Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Высшая школа бизнеса

КОНТРОЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

«Проектировка архитектуры и разработка функций для супермакрета»

по направлению подготовки 38.03.05 «Бизнес-информатика» образовательная программа «Бизнес-информатика»

Проект выполнил:

Чубукин Кирилл, 222

Москва

**Оглавление**

[Анализ предметной области 2](#_Toc138193481)

[Разработка моделей 7](#_Toc138193482)

[Размещение базы данных в СУБД 12](#_Toc138193483)

[Генерация данных 17](#_Toc138193484)

[Полезные запросы 17](#_Toc138193485)

[Хранимые процедуры 31](#_Toc138193486)

[Триггеры 38](#_Toc138193487)

[Функции 39](#_Toc138193488)

[Представления 42](#_Toc138193489)

[Индексы 43](#_Toc138193490)

[Отчеты 43](#_Toc138193491)

[Вывод 47](#_Toc138193492)

#### **Общие сведения о предметной области**

Предметной областью является управление данными супермаркета. Система включает информацию о клиентах, заказах, продуктах и поставщиках. Эта система предназначена для автоматизации процессов обработки заказов, управления запасами и взаимодействия с клиентами.

#### Цель создания базы данных

Цель создания базы данных - обеспечить эффективное управление информацией супермаркета, автоматизировать обработку заказов, отслеживание запасов и улучшить взаимодействие с клиентами. Это позволит оптимизировать процессы, такие как сортировка заказов, управление клиентской базой и анализ продаж.

#### Возможные пользователи базы данных

Потенциальными пользователями базы данных являются:

* **Менеджеры супермаркета:** управляют клиентской базой, контролируют заказы и запасы.
* **Сотрудники отдела продаж:** обрабатывают заказы, взаимодействуют с клиентами.
* **Аналитики:** анализируют данные о продажах и запасах для принятия управленческих решений.

#### Возможные сценарии взаимодействия пользователей и БД

* **Менеджеры супермаркета** могут:
  + Создавать и редактировать записи клиентов.
  + Управлять информацией о продуктах и поставщиках.
  + Генерировать отчеты о запасах и продажах.
  + Настраивать индексы и представления для оптимизации запросов.
* **Сотрудники отдела продаж** могут:
  + Обрабатывать заказы клиентов.
  + Просматривать информацию о заказах и клиентах.
  + Обновлять статусы заказов и следить за их выполнением.
* **Аналитики** могут:
  + Выполнять запросы для анализа данных о продажах и запасах.
  + Создавать сложные отчеты для руководства.
  + Использовать представления и функции для получения агрегированных данных.

#### Основные сущности и их описание

#### 1. **Customer (Клиенты)**

* Содержит информацию о клиентах супермаркета (ID клиента, фамилия, имя, город клиента).
* Используется для хранения данных о клиентах, которые делают заказы.

#### 2. **Orders (Заказы)**

* Содержит информацию о заказах, сделанных клиентами (ID заказа, дата заказа, ID клиента, общая сумма заказа).
* Используется для управления и отслеживания заказов клиентов.

#### 3. **OrderItem (Позиции заказа)**

* Содержит информацию о позициях в заказах (ID позиции, ID заказа, ID продукта, количество, цена за единицу).
* Используется для детализации заказов и управления товарами в заказах.

#### 4. **Product (Продукты)**

* Содержит информацию о продуктах, доступных в супермаркете (ID продукта, название продукта, ID поставщика, цена за единицу).
* Используется для хранения данных о товарах, которые предлагаются для продажи.

#### 5. **Supplier (Поставщики)**

* Содержит информацию о поставщиках продуктов (ID поставщика, название компании, ID города).
* Используется для управления данными о компаниях, которые поставляют товары супермаркету.

#### 6. **Country (Страны) и City (Города)**

* Содержат информацию о географическом расположении (ID страны, название страны, ID города, название города).
* Используются для структурирования данных о местоположении клиентов и поставщиков.

#### 7. **Package (Упаковки)**

* Содержит информацию об упаковках продуктов (ID упаковки, название упаковки).
* Используется для хранения данных о различных типах упаковок, которые могут использоваться для продуктов.

Список основных сущностей предметной области и их атрибуты, описание связей между сущностями

1. **Customer (Клиенты)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + LastName (nvarchar(100), NOT NULL)
     + FirstName (nvarchar(100), NOT NULL)
     + CityId (int, NOT NULL, FK)
2. **Orders (Заказы)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + OrderDate (datetime, NOT NULL)
     + OrderNumber (nvarchar(100), NOT NULL)
     + TotalAmount (decimal, NOT NULL)
     + CustomerId (int, NOT NULL, FK)
3. **OrderItem (Позиции заказа)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + OrderId (int, NOT NULL, FK)
     + ProductId (int, NOT NULL, FK)
     + Quantity (int, NOT NULL)
     + UnitPrice (decimal, NOT NULL)
4. **Product (Продукты)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + ProductName (nvarchar(100), NOT NULL)
     + SupplierID (int, NOT NULL, FK)
     + UnitPrice (decimal, NOT NULL)
5. **Supplier (Поставщики)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + CompanyName (nvarchar(100), NOT NULL)
     + CityId (int, NOT NULL, FK)
6. **Country (Страны)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + Name (nvarchar(100), NOT NULL)
7. **City (Города)**
   * Атрибуты:
     + id (int, NOT NULL, PK)
     + Name (nvarchar(100), NOT NULL)
     + CountryId (int, NOT NULL, FK)

**8. Package (Упаковки)**

* **Атрибуты:**
  + Id (int, NOT NULL, PK) - Уникальный идентификатор упаковки.
  + Name (nvarchar(50), NOT NULL) - Название упаковки.

