

Проектирование базы данных для информационной системы сбыта кондитерского цеха

Уровень развития информационных технологий заставляет задуматься большинство организаций о внедрении информационных систем. Основопологающим компонентом любой информационной системы выступает база данных. Компания Gartner Groups и IDC с определенной периодичностью проводят исследования IT-проектов. Данные исследования говорят о том, что около половины проектов завершаются с плохим результатом в области проектирования баз данных [7; с.6].

База данных – сложный объект, который подлежит тщательной проработке. Проектирование базы данных выступает слабо структурированным и весьма размытым этапом. Грамотная постановка задачи проектирования базы данных поможет успешно завершить проект в заданные сроки.

Информационные системы с включением в их состав баз данных активно внедряются сейчас на производственные предприятия, в том числе на базе кондитерских цехов. Это помогает обеспечить конкурентоспособность на данном рынке и успешно реализовывать свою продукцию.

Проектируемая база данных предназначена для использования в кондитерском цехе «Благодатное утро», расположенном в г. Курск. Цех был основан в 2013 году и изначально специализировался на выпуске песочного печенья. В связи с постоянно увеличивающимся объемом заказов возникла необходимость внедрения информационной системы с проектированием базы данных. С ее помощью упростится заказ продукции и оплата заказов клиентами, работа менеджеров, сотрудников склада и курьеров.

Анализ предметной области – первая задача, которую необходимо решить при проектировании базы данных. Ниже приведено описание предметной области и поставки заказа продукции кондитерского цеха.

Процесс сбыта продукции начинается с получения заказа. Предприятие ведет оптовую продажу продукции, поэтому основными клиентами являются юридические лица. Прием заказов осуществляется по телефону или по электронной почте. В процессе приема заказа менеджером обговариваются с клиентом условия поставки. Далее менеджером оформляется договор на поставку продукции, который высылается по электронной почте в двух экземплярах, один из которых клиент должен подписать и вернуть компании.

Если сумма заказа больше 30 000 рублей, то клиент вносит предоплату. Предоплата и оплата заказа осуществляется путем перевода денежных средств на расчетный счет.

Далее осуществляется сбор заказа на одном из складов предприятия, где сотрудники проверяют наличие указанной в заказе продукции на складе. Если необходимой продукции

нет в наличии на складе, то данный заказ передается другому складу. Сотрудники склада учитывают эту информацию и отправляют заявку на заказ отсутствующей продукции в цех. Также оформляется товарная накладная, один экземпляр которой остается на складе, а другой передается клиенту вместе с заказом.

После того, как заказ был собран, он передается на доставку. Доставку осуществляет сторонняя транспортная организация. Менеджер кондитерского цеха с определенной периодичностью получает информацию от транспортной компании о доставке. Также со стороны клиента менеджеру должна поступить информация о получении или неполучении заказа. Если все прошло успешно, то на этом заканчивается данный бизнес-процесс. Если нет, то менеджер должен ожидать, пока транспортная компания доставит заказ клиенту.

Модель бизнес-процесса в нотации BPMN представлена на рисунке 1.

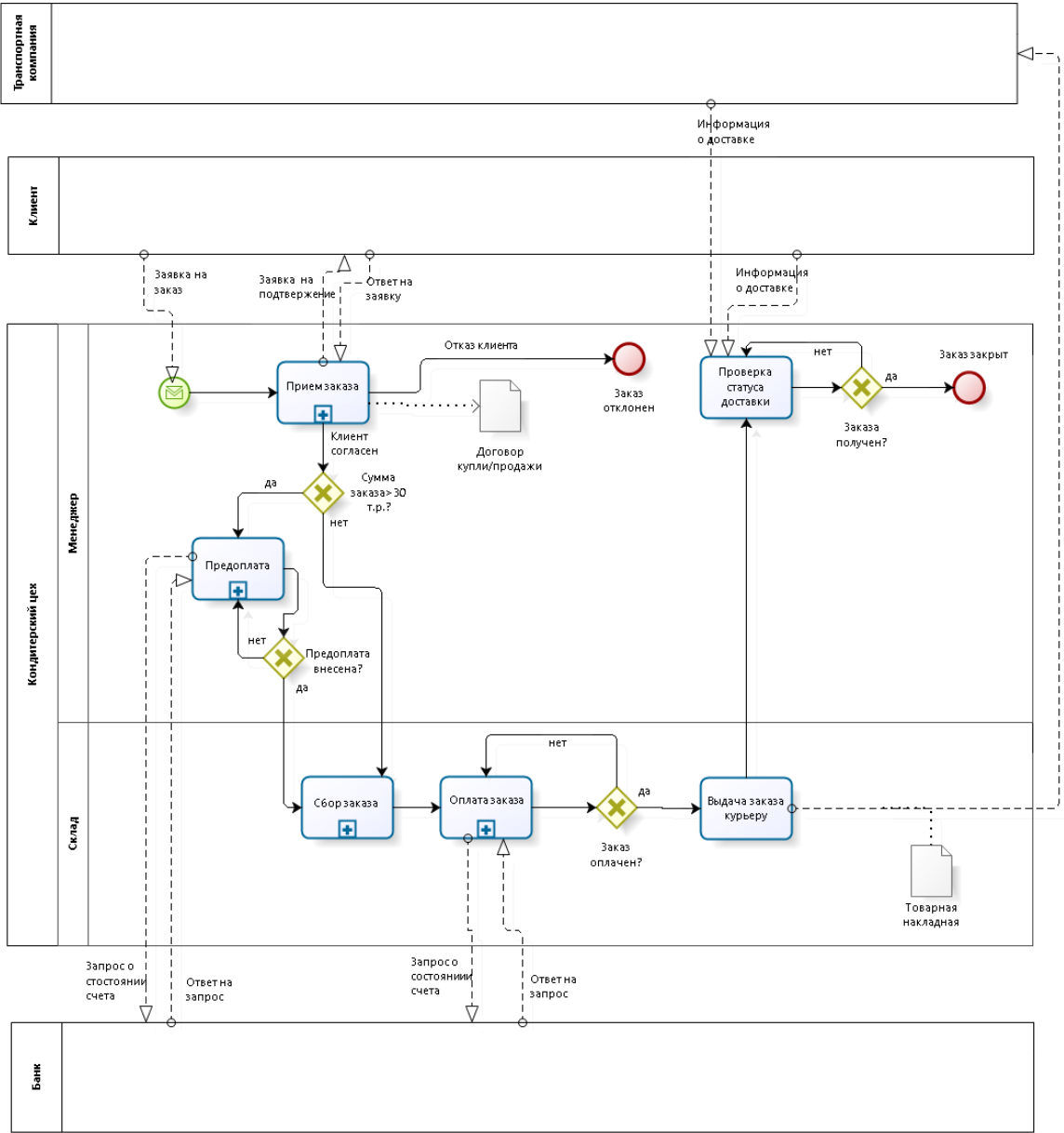


Рисунок 1 – Модель бизнес-процесса «Заказ продукции»

На рисунке 2 представлен подпроцесс «Прием заказа».

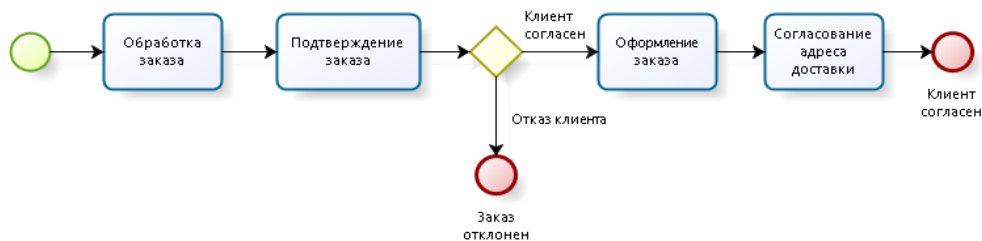


Рисунок 1 – Диаграмма подпроцесса «Прием заказа»

Разрабатываемая база данных предназначена для хранения информации о продукции, работниках и клиентах предприятия, данных о заказе, доставке и складе.

