Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Автоматизированных систем управления (АСУ)

Контрольное домашнее задание

На тему

«Продуктовый супермаркет»

Выполнил:
студент группы БИВТ-23-2

Николаева П.А.

Проверила:
Валова А.А.

Москва, 2024

Оглавление

1.Постановка задачи

Задача состоит в проектировании базы данных для продуктового супермаркета, которая должна учитывать различные аспекты работы супермаркета, включая товарные разделы, сотрудники, графики работы, поставщиков, поставки товаров, продажи и клиентов.

БД должна осуществлять: учёт поставщиков и поставок, учёт продаж по отделам и по кассам, ведение информации о сотрудниках и их графиках работы на различных кассах, применение скидок для клиентов. Также требуется создание представлений, функций, хранимых процедур и триггеров для обеспечения удобного доступа к данным.

2. Описание структуры БД

2.1 Вербальная модель

База данных состоит из таблиц:

2.1. Вербальная модель

1. **Sections (Разделы)** – Представляют собой отделы супермаркета (например, овощи, мясо, молочные продукты).
2. **Desks (Кассы)** – Места, где происходят продажи товаров. Каждая касса связана с определенным отделом супермаркета.
3. **Employees (Сотрудники)** – Работники супермаркета.
4. **WorkSchedule (График работы)** – График работы сотрудников на различных кассах по дням.
5. **Suppliers (Поставщики)** – Компании, которые поставляют товары в супермаркет.
6. **Supplies (Поставки)** – Информация о поставках товаров от поставщиков, включая дату и сумму поставки.
7. **SupplyDetails (Детали поставки)** – Детализированная информация о каждом товаре в составе поставки, включая количество и цену.
8. **Products (Товары)** – Информация о товарах, включая название, категорию, срок хранения и связь с разделами супермаркета.
9. **Customer (Клиенты)** – Информация о клиентах, включая имя, фамилию и скидку.
10. **Sales (Продажи)** – Информация о продажах, включая дату, торговую точку и клиента.

11. **SalesDetails (Детали продажи)** – Детализированная информация о каждом товаре, проданном в рамках конкретной продажи.

