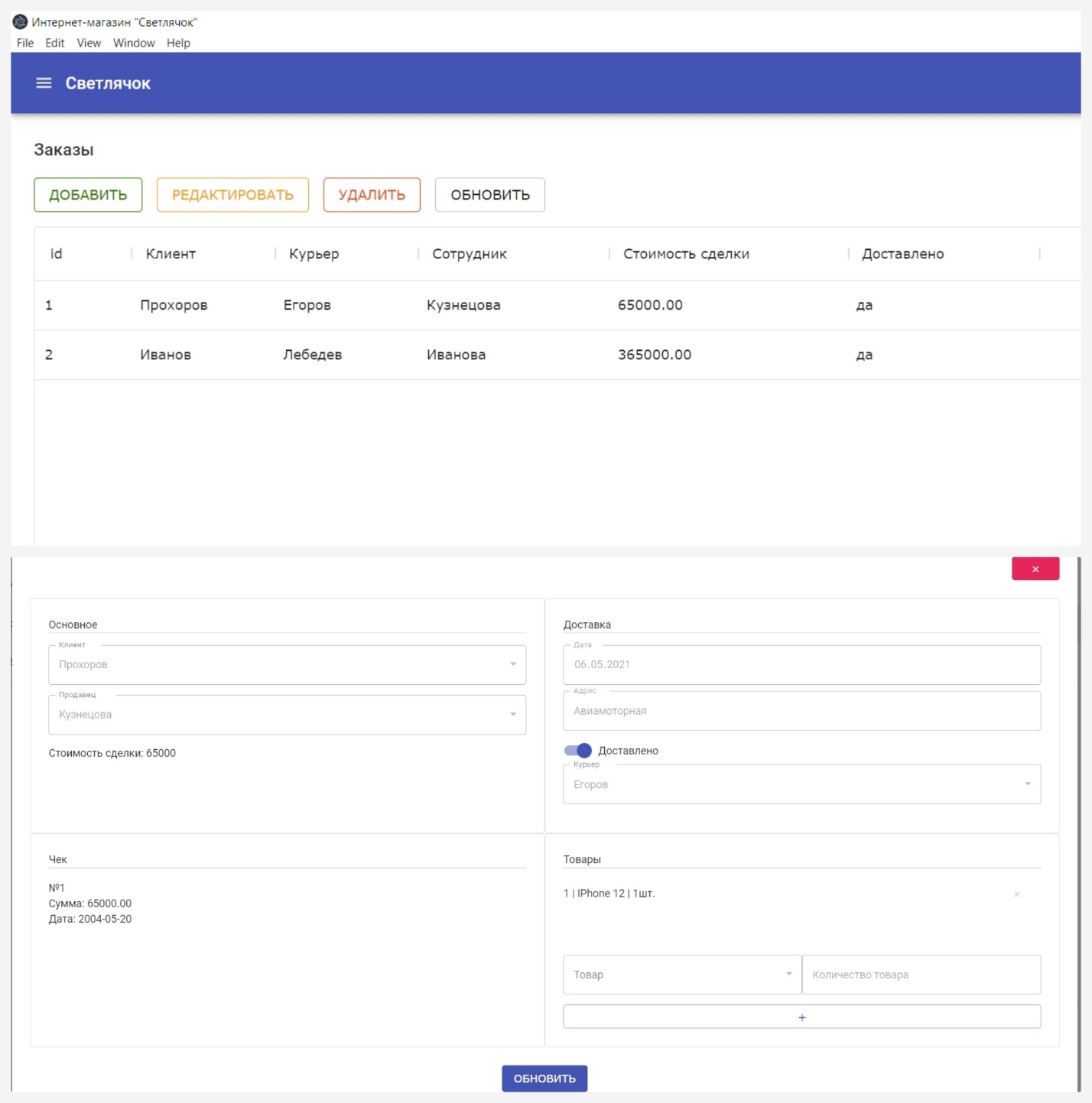


Рисунок 17 – Формы для просмотра, добавления и редактирования информации о сделках.

На рисунке 18 представлены формы отчетов.