В соответствии с информационными требованиями пользователя база данных должна обеспечивать ввод, хранение, выбор и модификацию следующих данных:

- наименование и цена продукции;
- дата, сумма заказа и список товаров в заказе;
- список заказов, принадлежащий клиенту;
- информация о клиенте;
- информация о складе и поступлении продукции на склад;
- информация о доставке
- информация о сотруднике отдела продаж и склада.

Основными этапами при проектировании баз данных выступает: инфологическое, логическое и физическое проектирование [4; с.84].

Цель инфологического этапа проектирования состоит в получении семантических (смысловых) моделей, отражающих информационное содержание проблемы [5; с.54]. Одна из наиболее популярных семантических моделей данных – модель "сущность-связь" (Entity-Relationship Model, ER-model).

Проведенный обзор и семантический анализ рассматриваемой предметной области позволил выделить семь типов сущностей:

- клиент,
- заказ,
- продукция,
- склад,
- доставка,
- сотрудник.

В таблице 1 представленные вышеперечисленные типы сущностей и атрибуты, используемые для их описания.

Таблица 1 – Типы сущностей и их атрибутов

Сущность	Атрибут
Клиент	Код клиента Наименование предприятия Адрес Телефон Расчетный счет
Заказ	Код заказа Дата заказа
Продукция	Код продукции Наименование Цена Единица измерения Количество
Склад	Код склада Адрес Телефон
Доставка	Код доставки Дата доставки
Сотрудник	Код сотрудника ФИО Дата рождения Отдел Должность Зарплата Пол

Далее необходимо определить все возможные виды связей между выделенными выше типами сущностей. Все связи, установленные между выделенными типами сущностей, приведены в таблице 2.

Таблица 2 – Связи между выделенными типами сущностей

Сущность А	Связь	Сущность Б	Тип связи
Продукция	включается/содержит	Заказ	М:М
Клиент	оформляет	Заказ	1:М
Клиент	получает	Доставка	1:М
Заказ	включается в	Доставка	1:М
Склад	принимает/поступает	Продукция	М:М
Сотрудник	собирает	Заказ	1:М
Сотрудник	оформляет	Доставка	1:М

Теперь, когда определены типы сущностей и установлены связи между ними, строится неформальная модель типа «сущность – связь» с использованием конструктивных элементов сущность, атрибут и связь. Эта модель представлена на рисунке 3.

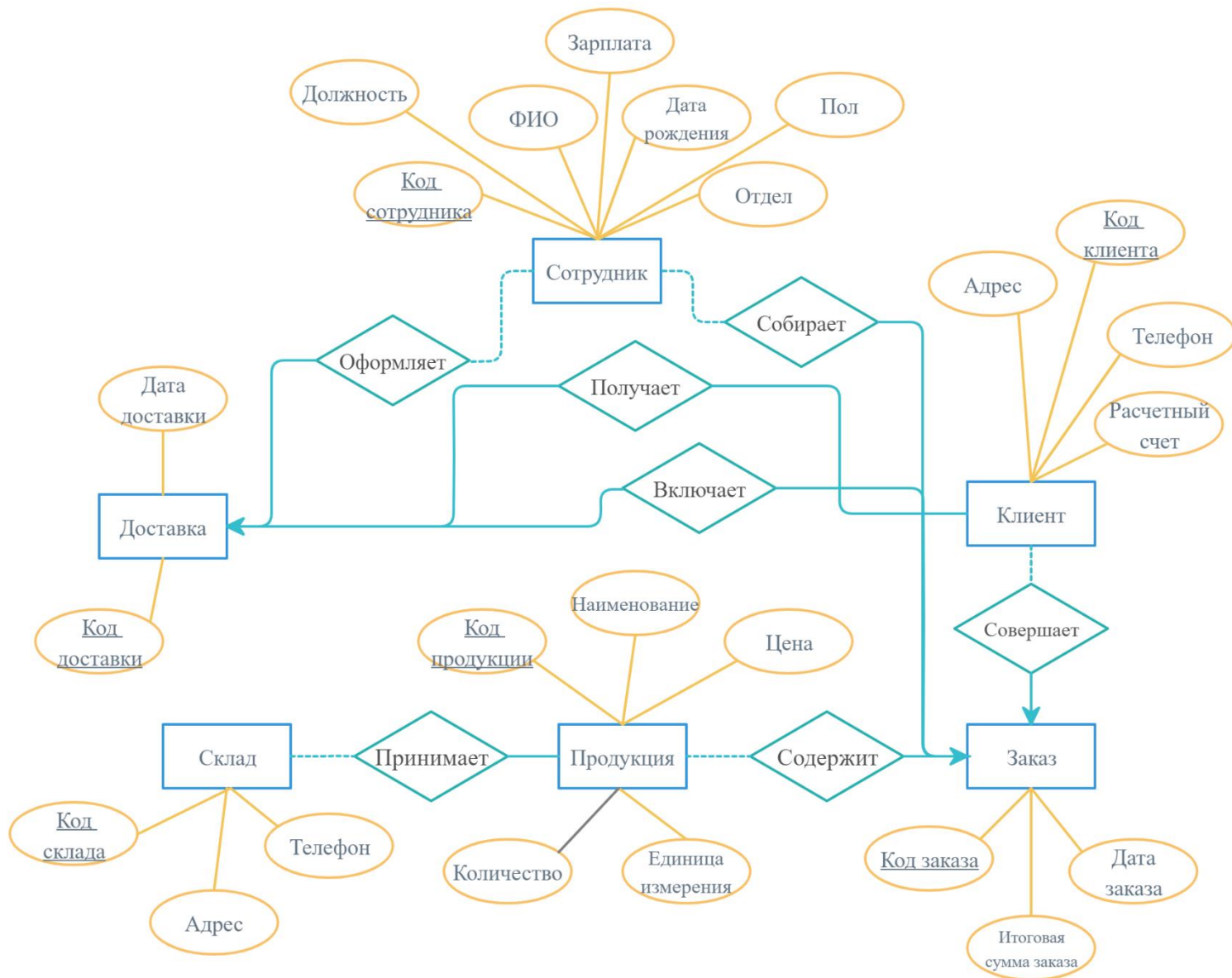


Рисунок 3 – Модуль «сущность-связь» предметной области

На этапе логического проектирования осуществляется переход от инфологической модели предметной области к логической структуре организации базы данных в соответствии с выбранной моделью данных. Модель данных формализовано описывает все множество допустимых логических структур данных.

Различают три основных типа модели данных:

- иерархическая модель данных,
- сетевая модель данных,
- реляционная модель данных.

Исторически первой моделью считается иерархическая модель данных. Пример такой модели изображен на рисунке 4, где П, П1 - П3 – преподаватели, а К1-К3 – курсы. Данные в такой модели организованы в виде структуры (иерархии). Основными компонентами такой модели выступают поле, сегмент (запись), связь [6; с.32].