#### Описание связей между сущностями

1. **Customer - City**
   * Связь "многие к одному" (many-to-one) между таблицами Customer и City. Один город может содержать несколько клиентов, но каждый клиент относится только к одному городу.
2. **Orders - Customer**
   * Связь "многие к одному" (many-to-one) между таблицами Orders и Customer. Один клиент может иметь несколько заказов, но каждый заказ относится только к одному клиенту.
3. **OrderItem - Orders**
   * Связь "многие к одному" (many-to-one) между таблицами OrderItem и Orders. Один заказ может содержать несколько позиций, но каждая позиция относится только к одному заказу.
4. **OrderItem - Product**
   * Связь "многие к одному" (many-to-one) между таблицами OrderItem и Product. Один продукт может быть включен в несколько позиций заказов, но каждая позиция заказа относится только к одному продукту.
5. **Product - Supplier**
   * Связь "многие к одному" (many-to-one) между таблицами Product и Supplier. Один поставщик может поставлять несколько продуктов, но каждый продукт относится только к одному поставщику.
6. **Supplier - City**
   * Связь "многие к одному" (many-to-one) между таблицами Supplier и City. Один город может содержать несколько поставщиков, но каждый поставщик относится только к одному городу.
7. **City - Country**
   * Связь "многие к одному" (many-to-one) между таблицами City и Country. Одна страна может содержать несколько городов, но каждый город относится только к одной стране.

#### **Product - Package**

* **Связь:** "многие к одному" (many-to-one) между таблицами Product и Package.

Алгоритмы обработки данных, используемые в сценариях

1. **Подсчёт выручки с каждого клиента на платформе супермаркета**
   * Подсчёт выручки с каждого клиента производится с помощью представления VIEW customer\_revenue, которое суммирует общую сумму заказов для каждого клиента
2. **Получение списка продуктов, заказанных каждым клиентом через запрос**
   * Получение списка продуктов, заказанных каждым клиентом, производится через запрос, который объединяет таблицы
3. **Фильтр для поиска заказов на определенные продукты**
   * Для поиска заказов, включающих определенные продукты, используется запрос с использованием ключевого слова FROM:
4. **Сравнение количества заказов в разные месяцы для определения трендов**
   * Сравнение количества заказов в разные месяцы для определения трендов выполняется с помощью функций LAG и и LEAD
5. **Вывод таблицы с количеством заказов для каждого продукта**
   * Вывод таблицы с количеством заказов для каждого продукта осуществляется через объединение таблиц Product и OrderItem с помощью функции JOIN:
6. **Фильтр для поиска клиентов, совершивших несколько заказов**
   * Для поиска клиентов, совершивших несколько заказов, используется запрос с функцией

**Планируемый перечень отчётных форм**

1. **Отчёт по выручке с каждого клиента**
   * Отчёт с использованием функции drill down показывает общую выручку по каждому клиенту. При клике на конкретного клиента, можно увидеть отчёт по каждому заказу этого клиента. После этого можно просмотреть более подробный отчёт, в котором указаны продукты, включенные в каждый заказ.
2. **Отчёт по количеству заказов у каждого клиента**
   * Отчёт демонстрирует количество заказов, сделанных каждым клиентом. В отчёте указываются все заказы, номер заказа, дата заказа и общая стоимость каждого заказа.

Возможная архитектура программного продукта на основе базы данных

Представленная разработка предполагает использование веб-сайта (приложения) для взаимодействия с базой данных супермаркета. Взаимодействие осуществляется как клиентами и сотрудниками отдела продаж, так и менеджерами для обработки запросов к базе данных. С помощью запросов, поступающих на сервер программного продукта, пользователи смогут находить необходимую информацию и выполнять различные операции. Например, менеджеры смогут запрашивать отчеты о продажах, а клиенты смогут находить информацию о своих заказах.

#### Взаимодействие пользователей с системой

1. **Клиентская сторона сайта/приложения**
   * **Запросы:**
     + Просмотр списка доступных продуктов.
     + Просмотр деталей заказов.
     + Оформление новых заказов.
   * **Ответы:**
     + Информация о продуктах (название, цена, доступность).
     + Подтверждение оформления заказа.
     + Статус выполнения заказа.
2. **Сторона сотрудников отдела продаж сайта/приложения**
   * **Запросы:**
     + Обработка новых заказов.
     + Обновление информации о заказах.
     + Просмотр деталей клиентов.
   * **Ответы:**
     + Подтверждение обработки заказа.
     + Информация о статусе заказов.
     + Данные о клиентах (контактная информация, история заказов).
3. **Администраторская сторона сайта/приложения**
   * **Запросы:**
     + Управление информацией о продуктах.
     + Управление списком клиентов.
     + Генерация отчетов о продажах и запасах.
   * **Ответы:**
     + Подтверждение выполнения административных операций.
     + Отчеты о продажах и запасах.
     + Список клиентов и их заказов.