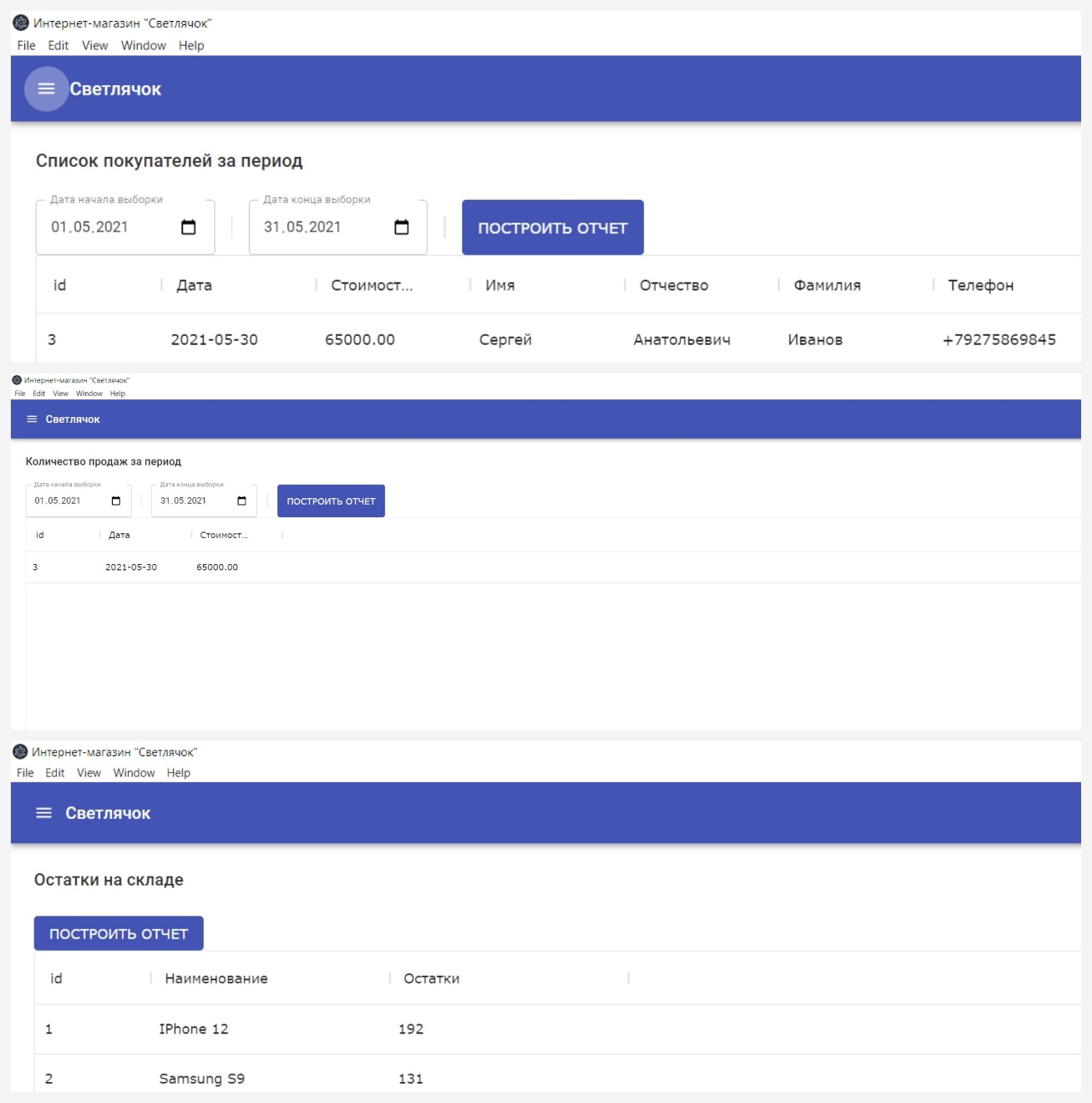


Рисунок 18 – Формы отчетов.

## Автоматизация обработки данных в БД

Для автоматизации обработки данных в БД были созданы предваряющие и завершающие триггеры, которые представлены в приложение №2.