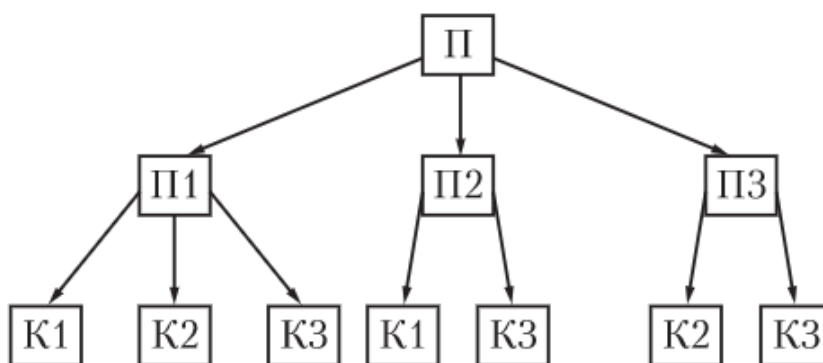


Рисунок 4 – Иерархическая модель данных

Поле данных – минимальная именованная область данных, которая доступна пользователю СУБД. Сегмент (запись) – основная единица обработки базы данных. Связь указывает на иерархическое отношение между записями двух типов. В модели имеется одна запись, которая является входом в данную структуру или корнем дерева. Все остальные записи имеют ровно одну вышестоящую вершину(предка) и любое число подчиненных вершин (потомков).

Сетевой подход считается расширением иерархического подхода (рис.5). Если в иерархическом подходе запись-потомок должна была иметь строго одного предка, то при сетевом подходе запись-потомок может иметь сколь угодно число предков. Основные элементы сетевой модели: элемент данных, агрегат данных, запись (группа), набор (групповое отношение).

Элемент данных – минимальная именованная единица данных, которая доступна пользователю посредством СУБД. Агрегат данных – именованная совокупность элементов,

которую можно разбивать на части. Запись – агрегат, который не входит в состав какого-либо другого агрегата и выступает в качестве единицы для обработки в базе данных. Набор в данной модели отвечает за иерархическое отношение между двумя типами записей.

Недостаток иерархической модели по сравнению с сетевой заключается в том, что в сетевой модели реализован более удобный способ реализации отношения «многие-ко-многим». В иерархической модели с этой целью приходится вводить два типа записей и частично дублировать значения.

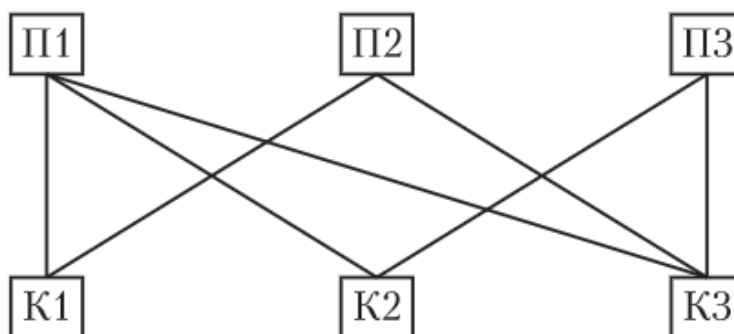


Рисунок 5 – Сетевая модель данных

Общими недостатками данных моделей являются:

- сложности в использовании;
- требуются знания о физической организации базы данных;
- прикладные системы зависят от организации базы данных;
- сложность логики из-за деталей организации доступа.

В любом случае имеются и достоинства данных моделей:

- развитые средства управления данными во внешней памяти на низком уровне;
- возможность построения эффективных прикладных систем вручную;
- возможность рационального использования памяти за счет разделения подобъектов (в сетевых системах).

В реляционной модели данные представлены в виде таблиц значений данных и отношений этих таблиц между собой. Реляционная модель данных представлена на рисунке 6, где НК – номер курса, а НП – номер преподавателя.

Математический термин отношение в данном случае определяется следующим образом. Пусть даны N множеств D_1, D_2, \dots, D_N . Отношение R над этими множествами называется множество упорядоченных N -кортежей вида $\langle d_1, d_2, \dots, d_N \rangle$, где $d_1 \in D_1, \dots, d_N \in D_N$. ($N \geq 1$). Множества D_1, D_2, \dots, D_N называются доменами отношения R .

Исходя из этого определения можно сопоставить различные терминологии следующим образом: отношение – таблица, упорядоченный кортеж – строка таблицы, атрибут – столбец таблицы. С точки зрения программной терминологии: отношение – файл, упорядоченный кортеж – запись в файле, атрибут – поле записи.

ПРЕП.	ЧИТАЕТ		КУРС
НП	НП	НК	НК
П1	П1	К1	К1
П2	П1	К2	К2
П3	П1	К3	К3
	П2	К1	
	П2	К3	
	П3	К2	
	П3	К3	

Рисунок 6 – Реляционная модель данных

Количество атрибутов в кортеже, иными словами число столбцов в таблице, называется степенью отношения. Текущее число кортежей или строк, называется мощностью отношения и обозначается $|R/|$. Степень отношения выступает постоянной переменной после создания отношения. Мощность же отношения может изменяться при добавлении или удалении кортежей. Схемой отношения R называется перечень атрибутов A_i данного отношения с указанием домена D_i , к которому они относятся [1; с.24]:

$$S_R = (A_1, A_2, \dots, A_N), \text{ где } A_i \subseteq D_i, 1 \leq i \leq N.$$

С целью однозначной идентификации определенного кортежа используется первичный ключ. Первичный ключ – атрибут, или набор оптимального числа атрибутов, который может однозначно идентифицировать определенный кортеж.

Таблицы в реляционной базе данных взаимосвязаны друг с другом. При этом одна таблица выступает главной, а другая подчиненной. Связь осуществляется посредством первичного ключа главной таблицы и внешнего ключа подчиненной таблицы. Внешний ключ – атрибут или набор атрибутов, включаемых в состав подчиненной таблицы и выступающих в качестве первичного ключа в главной таблице.

В файле, в котором хранится отношение, могут храниться множество записей. Для ускорения доступа файл можно индексировать. Иногда в качестве индексного ключа используется первичный ключ.

Достоинства реляционной модели данных:

- данные представлены в наиболее удобной и естественной форме;

- реляционная модель данных базируется на строгом математическом аппарате;
- полная независимость данных;
- высокий уровень защиты и разграничение прав доступа;
- изменение данных без сложной физической реорганизации хранилища;
- наличие декларативного языка запросов.

Недостатки реляционной модели данных:

- на начальной стадии проектирования иногда искусственно приходится описывать предметную область с помощью отношений;
- строится на основе учета зависимостей, но не имеет описания этих зависимостей;
- базируется на выделении из предметной области главных объектов, их свойств и связей между ними, но не имеет четкого аппарата для их выявления;
- низкая скорость доступа к данным.

Но с учетом ограничений, которые были рассмотрены выше, реляционная модель данных получила достаточно широкое распространение. СУБД с использованием данной модели позволяют проектировать базы данных практически любой сложности. Реляционная модель обладает гибкой структурой по сравнению с другими. В сетевой и иерархической модели структура базы данных лежит в самом приложении. Поэтому изменение структуры требует реорганизации самого приложения.

Таким образом, для дальнейшего проектирования базы данных была выбрана реляционная модель данных.

При переходе на логическое проектирование необходимо создать схему базы данных. Схема базы данных получается путем преобразования ER-диаграммы на основе определённого набора правил.

Связь «один-ко-многим» между сущностями «Клиент» и «Заказ», «Клиент» и «Доставка», «Доставка» и «Заказ», «Сотрудник» и «Заказ», «Сотрудник» и «Доставка» реализуется через внешний ключ.

Связь между «Продукция» и «Заказ», а также «Склад» и «Продукция» принадлежит к типу «многие-ко-многим». Этот тип связи реализуется через вспомогательные отношения «Список товаров в заказе» и «Поступление продукции на склад» соответственно. Схема реляционной базы данных, полученная из ER-диаграммы, представлена на рисунке 7.