#### **Схема взаимодействия**

1. **Клиентская сторона сайта/приложения**
   * Запросы от клиентов поступают на сервер приложения, где обрабатываются и перенаправляются к серверу базы данных для получения необходимой информации. Ответы возвращаются на клиентскую сторону через сервер приложения.
2. **Сторона сотрудников отдела продаж сайта/приложения**
   * Запросы от сотрудников отдела продаж поступают на сервер приложения, который взаимодействует с сервером базы данных для выполнения операций, таких как обработка заказов и обновление информации о заказах. Ответы возвращаются через сервер приложения.
3. **Администраторская сторона сайта/приложения**
   * Администраторы отправляют запросы на сервер приложения для выполнения административных задач, таких как управление продуктами и генерация отчетов. Сервер приложения взаимодействует с сервером базы данных для выполнения этих операций и возвращает ответы администраторам.

**Разработка моделей**

#### **Основные понятия**

* **PK (Primary Key)**: Первичный ключ. Атрибут, принимающий значения, по которым можно найти необходимую сущность, без опасения неоднозначности.
* **FK (Foreign Key)**: Внешний ключ. Атрибут, опираясь на который, данные получают свойства различия, что не позволяет повторяющимся объектам, которые различны по своей природе, быть неправильно сортированы.

В модели данных "супермаркет" будет 7 сущностей: "клиенты", "заказы", "позиции заказа", "продукты", "поставщики", "страны", "города". Далее более подробно будут описаны связи и атрибуты данных сущностей.

#### **Сущности и их атрибуты**

1. **Клиенты (Customer)**
   * Связь "один ко многим" (one-to-many) между таблицами "страны" и "клиенты", так как одна страна может содержать несколько клиентов, но каждый клиент относится только к одной стране.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + LastName (nvarchar(100), NOT NULL)
     + FirstName (nvarchar(100), NOT NULL)
     + CityId (int, NOT NULL, FK)
2. **Заказы (Orders)**
   * Связь "один ко многим" (one-to-many) между таблицами "клиенты" и "заказы", так как один клиент может иметь несколько заказов, но каждый заказ относится только к одному клиенту.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + OrderDate (datetime, NOT NULL)
     + OrderNumber (nvarchar(100), NOT NULL)
     + TotalAmount (decimal, NOT NULL)
     + CustomerId (int, NOT NULL, FK)
3. **Позиции заказа (OrderItem)**
   * Связь "один ко многим" (one-to-many) между таблицами "заказы" и "позиции заказа", так как один заказ может содержать несколько позиций, но каждая позиция относится только к одному заказу.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + OrderId (int, NOT NULL, FK)
     + ProductId (int, NOT NULL, FK)
     + Quantity (int, NOT NULL)
     + UnitPrice (decimal, NOT NULL)
4. **Продукты (Product)**
   * Связь "один ко многим" (one-to-many) между таблицами "поставщики" и "продукты", так как один поставщик может поставлять несколько продуктов, но каждый продукт относится только к одному поставщику.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + ProductName (nvarchar(100), NOT NULL)
     + SupplierID (int, NOT NULL, FK)
     + UnitPrice (decimal, NOT NULL)
5. **Поставщики (Supplier)**
   * Связь "один ко многим" (one-to-many) между таблицами "города" и "поставщики", так как один город может содержать несколько поставщиков, но каждый поставщик относится только к одному городу.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + CompanyName (nvarchar(100), NOT NULL)
     + CityId (int, NOT NULL, FK)
6. **Страны (Country)**
   * Связь "один ко многим" (one-to-many) между таблицами "страны" и "города", так как одна страна может содержать несколько городов, но каждый город относится только к одной стране.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + Name (nvarchar(100), NOT NULL)
7. **Города (City)**
   * Связь "один ко многим" (one-to-many) между таблицами "страны" и "города", так как одна страна может содержать несколько городов, но каждый город относится только к одной стране.
   * **Основные поля:**
     + id (int, NOT NULL, PK)
     + Name (nvarchar(100), NOT NULL)
     + CountryId (int, NOT NULL, FK)

#### **Связи между сущностями**

1. **Customer - City**
   * Связь "многие к одному" (many-to-one) между таблицами Customer и City. Один город может содержать несколько клиентов, но каждый клиент относится только к одному городу.
2. **Orders - Customer**
   * Связь "многие к одному" (many-to-one) между таблицами Orders и Customer. Один клиент может иметь несколько заказов, но каждый заказ относится только к одному клиенту.
3. **OrderItem - Orders**
   * Связь "многие к одному" (many-to-one) между таблицами OrderItem и Orders. Один заказ может содержать несколько позиций, но каждая позиция относится только к одному заказу.
4. **OrderItem - Product**
   * Связь "многие к одному" (many-to-one) между таблицами OrderItem и Product. Один продукт может быть включен в несколько позиций заказов, но каждая позиция заказа относится только к одному продукту.
5. **Product - Supplier**
   * Связь "многие к одному" (many-to-one) между таблицами Product и Supplier. Один поставщик может поставлять несколько продуктов, но каждый продукт относится только к одному поставщику.
6. **Supplier - City**
   * Связь "многие к одному" (many-to-one) между таблицами Supplier и City. Один город может содержать несколько поставщиков, но каждый поставщик относится только к одному городу.
7. **City - Country**
   * Связь "многие к одному" (many-to-one) между таблицами City и Country. Одна страна может содержать несколько городов, но каждый город относится только к одной стране.

*Даталогическая модель данных*

*Изображение выглядит как текст, диаграмма, План, Параллельный

Автоматически созданное описание*

Изображение выглядит как текст, снимок экрана, чек, дизайн

Автоматически созданное описание

*Изображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описание*

*Получившаяся модель в СУБД*

*Изображение выглядит как диаграмма, План, Технический чертеж, зарисовка

Автоматически созданное описаниеСхема модели в СУБД*

База данных супермаркета имеет 8 таблиц. Несмотря на то, что названия таблиц и атрибутов прописаны на английском языке, кортежи (записи) могут быть заполнены на любом языке. Это было сделано для удобства и совместимости с различными системами.