Список триггеров:

* При добавлении сделки. В результате создается новая запись чека для сделки.
* При изменении сделки. В результате редактируется запись чека для сделки.
* При удалении сделки. В результате удаляется запись чека для сделки.
* При добавлении товара поставки. В результате изменяется количество товара на складе в таблице товары.
* При изменении товара поставки. В результате изменяется количество товара на складе в таблице товары.
* При удалении товара поставки. В результате изменяется количество товара на складе в таблице товары.
* При добавлении сделки. В результате изменяется количество товара на складе в таблице товары.
* При изменении сделки. В результате изменяется количество товара на складе в таблице товары.
* При удалении сделки. В результате изменяется количество товара на складе в таблице товары.

## Безопасность и контроль

В MySQL предусмотрен значительно усовершенствованный контроль доступа, ис­ключающий возможность несанкционированного проникновения в систему. Этот кон­троль базируется на пятиуровневой иерархии привилегий, которая определяет общие правила доступа при обработке запросов пользователей.

* Подключение к серверу производится только тогда, когда субъект доступа соблюдает правила доступа, которые заложены в основу системы привилегий MySQL. Эти правила могут быть заданы самими пользователями и/или узлом, для того чтобы ограничить доступ с внешних узлов, определенных подсетей, а также из оп­ределенной области IP-адресов. Подключение к серверу возможно только в одном случае - если пользователь введет правильный логин и пароль.
* Как только связь установлена, система начинает следить за каждым шагом поль­зователя, чтобы он не выходил за рамки своих привилегий. Пользователя можно ограничить, позволяя ему работать только с определенными базами данных. MySQL постоянно контролирует все запросы пользователя к базе данных, таблице или полю.

Защита системы значительно усовершенствована путем организации одностороннего кодирования пароля пользователя. Так, схема кодирования была существенно усовершенствована в MySQL 4.x. Новейшие версии MySQL поддерживают работу с SSL-протоколом, позволяющим для обеспечения лучшей защиты кодировать информацию, которая проходит через соединение клиент/сервер.

## Выводы

В данной главе произведено физическое проектирование базы данных магазина электронной техники «Светлячок». Определены ограничения. Произведен анализ безопасности в СУБД MySQL.

# Заключение

Курсовая работа посвящена разработке базы данных магазина электронной техники «Светлячок».

Решены следующие задачи:

* Произведен системный анализ предметной области;
* Построена инфологическая модель;
* Построена реляционная модель;
* Реализация базы данных в СУБД MySQL;
* Реализация форм для работы с БД с помощью JavaScript на nodeJS.

Разработана реляционная база данных, содержащая элементы автоматизации и обработки данных. В ее составе: 14 таблиц, 9 триггеров.

# Источники информации

1. Воронова Л.И. «Лабораторный практикум по дисциплине «базы данных» - Москва 2010г.
2. ГОСТ 2.105-95 Единая система конструкторской документации. Общие требования к текстовым документам
3. https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/ - статистика мировых интернет-продаж
4. https://www.amocrm.ru/vozmozhnosti-crm/ - amoCRM
5. https://1crm.ru/ - 1C: CRM
6. https://www.bitrix24.ru/ - Битрикс24
7. https://www.oracle.com/ru/database/technologies/ - Oracle Database
8. https://www.mysql.com/ - MySql
9. https://www.postgresql.org/ - PostgreSql
10. https://www.mongodb.com/ - MongoDB
11. https://habr.com/ru/company/ruvds/blog/324936/ - о видах баз данных

# Приложение №1. Программный код модуля для создания таблиц

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

-- Schema OnlineStore

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

CREATE SCHEMA IF NOT EXISTS `OnlineStore` DEFAULT CHARACTER SET utf8 ;

USE `OnlineStore` ;

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

-- Table `OnlineStore`.`Категории`

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

DROP TABLE IF EXISTS `OnlineStore`.`Категории` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Категории` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Наименование` VARCHAR(100) NOT NULL,

  PRIMARY KEY (`id`),

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

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

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Производители`

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

DROP TABLE IF EXISTS `OnlineStore`.`Производители` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Производители` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Наименование` VARCHAR(100) NOT NULL,

  `Описание` VARCHAR(500) NULL,

  PRIMARY KEY (`id`),

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

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

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Товары`

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

DROP TABLE IF EXISTS `OnlineStore`.`Товары` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Товары` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Наименование` VARCHAR(100) NOT NULL,