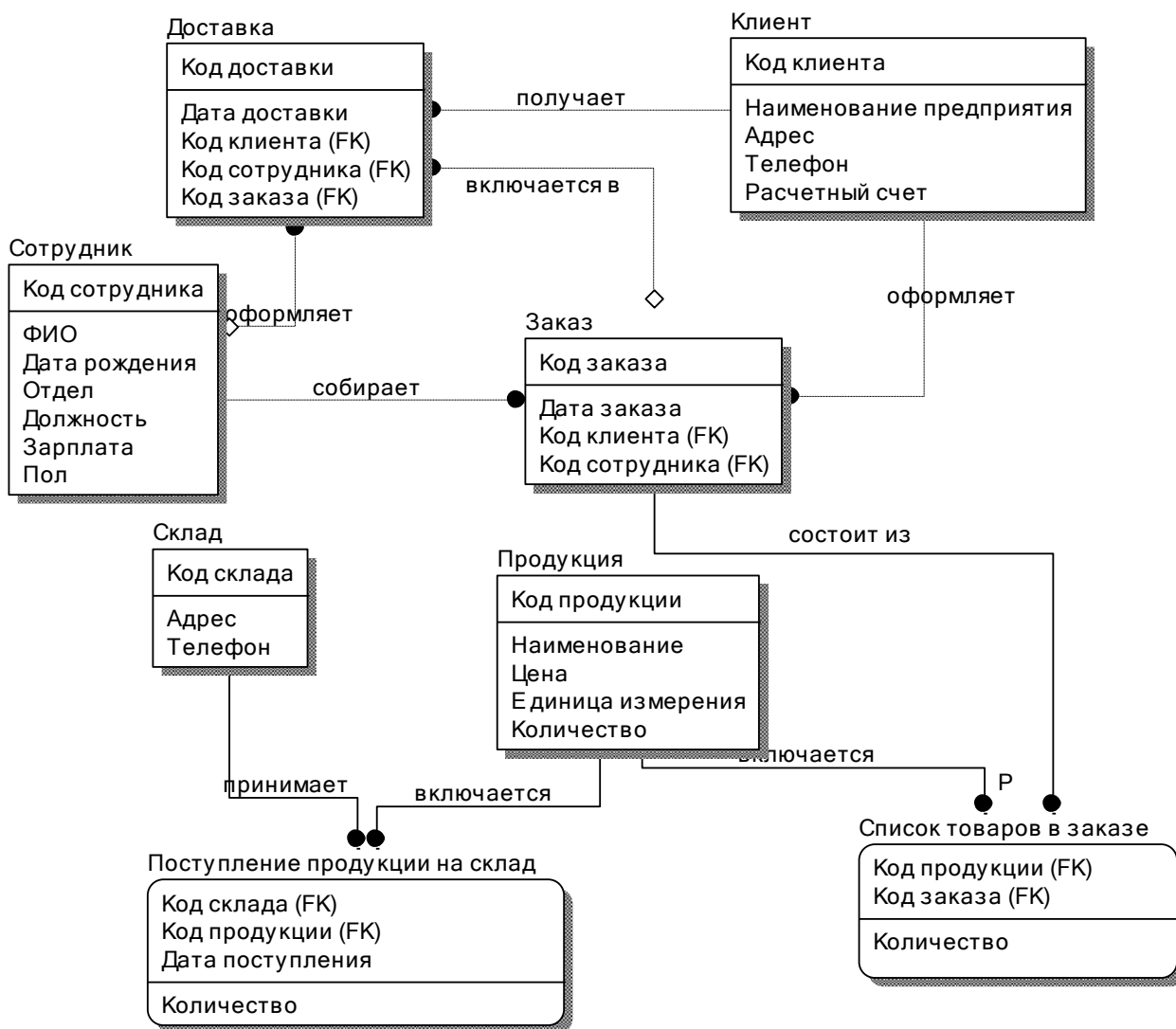


Рисунок 7 – Ненормализованная схема реляционной базы данных «Кондитерский цех»

В основе логического проектирования реляционной базы данных лежит метод нормализации отношений.

При нормализации происходит процесс преобразования структуры из одной нормальной формы в другую [3; с. 25]. В данной работе ER-модель приводится к следующим нормальным формам:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- нормальная форма Бойса-Кодда (НФБК).

При приведении к 1НФ были выявлены неатомарные атрибуты в сущностях «Сотрудник», «Клиент», «Склад».

На рисунках 8-10 приведены данные ненормализованные сущности и результаты их приведения к первой нормальной форме.

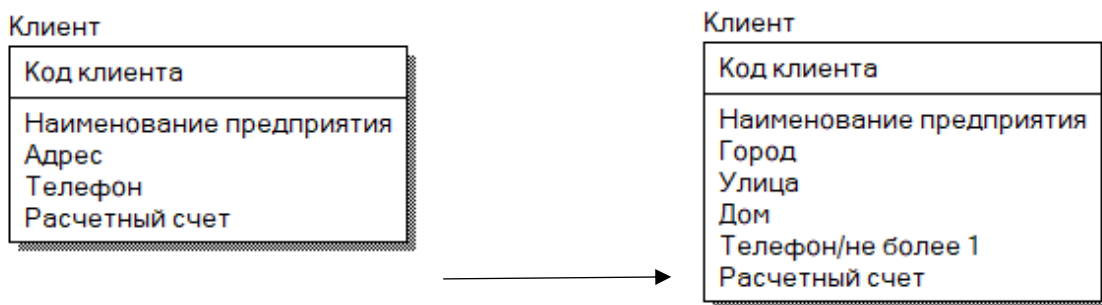


Рисунок 8 – Приведение к первой нормальной форме сущности «Клиент»

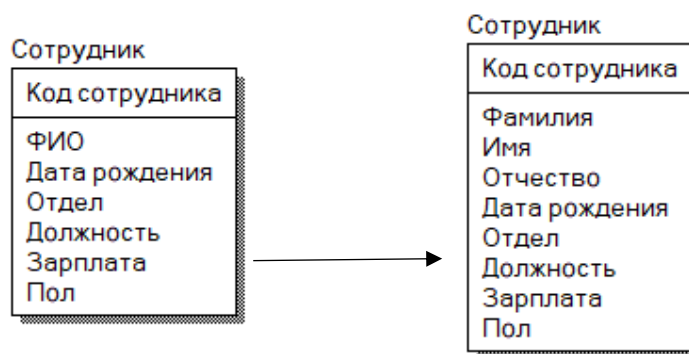


Рисунок 9 – Приведение к первой нормальной форме сущности «Сотрудник»

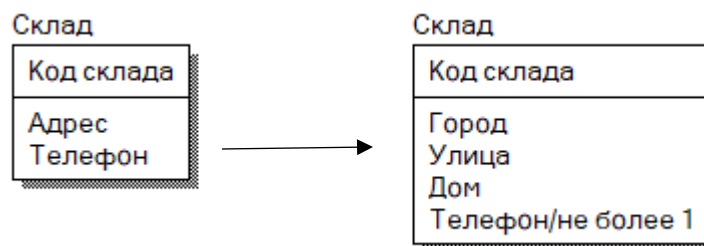


Рисунок 10 – Приведение к первой нормальной форме сущности «Склад»

При приведении ко второй нормальной форме никаких преобразований осуществлять не надо. Составной первичный ключ имеют две сущности «Поступление продукции на склад», «Список товаров в заказе». Все неключевые атрибуты данных сущностей функционально полно зависят от первичного ключа.