#### Нормализация базы данных

База данных находится в третьей нормальной форме (3НФ). Это достигается следующим образом:

1. **Первая нормальная форма (1НФ):**
   * Отсутствуют повторяющиеся атрибуты.
   * Атрибуты являются атомарными.
   * Ни один атрибут не является составным.
2. **Вторая нормальная форма (2НФ):**
   * База данных находится в 1НФ.
   * Первичный ключ однозначно определяет запись.
   * Все поля таблицы полностью зависят от первичного ключа.
3. **Третья нормальная форма (3НФ):**
   * База данных находится в 2НФ.
   * Значения всех полей, не входящих в первичный ключ, не зависят от значений других полей.
   * Неключевые атрибуты находятся в функциональной зависимости от одного или нескольких неключевых атрибутов.

Эти шаги нормализации обеспечивают устранение избыточности данных и предотвращают аномалии при обновлении, удалении и вставке данных.

**Размещение базы данных в СУБД**

-- DROP SCHEMA dbo;

CREATE SCHEMA dbo;

-- Supermarket.dbo.Country определение

-- Drop table

-- DROP TABLE Supermarket.dbo.Country;

CREATE TABLE Supermarket.dbo.Country (

ID int NOT NULL,

Name nchar(50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

CONSTRAINT PK\_Country PRIMARY KEY (ID)

);

-- Supermarket.dbo.Package определение

-- Drop table

-- DROP TABLE Supermarket.dbo.Package;

CREATE TABLE Supermarket.dbo.Package (

Id int NOT NULL,

Name nvarchar(50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

CONSTRAINT PK\_Package PRIMARY KEY (Id)

);

-- Supermarket.dbo.sysdiagrams определение

-- Drop table

-- DROP TABLE Supermarket.dbo.sysdiagrams;

CREATE TABLE Supermarket.dbo.sysdiagrams (

name sysname COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

principal\_id int NOT NULL,

diagram\_id int IDENTITY(1,1) NOT NULL,

version int NULL,

definition varbinary(MAX) NULL,

CONSTRAINT PK\_\_sysdiagr\_\_C2B05B6179E0D624 PRIMARY KEY (diagram\_id),

CONSTRAINT UK\_principal\_name UNIQUE (principal\_id,name)

);

-- Supermarket.dbo.City определение

-- Drop table

-- DROP TABLE Supermarket.dbo.City;

CREATE TABLE Supermarket.dbo.City (

ID int NOT NULL,

Name nchar(50) COLLATE Cyrillic\_General\_CI\_AS NULL,

CountryId int NULL,

CONSTRAINT PK\_City PRIMARY KEY (ID),

CONSTRAINT FK\_City\_Country FOREIGN KEY (CountryId) REFERENCES Supermarket.dbo.Country(ID)

);

-- Supermarket.dbo.Customer определение

-- Drop table

-- DROP TABLE Supermarket.dbo.Customer;

CREATE TABLE Supermarket.dbo.Customer (

Id int NOT NULL,

FirstName nvarchar(50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

LastName nvarchar(50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

Secondname nvarchar(50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

Phone nvarchar(20) COLLATE Cyrillic\_General\_CI\_AS NULL,

CityId int NULL,

CONSTRAINT PK\_Customer PRIMARY KEY (Id),

CONSTRAINT FK\_Customer\_City FOREIGN KEY (CityId) REFERENCES Supermarket.dbo.City(ID)

);

CREATE NONCLUSTERED INDEX OrderCustomer ON dbo.Customer ( LastName ASC )

WITH ( PAD\_INDEX = OFF ,FILLFACTOR = 100 ,SORT\_IN\_TEMPDB = OFF , IGNORE\_DUP\_KEY = OFF , STATISTICS\_NORECOMPUTE = OFF , ONLINE = OFF , ALLOW\_ROW\_LOCKS = ON , ALLOW\_PAGE\_LOCKS = ON )

ON [PRIMARY ] ;

-- Supermarket.dbo.Orders определение

-- Drop table

-- DROP TABLE Supermarket.dbo.Orders;

CREATE TABLE Supermarket.dbo.Orders (

id int NOT NULL,

OrderDate datetime NOT NULL,

OrderNumber varchar(10) COLLATE Cyrillic\_General\_CI\_AS NULL,

CustomerID int NOT NULL,

TotalAmount numeric(12,2) NULL,

CONSTRAINT IX\_Orders UNIQUE (id),

CONSTRAINT PK\_\_Orders\_\_3213E83FF6F22108 PRIMARY KEY (id),

CONSTRAINT FK\_Orders\_Customer FOREIGN KEY (CustomerID) REFERENCES Supermarket.dbo.Customer(Id)

);

CREATE NONCLUSTERED INDEX DateOrder ON dbo.Orders ( id ASC )

WITH ( PAD\_INDEX = OFF ,FILLFACTOR = 100 ,SORT\_IN\_TEMPDB = OFF , IGNORE\_DUP\_KEY = OFF , STATISTICS\_NORECOMPUTE = OFF , ONLINE = OFF , ALLOW\_ROW\_LOCKS = ON , ALLOW\_PAGE\_LOCKS = ON )

ON [PRIMARY ] ;