  `Цена` DECIMAL(10,2) NOT NULL,

  `Описание` VARCHAR(500) NULL,

  `КоличествоНаСкладе` INT NOT NULL,

  `Категории\_id` INT NOT NULL,

  `Производитель\_id` INT NOT NULL,

  PRIMARY KEY (`id`),

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

  INDEX `fk\_Товары\_Категории\_idx` (`Категории\_id` ASC) VISIBLE,

  INDEX `fk\_Товары\_Производитель1\_idx` (`Производитель\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_Товары\_Категории`

    FOREIGN KEY (`Категории\_id`)

    REFERENCES `OnlineStore`.`Категории` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Товары\_Производитель1`

    FOREIGN KEY (`Производитель\_id`)

    REFERENCES `OnlineStore`.`Производители` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Поставщики`

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

DROP TABLE IF EXISTS `OnlineStore`.`Поставщики` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Поставщики` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Наименование` VARCHAR(100) NOT NULL,

  PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Сотрудники`

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

DROP TABLE IF EXISTS `OnlineStore`.`Сотрудники` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Сотрудники` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Имя` VARCHAR(100) NOT NULL,

  `Отчество` VARCHAR(100) NULL,

  `Фамилия` VARCHAR(100) NOT NULL,

  `Телефон` VARCHAR(12) NULL,

  `Email` VARCHAR(100) NULL,

  `Login` VARCHAR(45) NOT NULL,

  `Password` VARCHAR(45) NULL,

  `Действителен` TINYINT NOT NULL,

  PRIMARY KEY (`id`),

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

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

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Поставки`

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

DROP TABLE IF EXISTS `OnlineStore`.`Поставки` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Поставки` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Дата` DATETIME NOT NULL,

  `Поставщики\_id` INT NOT NULL,

  `Сотрудники\_id` INT NOT NULL,

  PRIMARY KEY (`id`),

  INDEX `fk\_Поставки\_Поставщики1\_idx` (`Поставщики\_id` ASC) VISIBLE,

  INDEX `fk\_Поставки\_Сотрудники1\_idx` (`Сотрудники\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_Поставки\_Поставщики1`

    FOREIGN KEY (`Поставщики\_id`)

    REFERENCES `OnlineStore`.`Поставщики` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Поставки\_Сотрудники1`

    FOREIGN KEY (`Сотрудники\_id`)

    REFERENCES `OnlineStore`.`Сотрудники` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Клиенты`

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

DROP TABLE IF EXISTS `OnlineStore`.`Клиенты` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Клиенты` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Имя` VARCHAR(100) NOT NULL,

  `Отчество` VARCHAR(100) NULL,

  `Фамилия` VARCHAR(100) NOT NULL,

  `Телефон` VARCHAR(12) NULL,

  `Email` VARCHAR(100) NULL,

  PRIMARY KEY (`id`),

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

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Сделки`

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

DROP TABLE IF EXISTS `OnlineStore`.`Сделки` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Сделки` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Клиенты\_id` INT NOT NULL,

  `Сотрудники\_id` INT NOT NULL,

  `Стоимость\_сделки` DECIMAL(10,2) NOT NULL,

  `Дата` DATETIME NULL,

  PRIMARY KEY (`id`),

  INDEX `fk\_Сделки\_Клиенты1\_idx` (`Клиенты\_id` ASC) VISIBLE,

  INDEX `fk\_Сделки\_Сотрудники1\_idx` (`Сотрудники\_id` ASC) VISIBLE,

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

  CONSTRAINT `fk\_Сделки\_Клиенты1`

    FOREIGN KEY (`Клиенты\_id`)

    REFERENCES `OnlineStore`.`Клиенты` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Сделки\_Сотрудники1`

    FOREIGN KEY (`Сотрудники\_id`)

    REFERENCES `OnlineStore`.`Сотрудники` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Чек`

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

DROP TABLE IF EXISTS `OnlineStore`.`Чек` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Чек` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Сумма` DECIMAL(10,2) NOT NULL,