При приведении к 3НФ была выявлена следующая аномалия. У сущности сотрудник есть зависимость между неключевыми атрибутами «Должность» и «Зарплата», т.к. зарплата зависит от первичного ключа «Код сотрудника» транзитивно через посредника «Должность» (рис.11).



Рисунок 11 – Приведение к третьей нормальной форме сущности «Сотрудник»

При приведении к нормальной форме Бойса-Кодда не было выявлено никаких отклонений. Таким образом, после нормализации схема данных примет вид, представленный на рисунке 12.

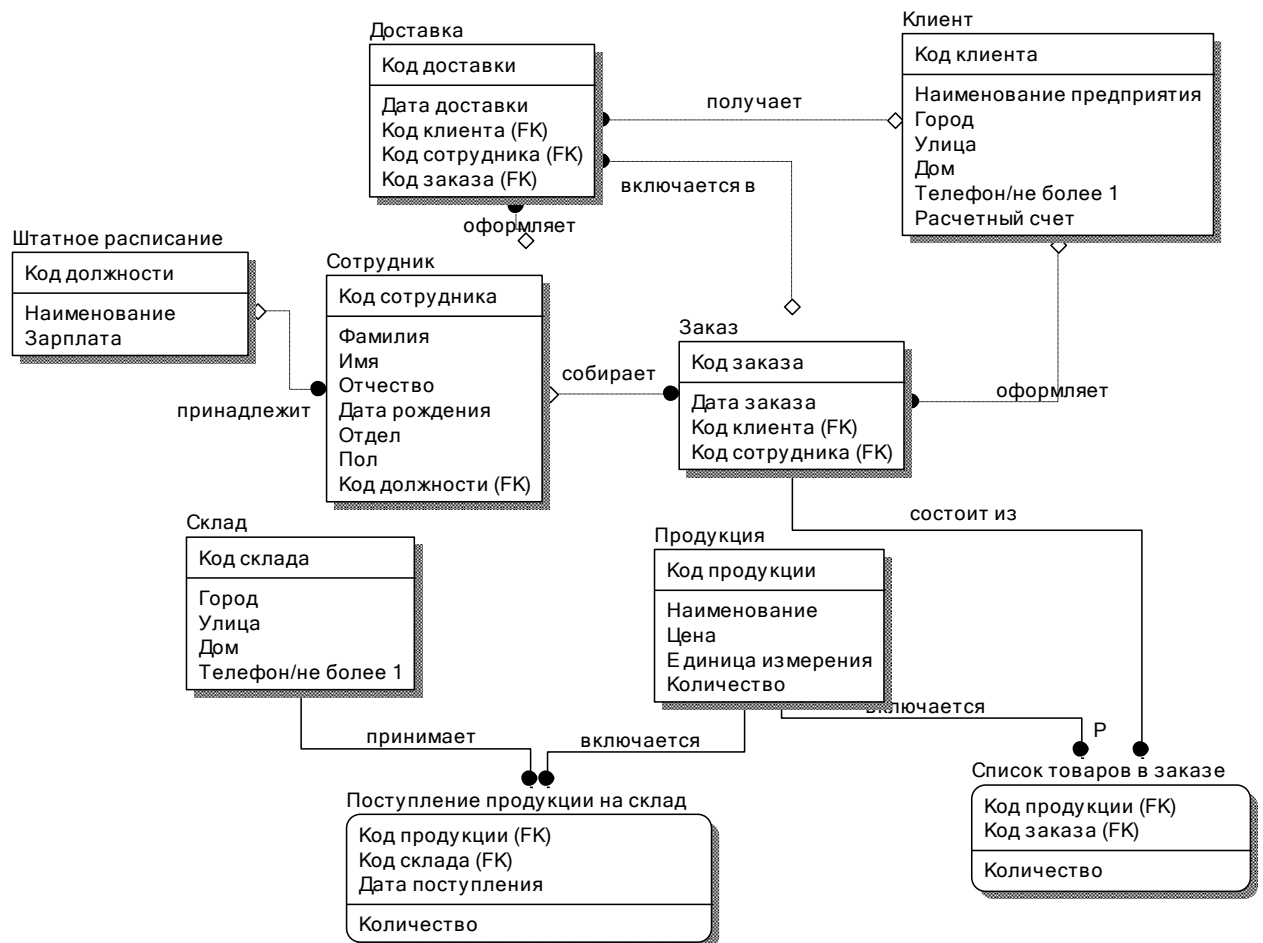


Рисунок 12 – Нормализованная схема реляционной базы данных

После логического проектирования необходимо осуществить переход к физическому проектированию. Физическое проектирование всегда ориентировано на конкретное СУБД. В качестве СУБД была выбрана система MySQL 8.0.

MySQL - это система управления реляционными базами данных с открытым исходным кодом при поддержке компании Oracle [2; с. 19].

Перед тем, как перейти к непосредственной реализации базы данных необходимо построить СУБД-зависимую модель данных. В данной модели наименования таблиц и полей должны быть записаны на латинице. Также должен быть определён тип и размер данных с учетом выбранной СУБД.

Для данной предметной области СУБД-зависимая модель примет вид, представленный на рисунке 13.