-- Supermarket.dbo.Supplier определение

-- Drop table

-- DROP TABLE Supermarket.dbo.Supplier;

CREATE TABLE Supermarket.dbo.Supplier (

id int NOT NULL,

CompanyName varchar(40) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

ContactName varchar(50) COLLATE Cyrillic\_General\_CI\_AS NULL,

ContactTitle varchar(40) COLLATE Cyrillic\_General\_CI\_AS NULL,

Phone varchar(30) COLLATE Cyrillic\_General\_CI\_AS NULL,

Fax varchar(30) COLLATE Cyrillic\_General\_CI\_AS NULL,

CityId int NOT NULL,

CONSTRAINT PK\_\_Supplier\_\_3213E83F46A533E8 PRIMARY KEY (id),

CONSTRAINT FK\_Supplier\_City FOREIGN KEY (CityId) REFERENCES Supermarket.dbo.City(ID)

);

-- Supermarket.dbo.Product определение

-- Drop table

-- DROP TABLE Supermarket.dbo.Product;

CREATE TABLE Supermarket.dbo.Product (

id int NOT NULL,

ProductName varchar(50) COLLATE Cyrillic\_General\_CI\_AS NOT NULL,

SupplierID int NOT NULL,

UnitPrice numeric(12,2) NULL,

IsDiscontinued bit NOT NULL,

PackageId int NULL,

CONSTRAINT PK\_\_Product\_\_3213E83F32047986 PRIMARY KEY (id),

CONSTRAINT FK\_Product\_Supplier1 FOREIGN KEY (SupplierID) REFERENCES Supermarket.dbo.Supplier(id)

);

-- Supermarket.dbo.OrderItem определение

-- Drop table

-- DROP TABLE Supermarket.dbo.OrderItem;

CREATE TABLE Supermarket.dbo.OrderItem (

id int NOT NULL,

OrderID int NOT NULL,

ProductID int NOT NULL,

UnitPrice numeric(12,2) NOT NULL,

Quantity int NOT NULL,

CONSTRAINT IX\_OrderItem UNIQUE (id),

CONSTRAINT PK\_\_OrderIte\_\_3213E83FEB1B80E5 PRIMARY KEY (id),

CONSTRAINT FK\_OrderItem\_Orders FOREIGN KEY (OrderID) REFERENCES Supermarket.dbo.Orders(id),

CONSTRAINT FK\_OrderItem\_Product FOREIGN KEY (ProductID) REFERENCES Supermarket.dbo.Product(id)

);

Генерация данных

Для того, чтобы заполнить таблицу на 1.5 тысячи строк был использован Excel. Чтобы сформировать запрос на вставку были заполнены необходимые поля таблицы данными, после чего с помощью конкатенации и заранее прописанных данных был создан запрос, после копирования и вставки в SQL редактор всего столбца с запросами, был запущен сам редактор.

Полезные запросы

#### Простые запросы с условием и формулами в SELECT

1. **Запрос показывает общую сумму продаж по каждому продукту за неделю:**

SELECT SUM(TotalAmount) AS total\_sales, ProductId

FROM OrderItem

WHERE OrderDate BETWEEN '2022-09-02' AND '2022-09-09'

GROUP BY ProductId;

1. **Запрос считает количество заказов на каждый продукт:**

SELECT ProductId, COUNT(id) AS order\_count

FROM OrderItem

GROUP BY ProductId;

#### Запросы с коррелированным подзапросом в SELECT

1. **Запрос показывает отклонение цены продукта от среднего значения по всем имеющимся продуктам:**

SELECT ProductName, UnitPrice,

UnitPrice - (SELECT AVG(UnitPrice) FROM Product) AS price\_deviation

FROM Product;

1. **Запрос выводит название города и количество клиентов в каждом городе, а также отклонение от среднего количества клиентов по всем городам:**

SELECT City.Name, COUNT(Customer.id) AS customer\_count,

COUNT(Customer.id) - (SELECT AVG(customer\_count)

FROM (SELECT CityId, COUNT(id) AS customer\_count

FROM Customer

GROUP BY CityId) AS avg\_customers) AS deviation\_from\_avg

FROM City

JOIN Customer ON City.id = Customer.CityId

GROUP BY City.Name;

#### Запросы с подзапросом в FROM

1. **Счёт стоимости каждого заказа. Запрос проходит по всем заказам и вычисляет общую сумму для каждого заказа:**

SELECT \*

FROM (SELECT OrderId, SUM(UnitPrice \* Quantity) AS total\_order\_cost

FROM OrderItem

GROUP BY OrderId) AS order\_costs;

1. **Запрос выбирает все продукты, которые относятся к конкретному поставщику:**

SELECT \*

FROM Product

WHERE SupplierID IN (SELECT id FROM Supplier WHERE CompanyName = 'Specific Supplier');

**Запросы с коррелированным подзапросом в WHERE – 2**

**1. Запрос выводит ФИО самого первого клиента, который сделал заказ**

Этот запрос выбирает фамилию, имя и отчество клиента, который первым сделал заказ.

SELECT LastName, FirstName, SecondName

FROM Customer

WHERE id = (

SELECT TOP 1 CustomerId

FROM Orders

ORDER BY OrderDate );

**2. Запрос выводит название и цену продуктов, которые дороже средней цены**

Этот запрос выбирает название и цену продуктов, которые стоят дороже средней цены всех продуктов.

SELECT ProductName, UnitPrice

FROM Product

WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Product);

### **Запрос, использующий оконную функцию LAG или LEAD**

#### Сравнение количества заказов в разные месяцы

Этот запрос помогает сравнить количество заказов в разные месяцы.