  `Дата` DATETIME NOT NULL,

  `Сделки\_id` INT NOT NULL,

  PRIMARY KEY (`id`),

  INDEX `fk\_Чек\_Сделки1\_idx` (`Сделки\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_Чек\_Сделки1`

    FOREIGN KEY (`Сделки\_id`)

    REFERENCES `OnlineStore`.`Сделки` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Должности`

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

DROP TABLE IF EXISTS `OnlineStore`.`Должности` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Должности` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Наименование` VARCHAR(100) NOT NULL,

  PRIMARY KEY (`id`),

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

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Должности\_Сотрудники`

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

DROP TABLE IF EXISTS `OnlineStore`.`Должности\_Сотрудники` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Должности\_Сотрудники` (

  `Должности\_id` INT NOT NULL,

  `Сотрудники\_id` INT NOT NULL,

  INDEX `fk\_Должности\_Сотрудники\_Должност\_idx` (`Должности\_id` ASC) VISIBLE,

  INDEX `fk\_Должности\_Сотрудники\_Сотрудни\_idx` (`Сотрудники\_id` ASC) VISIBLE,

  PRIMARY KEY (`Должности\_id`, `Сотрудники\_id`),

  CONSTRAINT `fk\_Должности\_Сотрудники\_Должности1`

    FOREIGN KEY (`Должности\_id`)

    REFERENCES `OnlineStore`.`Должности` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Должности\_Сотрудники\_Сотрудник1`

    FOREIGN KEY (`Сотрудники\_id`)

    REFERENCES `OnlineStore`.`Сотрудники` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Поставки\_Товары`

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

DROP TABLE IF EXISTS `OnlineStore`.`Поставки\_Товары` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Поставки\_Товары` (

  `Штука\_Стоимость` DECIMAL(10,2) NOT NULL,

  `КоличествоТовара` INT NOT NULL,

  `Поставка\_id` INT NOT NULL,

  `Товар\_id` INT NOT NULL,

  INDEX `fk\_Поставки\_Товары\_Поставки1\_idx` (`Поставка\_id` ASC) VISIBLE,

  PRIMARY KEY (`Поставка\_id`, `Товар\_id`),

  INDEX `fk\_Поставки\_Товары\_Товары1\_idx` (`Товар\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_Поставки\_Товары\_Поставки1`

    FOREIGN KEY (`Поставка\_id`)

    REFERENCES `OnlineStore`.`Поставки` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Поставки\_Товары\_Товары1`

    FOREIGN KEY (`Товар\_id`)

    REFERENCES `OnlineStore`.`Товары` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Сделки\_Товары`

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

DROP TABLE IF EXISTS `OnlineStore`.`Сделки\_Товары` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Сделки\_Товары` (

  `Сделки\_id` INT NOT NULL AUTO\_INCREMENT,

  `Товары\_id` INT NOT NULL,

  `КоличествоТовара` INT NOT NULL,

  `Штука\_Стоимость` INT NOT NULL,

  INDEX `fk\_Сделки\_has\_Товары\_Сделки1\_idx` (`Сделки\_id` ASC) VISIBLE,

  INDEX `fk\_Сделки\_has\_Товары\_Товары1\_idx` (`Товары\_id` ASC) VISIBLE,

  PRIMARY KEY (`Сделки\_id`, `Товары\_id`),

  CONSTRAINT `fk\_Сделки\_has\_Товары\_Сделки1`

    FOREIGN KEY (`Сделки\_id`)

    REFERENCES `OnlineStore`.`Сделки` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Сделки\_has\_Товары\_Товары1`

    FOREIGN KEY (`Товары\_id`)

    REFERENCES `OnlineStore`.`Товары` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `OnlineStore`.`Доставки`

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

DROP TABLE IF EXISTS `OnlineStore`.`Доставки` ;

CREATE TABLE IF NOT EXISTS `OnlineStore`.`Доставки` (

  `id` INT NOT NULL AUTO\_INCREMENT,

  `Дата` DATE NULL,

  `Адрес` VARCHAR(500) NOT NULL,

  `Доставлено` TINYINT NOT NULL,

  `Сотрудники\_id\_курьер` INT NOT NULL,

  `Сделки\_id` INT NOT NULL,

  PRIMARY KEY (`id`),

  INDEX `fk\_Доставки\_Сотрудники1\_idx` (`Сотрудники\_id\_курьер` ASC) VISIBLE,

  INDEX `fk\_Доставки\_Сделки1\_idx` (`Сделки\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_Доставки\_Сотрудники1`