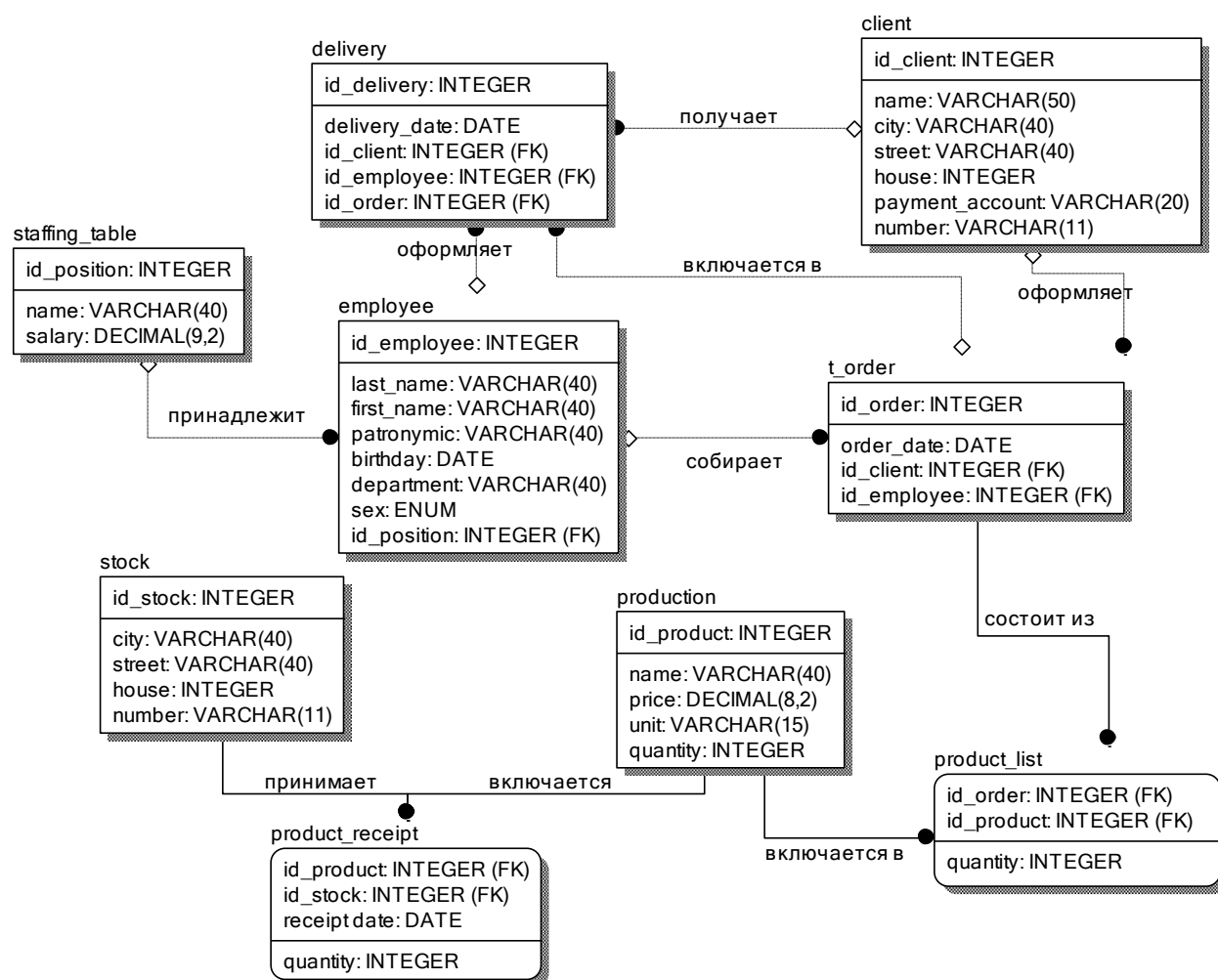


Рисунок 13 – СУБД-зависимая модель данных

Далее непосредственно был осуществлен переход к программной реализации базы данных. Программные код спроектированной базы данных представлен ниже.

```

CREATE DATABASE Confectionary_shop;
USE Confectionary_shop;

```

```

CREATE TABLE IF NOT EXISTS client (
    id_client INT NOT NULL,

```

```
name VARCHAR(50) NULL,  
city VARCHAR(40) NOT NULL,  
street VARCHAR(40) NOT NULL,  
house INT NOT NULL,  
payment_account VARCHAR(20) NULL,  
number VARCHAR(11) NULL,  
PRIMARY KEY (id_client)  
);
```

```
CREATE TABLE IF NOT EXISTS staffing_table (  
id_position INT NOT NULL,  
name VARCHAR(40) NULL,  
salary DECIMAL(9,2) NULL,  
PRIMARY KEY (id_position));
```

```
CREATE TABLE IF NOT EXISTS employee (  
id_employee INT NOT NULL,  
last_name VARCHAR(40) NOT NULL,  
first_name VARCHAR(40) NOT NULL,  
patronymic VARCHAR(40) NOT NULL,  
birthday DATE NULL,  
department VARCHAR(40) NULL,  
sex ENUM('м', 'ж') NULL,  
id_position INT NULL,  
PRIMARY KEY (id_employee),  
INDEX position_idx (id_position ASC) VISIBLE,  
CONSTRAINT position  
FOREIGN KEY (id_position)  
REFERENCES Staffing_table (id_position)  
ON DELETE CASCADE  
ON UPDATE CASCADE);
```

```
CREATE TABLE IF NOT EXISTS t_order (  
id_order INT NOT NULL,  
order_date DATE NULL,  
id_client INT NULL,  
id_employee INT NULL,  
PRIMARY KEY (id_order),  
INDEX client_idx (id_client ASC) VISIBLE,  
INDEX employee_idx (id_employee ASC) VISIBLE,  
CONSTRAINT client  
FOREIGN KEY (id_client)  
REFERENCES Client(id_client)  
ON DELETE CASCADE  
ON UPDATE CASCADE,  
CONSTRAINT employee  
FOREIGN KEY (id_employee)  
REFERENCES Employee(id_employee)  
ON DELETE CASCADE  
ON UPDATE CASCADE);
```

```
CREATE TABLE IF NOT EXISTS production (  
  id_product INT NOT NULL,  
  name VARCHAR(40) NULL,  
  price DECIMAL(8,2) NULL,  
  unit VARCHAR(15),  
  quantity INT,  
  PRIMARY KEY (id_product)  
);
```

```
CREATE TABLE IF NOT EXISTS product_list (  
  id_product INT NOT NULL,  
  id_order INT NOT NULL,  
  quantity INT NULL,  
  INDEX product_idx (id_product ASC) VISIBLE,  
  INDEX order_idx (id_order ASC) VISIBLE,  
  PRIMARY KEY (id_product, id_order),  
  CONSTRAINT product  
    FOREIGN KEY (id_product)  
    REFERENCES Production (id_product)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
  CONSTRAINT order  
    FOREIGN KEY (id_order)  
    REFERENCES Order (id_order)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE);
```

```
CREATE TABLE IF NOT EXISTS stock (  
  id_stock INT NOT NULL,  
  city VARCHAR(40) NOT NULL,  
  street VARCHAR(40) NOT NULL,  
  house INT NOT NULL,  
  number VARCHAR(11) NULL,  
  PRIMARY KEY (id_stock)  
);
```

```
CREATE TABLE IF NOT EXISTS product_receipt (  
  id_product INT NOT NULL,  
  id_stock INT NOT NULL,  
  quantity INT NULL,  
  receiptdate DATE NULL,  
  PRIMARY KEY (id_product, id_stock, receiptdate),  
  INDEX stock_idx (id_stock ASC) VISIBLE,  
  CONSTRAINT product_stock  
    FOREIGN KEY (id_product)  
    REFERENCES Production (id_product)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
  CONSTRAINT stock  
    FOREIGN KEY (id_stock)  
    REFERENCES Stock (id_stock)  
    ON DELETE CASCADE
```

ON UPDATE CASCADE);

```
CREATE TABLE delivery (  
  id_delivery INT NOT NULL,  
  delivery_date DATE NOT NULL,  
  id_client INT NULL,  
  id_employee INT NULL,  
  id_order INT NULL,  
  PRIMARY KEY (id_delivery),  
  INDEX client_delivery_idx (id_client ASC) VISIBLE,  
  INDEX employee_delivery_idx (id_employee ASC) VISIBLE,  
  INDEX order_delivery_idx (id_order ASC) VISIBLE,  
  CONSTRAINT client_delivery  
    FOREIGN KEY (id_client)  
    REFERENCES client (id_client)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
  CONSTRAINT employee_delivery  
    FOREIGN KEY (id_employee)  
    REFERENCES employee (id_employee)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
  CONSTRAINT order_delivery  
    FOREIGN KEY (id_order)  
    REFERENCES order (id_order)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE);
```

После создания структуры базы данных, необходимо заполнить базу набором данных. Ниже приведен программный код добавления записей в таблицы, а также результат выполнения данного скрипта в консоли.

```
INSERT INTO Client (id_client,name,city,street,house,payment_account, number)  
VALUES (1, 'Радуга', 'Курск', 'Ленина', 36, '23657123581475625789', '89510867535' ),  
(2, 'Пряничный домик','Белгород', 'Ленина', 36, '23657123581475625123',  
'89200837454' ),  
(3, 'Шоколадные реки', 'Курск', 'Радищева', 58, '23657123581475628964',  
'89081208456' );
```

```
mysql> select * from Client;  
+-----+-----+-----+-----+-----+-----+-----+  
| id_client | name          | city   | street | house | payment_account | number |  
+-----+-----+-----+-----+-----+-----+-----+  
| 1         | Радуга        | Курск  | Ленина | 36     | 23657123581475625789 | 89510867535 |  
| 2         | Пряничный домик | Белгород | Ленина | 36     | 23657123581475625123 | 89200837454 |  
| 3         | Шоколадные реки | Курск   | Радищева | 58     | 23657123581475628964 | 89081208456 |  
+-----+-----+-----+-----+-----+-----+-----+  
3 rows in set (0.00 sec)
```