SELECT Month, OrderCount,

LAG(OrderCount) OVER (ORDER BY Month) AS PreviousMonth, LEAD(OrderCount) OVER (ORDER BY Month) AS NextMonth

FROM Orders;

**Запросы с агрегированием и выражением JOIN, включающим не менее 2 таблиц**

#### **1. Запрос, вычисляющий, сколько каждый клиент заплатил компании за определенный период**

#### SELECT c.id AS CustomerId, SUM(o.TotalAmount) AS TotalSpent

#### FROM Customer c

#### JOIN Orders o ON c.id = o.CustomerId

#### WHERE o.OrderDate BETWEEN '2024-01-01' AND '2024-01-03'

#### GROUP BY c.id;

#### **2. Запрос на подсчет количества заказов для каждого продукта**

SELECT p.ProductName, COUNT(oi.id) AS OrderCount

FROM Product p

JOIN OrderItem oi ON p.id = oi.ProductId

GROUP BY p.ProductName;

#### **3. Запрос на подсчет количества клиентов в каждой стране**

SELECT co.Name AS CountryName, COUNT(c.id) AS CustomerCount

FROM Country co

JOIN City ci ON co.id = ci.CountryId

JOIN Customer c ON ci.id = c.CityId

GROUP BY co.Name;

### **Запрос с EXISTS**

#### Запрос на нахождение клиентов, которые сделали заказы на определенный продукт

SELECT DISTINCT c.id AS CustomerId

FROM Customer c

WHERE EXISTS (

SELECT 1

FROM Orders o

JOIN OrderItem oi ON o.id = oi.OrderId

WHERE oi.ProductId = 1 AND o.CustomerId = c.id

);

### **Запрос, использующий манипуляции с множествами – 1**

SELECT OrderDate

FROM Orders o

JOIN Customer c ON o.CustomerId = c.id

JOIN City ci ON c.CityId = ci.id

WHERE ci.Name = 'City1'

INTERSECT

SELECT OrderDate

FROM Orders o

JOIN Customer c ON o.CustomerId = c.id

JOIN City ci ON c.CityId = ci.id

WHERE ci.Name = 'City2';

### **Запрос с внешним соединением и проверкой на наличие NULL – 1**

#### Запрос на нахождение клиентов, которые не сделали ни одного заказа

SELECT \*

FROM Customer c

LEFT JOIN Orders o ON c.id = o.CustomerId

WHERE o.id IS NULL;

### **Запрос с агрегированием и выражением JOIN, включающим не менее 3 таблиц/выражений – 1**

#### Выводит все продукты, их поставщиков и страны поставщиков

SELECT p.ProductName, s.CompanyName, co.Name AS CountryName

FROM Product p

JOIN Supplier s ON p.SupplierID = s.id

JOIN City ci ON s.CityId = ci.id

JOIN Country co ON ci.CountryId = co.id;

### **Запрос с CASE (IIF) и агрегированием – 1**

#### Запрос выводит название продукта, его цену и категорию цены (дорогой, средний, дешевый)

SELECT ProductName, UnitPrice,

CASE

WHEN UnitPrice > (SELECT AVG(UnitPrice) FROM Product) THEN 'Expensive'

WHEN UnitPrice < (SELECT AVG(UnitPrice) FROM Product) THEN 'Cheap'

ELSE 'Average'

END AS PriceCategory

FROM Product;

### **Запрос с HAVING и агрегированием – 1**

#### Выводит названия продуктов, у которых количество заказов больше, чем у определенного продукта

SELECT p.ProductName, COUNT(oi.id) AS OrderCount

FROM Product p

JOIN OrderItem oi ON p.id = oi.ProductId

GROUP BY p.ProductName

HAVING COUNT(oi.id) > (

SELECT COUNT(oi2.id)

FROM Product p2

JOIN OrderItem oi2 ON p2.id = oi2.ProductId

WHERE p2.id = 3

);

### **Запрос SELECT INTO для подготовки выгрузки – 1**

SELECT \*

INTO ExportedCustomers

FROM Customer;

Хранимые процедуры

#### **1. Процедура для создания расписания заказов**

Эта основная процедура составляет расписание заказов, используя курсор для прохода по каждой строке таблицы заказов. Перед добавлением новых позиций заказов удаляются старые, которые еще не были выполнены. Затем вызывается другая процедура для вставки позиций заказа в расписание.

CREATE PROCEDURE create\_schedule

AS

BEGIN

DECLARE @order\_id INT;

DECLARE @order\_date DATETIME;

DECLARE OrderCursor CURSOR FOR

SELECT id, OrderDate

FROM Orders;

OPEN OrderCursor;

FETCH NEXT FROM OrderCursor INTO @order\_id, @order\_date;

WHILE @@FETCH\_STATUS = 0

BEGIN

BEGIN TRANSACTION;

DELETE FROM Schedule

WHERE completed = 0 AND EXISTS (

SELECT 1

FROM OrderItem oi

INNER JOIN Schedule s ON s.order\_item\_id = oi.id

WHERE oi.OrderId = @order\_id

);

BEGIN TRY

EXEC insert\_order @order\_id, @order\_date;

END TRY

BEGIN CATCH

PRINT('Error in procedure');

ROLLBACK;

END CATCH;

COMMIT;

FETCH NEXT FROM OrderCursor INTO @order\_id, @order\_date;

END

CLOSE OrderCursor;

DEALLOCATE OrderCursor;

END

#### **2. Процедура для вставки заказа в расписание**

Эта процедура вставляет новую позицию заказа в расписание. Она принимает идентификатор заказа и дату заказа в качестве входных параметров.