    FOREIGN KEY (`Сотрудники\_id\_курьер`)

    REFERENCES `OnlineStore`.`Сотрудники` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Доставки\_Сделки1`

    FOREIGN KEY (`Сделки\_id`)

    REFERENCES `OnlineStore`.`Сделки` (`id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

USE `OnlineStore`;

DELIMITER $$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Сделки\_AFTER\_INSERT` $$

USE `OnlineStore`$$

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

BEGIN

  INSERT INTO Чек(Сумма, Дата, Сделки\_id) VALUES(NEW.Стоимость\_Сделки, NEW.Дата, NEW.id);

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Сделки\_AFTER\_UPDATE` $$

USE `OnlineStore`$$

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

BEGIN

  UPDATE Чек

    SET Сумма = NEW.Стоимость\_сделки,

        Сделки\_id = NEW.id

    WHERE Сделки\_id = OLD.id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Сделки\_BEFORE\_DELETE` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_BEFORE\_DELETE` BEFORE DELETE ON `Сделки` FOR EACH ROW

BEGIN

  DELETE FROM Чек WHERE Сделки\_id = OLD.id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Поставки\_Товары\_AFTER\_INSERT` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Поставки\_Товары\_AFTER\_INSERT` AFTER INSERT ON `Поставки\_Товары` FOR EACH ROW

BEGIN

  DECLARE lastValue int;

    SELECT КоличествоНаСкладе

    INTO lastValue FROM Товары

    WHERE id = NEW.Товар\_id;

  UPDATE Товары

    SET КоличествоНаСкладе = lastValue + NEW.КоличествоТовара

    WHERE id = NEW.Товар\_id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Поставки\_Товары\_AFTER\_UPDATE` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Поставки\_Товары\_AFTER\_UPDATE` AFTER UPDATE ON `Поставки\_Товары` FOR EACH ROW

BEGIN

  DECLARE lastValue int;

    SELECT КоличествоНаСкладе

    INTO lastValue FROM Товары

    WHERE id = NEW.Товар\_id;

  UPDATE Товары

    SET КоличествоНаСкладе = lastValue + (NEW.КоличествоТовара - OLD.КоличествоТовара)

    WHERE id = NEW.Товар\_id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Поставки\_Товары\_AFTER\_DELETE` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Поставки\_Товары\_AFTER\_DELETE` AFTER DELETE ON `Поставки\_Товары` FOR EACH ROW

BEGIN

    DECLARE lastValue int;

    SELECT КоличествоНаСкладе

    INTO lastValue FROM Товары

    WHERE id = OLD.Товар\_id;

  UPDATE Товары

    SET КоличествоНаСкладе = lastValue - OLD.КоличествоТовара

    WHERE id = OLD.Товар\_id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Сделки\_Товары\_AFTER\_INSERT` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_Товары\_AFTER\_INSERT` AFTER INSERT ON `Сделки\_Товары` FOR EACH ROW

BEGIN

  DECLARE lastValue int;

    SELECT КоличествоНаСкладе

    INTO lastValue FROM Товары

    WHERE id = NEW.Товары\_id;

  UPDATE Товары

    SET КоличествоНаСкладе = lastValue - NEW.КоличествоТовара

    WHERE id = NEW.Товары\_id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Сделки\_Товары\_AFTER\_UPDATE` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_Товары\_AFTER\_UPDATE` AFTER UPDATE ON `Сделки\_Товары` FOR EACH ROW

BEGIN

  DECLARE lastValue int;

    SELECT КоличествоНаСкладе

    INTO lastValue FROM Товары

    WHERE id = NEW.Товары\_id;

  UPDATE Товары

    SET КоличествоНаСкладе = lastValue - (NEW.КоличествоТовара - OLD.КоличествоТовара)

    WHERE id = NEW.Товары\_id;