Рисунок 14 – Таблица «client» с заполненными данными


```
INSERT INTO Staffing_table (id_position,name,salary)
VALUES (1, 'Менеджер отдела продаж', 21500.00),
(2, 'Комплектовщик склада', 18700.00);
```

id_position	name	salary
1	Менеджер отдела продаж	21500.00
2	Комплектовщик склада	18700.00

2 rows in set (0.00 sec)

Рисунок 15 – Таблица «staffing_table» с заполненными данными

```
INSERT INTO Employee
(id_employee,last_name,first_name,patronymic,birthday,department,sex,id_position)
VALUES (1, 'Авдеенко', 'Михаил', 'Иванович', '1986-05-23', 'отдел продаж','м',1),
(2, 'Петров', 'Иван', 'Александрович', '1994-06-02', 'отдел продаж','м',1),
(3, 'Симонов', 'Григорий', 'Михайлович', '1979-07-26', 'отдел складкой
логистики','м',2),
(4, 'Матвиенко', 'Николай', 'Иванович', '1993-08-15', 'отдел складкой
логистики','м',2);
```

```
mysql> select * from Employee;
```

id_employee	last_name	first_name	patronymic	birthday	department	sex	id_position
1	Авдеенко	Михаил	Иванович	1986-05-23	отдел продаж	м	1
2	Петров	Иван	Александрович	1994-06-02	отдел продаж	м	1
3	Симонов	Григорий	Михайлович	1979-07-26	отдел складкой логистики	м	2
4	Матвиенко	Николай	Иванович	1993-08-15	отдел складкой логистики	м	2

4 rows in set (0.00 sec)

Рисунок 16 – Таблица «employee» с заполненными данными

```
INSERT INTO t_order (id_order,order_date, id_client, id_employee)
VALUES (1, '2020-07-10', 1,3),
(2, '2020-07-05', 2,3),
(3, '2020-07-07', 3,4),
(4, '2020-07-09', 1,4);
```

id_order	order_date	id_client	id_employee
1	2020-07-10	1	3
2	2020-07-05	2	3
3	2020-07-07	3	4
4	2020-07-09	1	4

Рисунок 17 – Таблица «t_order» с заполненными данными

```
INSERT INTO production (id_product,name,price,unit)
VALUES (1, 'Трубочка слоеная с молочной начинкой', 432.50, 'ящ.', 30),
(2, 'Медовик', 163.00, 'кг.', 30),
(3, 'Рулет Фруктовый ЛИМОННЫЙ', 721.50, 'ящ.', 30),
(4, 'Рулет Фруктовый АБРИКОСОВЫЙ', 721.50, 'ящ.', 40),
(5, 'Печенье Вишня с шоколадом', 383.00, 'ящ.', 40),
(6, 'Печенье Лунное с миндалем', 244.00, 'ящ.', 40);
```

```
mysql> select * from production;
```

id_product	name	price	unit	quantity
1	Трубочка слоеная с молочной начинкой	432.50	ящ.	30
2	Медовик	163.00	кг.	30
3	Рулет Фруктовый ЛИМОННЫЙ	721.50	ящ.	30
4	Рулет Фруктовый АБРИКОСОВЫЙ	721.50	ящ.	40
5	Печенье Вишня с шоколадом	383.00	ящ.	40
6	Печенье Лунное с миндалем	244.00	ящ.	40

6 rows in set (0.00 sec)

Рисунок 18 – Таблица «production» с заполненными данными

```
INSERT INTO product_list (id_order,id_product,quantity)
VALUES (1, 1, 2), (1, 2, 3), (2, 3, 2),
(3, 4, 5), (4, 5, 4), (4, 6, 2);
```

```
mysql> select * from product_list;
```

id_product	id_order	quantity
1	1	2
2	1	3
3	2	2
4	3	5
5	4	4
6	4	2

Рисунок 19 – Таблица «product_list» с заполненными данными

```
INSERT INTO stock (id_stock,city,street,house,number)
VALUES (1, 'Курск', 'Литовская', 42, '84712124578'),
(2, 'Курск', 'Овечкина', 24, '84712124556'),
(3, 'Курск', 'Союзная', 58, '84712128932');
```

```
mysql> select * from stock;
```

id_stock	city	street	house	number
1	Курск	Литовская	42	84712124578
2	Курск	Овечкина	24	84712124556
3	Курск	Союзная	58	84712128932

Рисунок 20 – Таблица «stock» с заполненными данными

```
INSERT INTO product_receipt (id_product,id_stock,quantity,receiptdate)
VALUES (1, 1, 10, '2020-07-01'),
(2, 1, 10, '2020-07-02'),
(3, 2, 10, '2020-06-29'),
(4, 2, 10, '2020-07-07'),
(5, 3, 10, '2020-07-03'),
(6, 3, 10, '2020-07-03');
```

```
mysql> select * from product_receipt;
```

id_product	id_stock	quantity	receiptdate
1	1	10	2020-07-01
2	1	10	2020-07-02
3	2	10	2020-06-29
4	2	10	2020-07-07
5	3	10	2020-07-03
6	3	10	2020-07-03

6 rows in set (0.00 sec)

Рисунок 21 – Таблица «product_receipt» с заполненными данными

```
INSERT INTO delivery (id_delivery,delivery_date,id_client,id_employee,id_order)
VALUES (1,'2020-07-14', 1, 1, 1),
(2,'2020-07-10', 2, 1, 2),
(3,'2020-07-11', 3, 2, 3),
(4,'2020-07-12', 1, 2, 4);
```

```
mysql> select * from delivery;
```

id_delivery	delivery_date	id_client	id_employee	id_order
1	2020-07-14	1	1	1
2	2020-07-10	2	1	2
3	2020-07-11	3	2	3
4	2020-07-12	1	2	4

4 rows in set (0.00 sec)

Рисунок 22 – Таблица «delivery» с заполненными данными

Создадим несколько представлений для базы данных.

Первое представление выбирает наименование продукции с ценой выше среднего.

```
CREATE VIEW Products_Above_Average_Price AS
SELECT name, price, unit
FROM production
WHERE price > (SELECT AVG(price) FROM production)
SELECT * FROM Products_Above_Average_Price;
```

```
mysql> SELECT * FROM Products_Above_Average_Price;
+-----+-----+-----+
| name                | price | unit |
+-----+-----+-----+
| Рулет Фруктовый ЛИМОННЫЙ | 721.50 | ящ. |
| Рулет Фруктовый АБРИКОСОВЫЙ | 721.50 | ящ. |
+-----+-----+-----+
2 rows in set (0.06 sec)
```

Рисунок 23 – Результат выполнения представления Products_Above_Average_Price

Второе представление выбирает информацию о заказе и клиенте, который совершил данный заказ, с сортировкой по дате.

```
CREATE VIEW orders_with_customer AS
SELECT o.order_date, o.id_order, c.name, c.number
FROM t_order o inner join client c
ON o.id_client=c.id_client
ORDER BY o.order_date;
```

```
mysql> select * from orders_with_customer;
+-----+-----+-----+-----+
| order_date | id_order | name                | number |
+-----+-----+-----+-----+
| 2020-07-05 | 2        | Пряничный домик    | 89200837454 |
| 2020-07-07 | 3        | Шоколадные реки    | 89081208456 |
| 2020-07-09 | 4        | Радуга              | 89510867535 |
| 2020-07-10 | 1        | Радуга              | 89510867535 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Рисунок 24 – Результат выполнения представления orders_with_customer

Теперь создадим несколько процедур для базы данных.

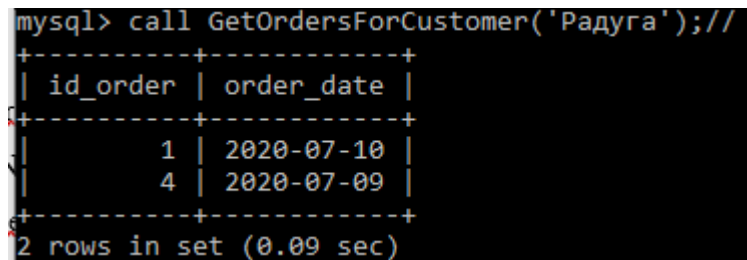
Следующая процедура выводит список заказов по наименованию организации клиента.