CREATE PROCEDURE insert\_order

@order\_id INT,

@order\_date DATETIME

AS

BEGIN

INSERT INTO Schedule (order\_item\_id, start\_date, completed)

SELECT id, @order\_date, 0

FROM OrderItem

WHERE OrderId = @order\_id;

END

#### **3. Процедура для получения информации о клиенте по имени и фамилии**

Эта процедура возвращает информацию о клиенте по его имени и фамилии.

CREATE PROCEDURE GetCustomerInfoByName

@firstName NVARCHAR(100),

@lastName NVARCHAR(100)

AS

BEGIN

SELECT \*

FROM Customer

WHERE FirstName = @firstName AND LastName = @lastName;

END

#### **4. Процедура для получения информации о клиентах с общей суммой заказов более определенного значения**

Эта процедура возвращает информацию о клиентах, у которых общая сумма заказов превышает заданное значение.

CREATE PROCEDURE GetCustomersWithHighOrders

@minTotalAmount DECIMAL

AS

BEGIN

SELECT c.id, c.LastName, c.FirstName, SUM(o.TotalAmount) AS TotalAmount

FROM Customer c

JOIN Orders o ON c.id = o.CustomerId

GROUP BY c.id, c.LastName, c.FirstName

HAVING SUM(o.TotalAmount) > @minTotalAmount;

END

#### **5. Процедура для обновления информации о продукте**

Эта процедура обновляет информацию о продукте, принимая в качестве входных параметров идентификатор продукта, новое название и новую цену.

CREATE PROCEDURE UpdateProductInfo

@productId INT,

@newProductName NVARCHAR(100),

@newUnitPrice DECIMAL

AS

BEGIN

UPDATE Product

SET ProductName = @newProductName, UnitPrice = @newUnitPrice

WHERE id = @productId;

END

#### **6. Процедура для удаления заказов, которые не были выполнены в течение определенного времени**

Эта процедура удаляет заказы, которые не были выполнены в течение заданного количества дней.

CREATE PROCEDURE DeleteOldUncompletedOrders

@days INT

AS

BEGIN

DELETE FROM Orders

WHERE DATEDIFF(DAY, OrderDate, GETDATE()) > @days AND id NOT IN (

SELECT OrderId FROM OrderItem

);

END

Триггеры

Эти триггеры обновляют количество позиций заказа в таблице заказов при изменении, удалении или добавлении записей в таблицу позиций заказа.

#### **1. Триггер для обновления количества позиций заказа при удалении позиции заказа**

Этот триггер обновляет количество позиций в заказе при удалении записи из таблицы OrderItem.

CREATE TRIGGER trg\_delete\_order\_item

ON OrderItem

AFTER DELETE

AS

BEGIN

UPDATE Orders

SET Orders.TotalAmount = (SELECT SUM(UnitPrice \* Quantity) FROM OrderItem WHERE OrderId = (SELECT OrderId FROM DELETED))

WHERE Orders.id = (SELECT OrderId FROM DELETED);

END

#### **2. Триггер для обновления количества позиций заказа при добавлении новой позиции заказа**

Этот триггер обновляет количество позиций в заказе при добавлении новой записи в таблицу OrderItem.

CREATE TRIGGER trg\_insert\_order\_item

ON OrderItem

AFTER INSERT

AS

BEGIN

UPDATE Orders

SET Orders.TotalAmount = (SELECT SUM(UnitPrice \* Quantity) FROM OrderItem WHERE OrderId = (SELECT OrderId FROM INSERTED))

WHERE Orders.id = (SELECT OrderId FROM INSERTED);

END

#### **3. Триггер для обновления количества позиций заказа при изменении записи в таблице позиций заказа**

CREATE TRIGGER trg\_update\_order\_item

ON OrderItem

AFTER UPDATE

AS

BEGIN

UPDATE Orders

SET Orders.TotalAmount = (SELECT SUM(UnitPrice \* Quantity) FROM OrderItem WHERE OrderId = (SELECT OrderId FROM INSERTED))

WHERE Orders.id = (SELECT OrderId FROM INSERTED);

END

4.**Триггер для уведомления при добавлении или изменении записи в таблице клиентов**

Этот триггер генерирует ошибку с сообщением "Notify Customer Relations" каждый раз при добавлении новой записи или обновлении существующей в таблице

CREATE TRIGGER reminder1

ON Customer

AFTER INSERT, UPDATE

AS RAISERROR ('Notify Customer Relations', 16, 10);

Функции

#### **1. Функция, возвращающая все заказы конкретного клиента**

Данная функция объединяет три таблицы - Orders, Customer и OrderItem по следующим атрибутам: CustomerId, OrderId и ProductId. На выводе мы получаем id клиента, его фамилию, имя, id заказа и общую сумму заказа.

CREATE FUNCTION get\_customer\_orders()

RETURNS TABLE

AS

RETURN

(

SELECT DISTINCT o.CustomerId, c.LastName, c.FirstName, o.id AS OrderId, o.TotalAmount

FROM Orders o

JOIN Customer c ON o.CustomerId = c.id

JOIN OrderItem oi ON o.id = oi.OrderId

)

Чтобы посмотреть заказы по определенному клиенту, нужно узнать его id (например, 1) и сделать запрос:

SELECT \* FROM get\_customer\_orders()

WHERE CustomerId = 1;

Также, если известно ФИО клиента, можно ввести его следующим образом:

SELECT \* FROM get\_customer\_orders()

WHERE LastName = 'Smith' AND FirstName = 'John';

#### **2. Функция, определяющая, включен ли продукт в заказ клиента**

Данная функция объединяет три таблицы - Customer, Orders и OrderItem. В следующих таблицах идет сравнение на совпадающие атрибуты: Customer.id и Orders.CustomerId, а также Orders.id и OrderItem.OrderId.