END$$

USE `OnlineStore`$$

DROP TRIGGER IF EXISTS `OnlineStore`.`Сделки\_Товары\_AFTER\_DELETE` $$

USE `OnlineStore`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_Товары\_AFTER\_DELETE` AFTER DELETE ON `Сделки\_Товары` FOR EACH ROW

BEGIN

    DECLARE lastValue int;

    SELECT КоличествоНаСкладе

    INTO lastValue FROM Товары

    WHERE id = OLD.Товары\_id;

  UPDATE Товары

    SET КоличествоНаСкладе = lastValue + OLD.КоличествоТовара

    WHERE id = OLD.Товары\_id;

END$$

DELIMITER ;

SET SQL\_MODE = '';

DROP USER IF EXISTS root;

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

CREATE USER 'root' IDENTIFIED BY '0';

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

# Приложение №2. Программный код триггеров

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

BEGIN

INSERT INTO Чек(Сумма, Дата, Сделки\_id) VALUES(NEW.Стоимость\_Сделки, NEW.Дата, NEW.id);

END

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

BEGIN

UPDATE Чек

SET Сумма = NEW.Стоимость\_сделки,

Сделки\_id = NEW.id

WHERE Сделки\_id = OLD.id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_BEFORE\_DELETE` BEFORE DELETE ON `Сделки` FOR EACH ROW

BEGIN

DELETE FROM Чек WHERE Сделки\_id = OLD.id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_Товары\_AFTER\_INSERT` AFTER INSERT ON `Сделки\_Товары` FOR EACH ROW

BEGIN

DECLARE lastValue int;

SELECT КоличествоНаСкладе

INTO lastValue FROM Товары

WHERE id = NEW.Товары\_id;

UPDATE Товары

SET КоличествоНаСкладе = lastValue - NEW.КоличествоТовара

WHERE id = NEW.Товары\_id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_Товары\_AFTER\_UPDATE` AFTER UPDATE ON `Сделки\_Товары` FOR EACH ROW

BEGIN

DECLARE lastValue int;

SELECT КоличествоНаСкладе

INTO lastValue FROM Товары

WHERE id = NEW.Товары\_id;

UPDATE Товары

SET КоличествоНаСкладе = lastValue - (NEW.КоличествоТовара - OLD.КоличествоТовара)

WHERE id = NEW.Товары\_id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Сделки\_Товары\_AFTER\_DELETE` AFTER DELETE ON `Сделки\_Товары` FOR EACH ROW

BEGIN

DECLARE lastValue int;

SELECT КоличествоНаСкладе

INTO lastValue FROM Товары

WHERE id = OLD.Товары\_id;

UPDATE Товары

SET КоличествоНаСкладе = lastValue + OLD.КоличествоТовара

WHERE id = OLD.Товары\_id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Поставки\_Товары\_AFTER\_INSERT` AFTER INSERT ON `Поставки\_Товары` FOR EACH ROW

BEGIN

DECLARE lastValue int;

SELECT КоличествоНаСкладе

INTO lastValue FROM Товары

WHERE id = NEW.Товар\_id;

UPDATE Товары

SET КоличествоНаСкладе = lastValue + NEW.КоличествоТовара

WHERE id = NEW.Товар\_id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Поставки\_Товары\_AFTER\_UPDATE` AFTER UPDATE ON `Поставки\_Товары` FOR EACH ROW

BEGIN

DECLARE lastValue int;

SELECT КоличествоНаСкладе

INTO lastValue FROM Товары

WHERE id = NEW.Товар\_id;

UPDATE Товары

SET КоличествоНаСкладе = lastValue + (NEW.КоличествоТовара - OLD.КоличествоТовара)

WHERE id = NEW.Товар\_id;

END

CREATE DEFINER = CURRENT\_USER TRIGGER `OnlineStore`.`Поставки\_Товары\_AFTER\_DELETE` AFTER DELETE ON `Поставки\_Товары` FOR EACH ROW

BEGIN

DECLARE lastValue int;

SELECT КоличествоНаСкладе

INTO lastValue FROM Товары

WHERE id = OLD.Товар\_id;

UPDATE Товары

SET КоличествоНаСкладе = lastValue - OLD.КоличествоТовара

WHERE id = OLD.Товар\_id;

END