```
DELIMITER //
CREATE PROCEDURE GetOrdersForCustomer (IN clientName varchar(50))
BEGIN
```

```

SELECT id_order, order_date
FROM t_order INNER JOIN client
ON t_order.id_client=client.id_client AND client.name=clientName;
END//

```



```

mysql> call GetOrdersForCustomer('Радуга');//
+-----+-----+
| id_order | order_date |
+-----+-----+
| 1 | 2020-07-10 |
| 4 | 2020-07-09 |
+-----+-----+
2 rows in set (0.09 sec)

```

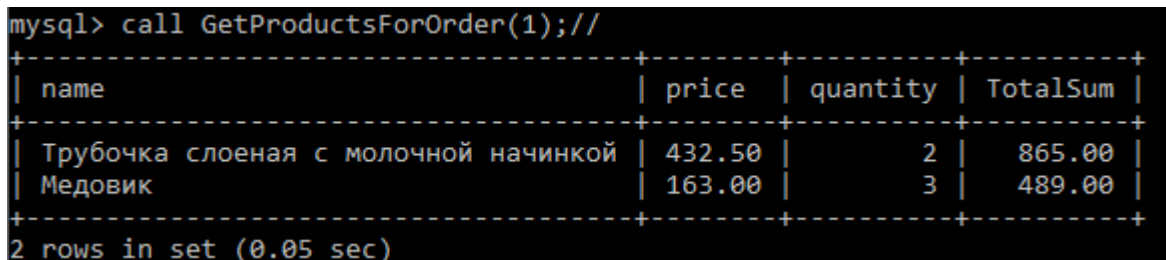
Рисунок 25 – Результат вызова процедуры GetOrdersForCustomer

Также была реализована процедура для просмотра списка товаров по каждому заказу с выводом общей суммы для каждого отдельного продукта.

```

CREATE PROCEDURE GetProductsForOrder (IN OrderID INT)
BEGIN
    SELECT p.name, p.price, l.quantity, p.price*l.quantity AS TotalSum
    FROM production p INNER JOIN product_list l
    ON p.id_product=l.id_product AND l.id_order=OrderID;
END//

```



```

mysql> call GetProductsForOrder(1);//
+-----+-----+-----+-----+
| name | price | quantity | TotalSum |
+-----+-----+-----+-----+
| Трубочка слоеная с молочной начинкой | 432.50 | 2 | 865.00 |
| Медовик | 163.00 | 3 | 489.00 |
+-----+-----+-----+-----+
2 rows in set (0.05 sec)

```

Рисунок 26 – Результат вызова процедуры GetProductsForOrder

Для базы данных была создана функция, которая возвращает итоговую сумму заказа.

```

CREATE FUNCTION TotalSumOrder (OrderID INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE TotalSum DECIMAL(10,2);
    SELECT SUM(p.price*l.quantity) INTO TotalSum
    FROM product_list l
    INNER JOIN production p

```

```

ON l.id_product=p.id_product
AND l.id_order=1
GROUP BY l.id_order;
RETURN TotalSum;
END//

```

```

mysql> select TotalSumOrder(1);
-> //
+-----+
| TotalSumOrder(1) |
+-----+
|          1354.00 |
+-----+
1 row in set (0.08 sec)

```

Рисунок 27 – Результат выполнения функции TotalSumOrder

Также был реализован триггер, который при поступлении новой продукции на склад, обновляет количество имеющейся готовой продукции.

```

CREATE TRIGGER upProductCount
AFTER INSERT ON product_receipt FOR EACH ROW
BEGIN
UPDATE production SET quantity = quantity + NEW.quantity
WHERE NEW.id_product = production.id_product;
END //
INSERT INTO product_receipt (id_product,id_stock,quantity,receiptdate)
VALUES (1, 1, 15, '2020-07-11');

```

```

mysql> select * from production;
+-----+-----+-----+-----+-----+
| id_product | name                                     | price | unit | quantity |
+-----+-----+-----+-----+-----+
| 1          | Трубочка слоеная с молочной начинкой | 432.50 | ящ.  | 45        |
| 2          | Медовик                                | 163.00 | кг.   | 30        |
| 3          | Рулет Фруктовый ЛИМОННЫЙ              | 721.50 | ящ.   | 30        |
| 4          | Рулет Фруктовый АБРИКОСОВЫЙ          | 721.50 | ящ.   | 40        |
| 5          | Печенье Вишня с шоколадом             | 383.00 | ящ.   | 40        |
| 6          | Печенье Лунное с миндалем             | 244.00 | ящ.   | 40        |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

```

Рисунок 28 – Результат выполнения триггера upProductCount

В ходе выполнения данной работы были пройдены все этапы проектирования базы данных с ее последующей программной реализацией в реляционной СУБД MySQL и заполнением тестовыми данными. Были реализованы представления, процедуры, функции и триггеры, которые существенно помогают облегчить работу пользователя.

Также в процессе проектирования базы данных были рассмотрены три основных подхода к построению модели данных и приведен сравнительный анализ данных подходов. В результате сравнения для дальнейшего проектирования была выбрана реляционная модель данных. Данная модель считается наиболее гибкой и удобной в использовании. Также приведено описание реляционной модели данных в математических терминах.

Результатом данной работы выступает база данных сбыта кондитерского цеха. Она удовлетворяет всем требованиям и нормам исследуемой предметной области. При внедрении на данный кондитерских цех упростится заказ продукции, работа менеджеров и сотрудников склада. Также в дальнейшем разработанную базу данных можно развить до уровня хранения данных, необходимых для обеспечения всех основных бизнес-процессов предприятия. Разработанную базу данных можно с легкостью внедрить на другое аналогичное предприятие.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гордеев С. И., Волошина В.Н. Организация баз данных в 2 ч. Часть 1: учебник для вузов – 2-е изд., испр. и доп. – Москва : Издательство Юрайт, 2020. — 310 с.
2. Дьяков И.А. Базы данных. Язык SQL: Учеб. Пособие – Тамбов: Изд-во Тамб. Гос. Техн. Ун-та, 2014 – 80 с.
3. Кара-Ушанов В. Ю. Разработка баз данных в CASE-среде Erwin: Учебное пособие – Екатеринбург: Екатеринбургская академия современного искусства, 2015. – 134 с.
4. Кумскова И.А. Базы данных: учебник – 2-е изд., стер. – М.: КНОРУС, 2016. – 488 с.
5. Советов Б. Я., Цехановский В.В., Чертовской В. Д Базы данных : учебник для вузов – 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2020. — 420 с.
6. Стружкин Н. П., Годин В.В. Базы данных: проектирование: учебник для – Москва: Издательство Юрайт, 2020. – 477 с.
7. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений / под ред. Проф. А.Д. Хомоненко. – СПб.: КОРОНА принт, 2016. – 672с.