CREATE FUNCTION is\_product\_in\_order()

RETURNS TABLE

AS

RETURN

(

SELECT DISTINCT c.id AS CustomerId, c.LastName, c.FirstName, oi.ProductId, p.ProductName

FROM Customer c

JOIN Orders o ON c.id = o.CustomerId

JOIN OrderItem oi ON o.id = oi.OrderId

JOIN Product p ON oi.ProductId = p.id

)

Проверка данных выглядит следующим образом: Например, вводим id клиента, равный 51 и название продукта 'Apple'. Если такой клиент, который заказал данный продукт, существует, то программа выведет его.

SELECT \* FROM is\_product\_in\_order()

WHERE CustomerId = 51 AND ProductName = 'Apple';

Также, если известно ФИО клиента, можно ввести его следующим образом:

SELECT \* FROM is\_product\_in\_order()

WHERE LastName = 'Doe' AND FirstName = 'Jane';

1. **Функция для получения клиента с наибольшей суммой заказов из определенной страны**

CREATE FUNCTION GetCustomer

(

@country AS nvarchar(40)

)

RETURNS

@result TABLE

(

FirstName nvarchar(40),

LastName nvarchar(40),

TotalAmount decimal(12,2)

)

AS

BEGIN

INSERT INTO @result

SELECT TOP(1) cu.LastName, cu.FirstName , sum(o.TotalAmount) AS SumTotalAmount

FROM Orders o

INNER JOIN Customer cu ON o.CustomerId=cu.id

INNER JOIN City ci ON ci.id=cu.CityId

INNER JOIN Country co ON co.id=ci.CountryId

WHERE co.Name =@country

GROUP BY CustomerId,cu.FirstName, cu.Lastname

ORDER BY SumTotalAmount DESC

RETURN

END

1. **Функция для расчета общей суммы заказа**

CREATE FUNCTION GetOrderTotal

(

@orderId AS integer

)

RETURNS decimal(12,2)

AS

BEGIN

DECLARE @totalAmount AS decimal(12,2);

SELECT @totalAmount = SUM(UnitPrice\*Quantity) FROM OrderItem

WHERE OrderId=@orderId;

RETURN @totalAmount;

END

Представления

#### **1. Представление Producs\_Orders**

Это представление объединяет таблицы Product, Supplier, OrderItem и Orders, чтобы предоставить информацию о товарах, заказанных клиентами, а также об их поставщиках.

CREATE VIEW Producs\_Orders

AS

SELECT p.ProductName as 'Название товара', oi.Quantity as 'Количество товара', oi.UnitPrice 'Стоимость в заказе', s.CompanyName as 'Название компании поставщика'

FROM Product p

LEFT JOIN Supplier s ON s.id = p.SupplierID

RIGHT JOIN OrderItem oi ON oi.ProductID = p.Id

INNER JOIN Orders o ON o.ID = oi.OrderID;

#### **2. Представление Product\_Account**

Это представление объединяет таблицы Country, City, Supplier и Product, чтобы предоставить информацию о поставщиках и их товарах, группируя данные по стране, городу и компании поставщика.

CREATE VIEW Product\_Account

as

SELECT s.CompanyName as 'Название компании' , co.Name as 'Название страны', ci.Name as 'Название города', count(p.id ) as "Количество товаров"

FROM Country co

INNER JOIN City ci ON ci.CountryId = co. Id

INNER JOIN Supplier s ON s.CityId = ci.Id

INNER JOIN Product p ON p.SupplierID = s.Id

GROUP BY s.CompanyName , co.Name , ci.Name;

Отчеты

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание*Список стран для отчета*

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание*Виды упаковок*

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

*Список городов*

Изображение выглядит как текст, чек, снимок экрана, алгебра

Автоматически созданное описание*Список поставщиков*

Изображение выглядит как текст, чек, снимок экрана

Автоматически созданное описание*Заказы покупателей*

График показывает количество заказов, сделанных разными покупателями.  
Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

График показывает количество заказанного товара по различным продуктам.

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описание

График показывает соотношение заказов в различных странах. Это связано с запросом, который подсчитывает количество заказов, сделанных в разных странах.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

График показывает соотношение количества клиентов каждого супермаркета. Это связано с запросом, который подсчитывает количество клиентов для каждого супермаркета.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

График показывает распределение магазинов по странам. Это связано с запросом, который подсчитывает количество магазинов для каждой страны.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Вывод

Таким образом, была спроектирована и разработана база данных, которую можно использовать для организации работы супермаркета. Было создано 8 сущностей: “Customer” (клиенты), “Orders” (заказы), “OrderItem” (позиции заказа), “Product” (продукты), “Supplier” (поставщики), “Country” (страны), “City” (города). База данных приведена к 3 нормальной форме. Также были сформированы отчеты и написаны процедуры, функции и триггеры, позволяющие в полной мере использовать возможности базы данных. Составлены запросы, релевантные для предметной области, а именно для управления заказами, продуктами и клиентами в супермаркете.