2.2 Реляционная модель

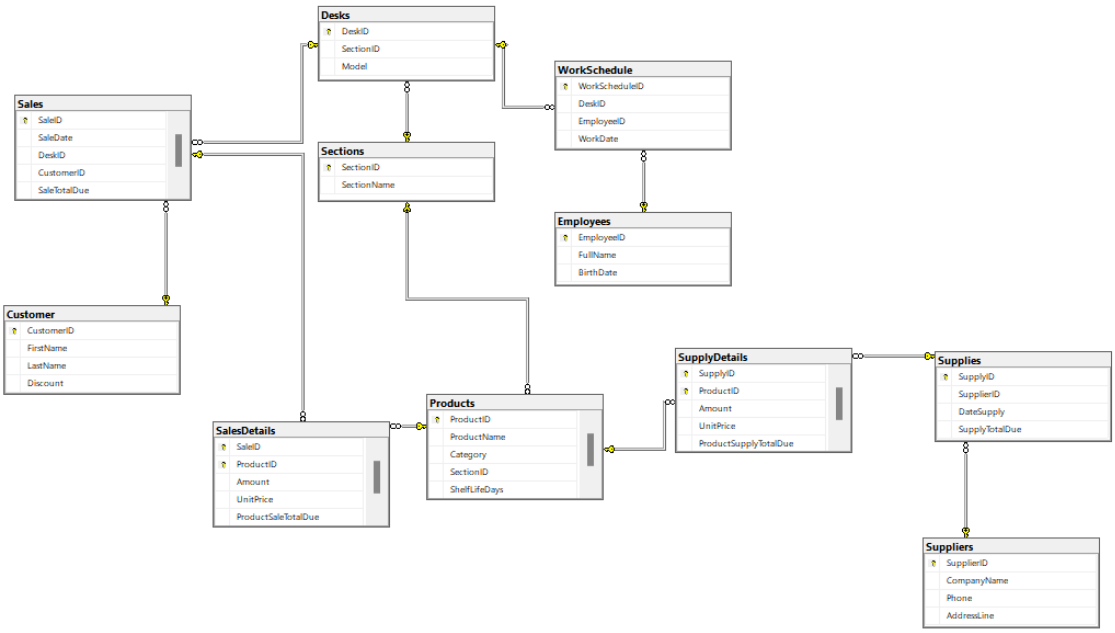

Рисунок 1 Диаграмма БД

2.3 Анализ функциональных зависимостей

Функциональные зависимости — это свойства отношений в реляционной базе данных, определяющие, какие значения в одном столбце или наборе столбцов могут быть вычислены на основе других значений в той же или других таблицах. Рассмотрим основные функциональные зависимости в созданной базе данных:

1. **Sections**:

   SectionID → SectionName

2. **Desks**:

   DeskID → SectionID, Model

3. **Employees**:

   EmployeeID → FullName, BirthDate

4. **WorkSchedule**:

WorkScheduleID → DeskID, EmployeeID, WorkDate

5. **Suppliers**:

   SupplierID → CompanyName, Phone, AddressLine

6. **Supplies**:

   SupplyID → SupplierID, DateSupply, SupplyTotalDue

7. **Products**:

   ProductID → ProductName, Category, SectionID, ShelfLifeDays

8. **SupplyDetails**:

   (SupplyID, ProductID) → Amount, UnitPrice, ProductSupplyTotalDue

9. **Customer**:

   CustomerID → FirstName, LastName, Discount

10. **Sales**:

    SaleID → SaleDate, DeskID, CustomerID, SaleTotalDue

11. **SalesDetails**:

    (SaleID, ProductID) → Amount, UnitPrice, ProductSaleTotalDue

В соответствии с анализом функциональных зависимостей, все таблицы БД соответствуют 3-й нормальной форме (3NF), так как: каждая таблица имеет уникальный первичный ключ, все атрибуты являются функциями от первичного ключа, не существует транзитивных зависимостей.

3. Заполнение БД информацией

Заполнение таблицы Desks

```sql
INSERT INTO Desks (SectionID, Model)
VALUES
(1, 'CashDesk-Model-A'),
(1, 'CashDesk-Model-B'),
(2, 'CashDesk-Model-C'),
(2, 'CashDesk-Model-D'),
(3, 'CashDesk-Model-E'),
(3, 'CashDesk-Model-F'),
(4, 'CashDesk-Model-G'),
(4, 'CashDesk-Model-H'),
(5, 'CashDesk-Model-I'),
(5, 'CashDesk-Model-J'),
(6, 'CashDesk-Model-K'),
(6, 'CashDesk-Model-L');
```

Рисунок 2 Заполнение таблицы Desks

Заполнение таблицы Employees

```sql
INSERT INTO Employees (FullName, BirthDate)
VALUES
('John Smith', '1985-05-15'),
('Sarah Johnson', '1990-07-20'),
('Michael Brown', '1982-11-11'),
('Emily Davis', '1995-03-08'),
('David Wilson', '1988-06-30'),
('Olivia Garcia', '1993-12-12'),
('James Martinez', '1987-08-25'),
('Sophia Hernandez', '1992-03-15'),
('William Lee', '1986-01-20'),
('Isabella Lopez', '1991-10-05');
```

Рисунок 3 Заполнение таблицы Employees

Заполнение таблицы Customers

```sql
INSERT INTO Customer (FirstName, LastName, Discount)
VALUES
('Alice', 'Brown', 3.00),
('Ethan', 'Miller', 5.00),
('Sophia', 'Davis', 7.00),
('Jacob', 'Garcia', 10.00),
('Mason', 'Martinez', 3.00),
('Ella', 'Lopez', 5.00),
('Liam', 'White', 7.00),
('Ava', 'Hernandez', 10.00),
('Lucas', 'Moore', 3.00),
('Emma', 'Clark', 5.00);
```

Рисунок 4 Заполнение таблицы Customers

4. Описание представлений

1. Представление **vDesksSalesSummary** предназначено для отображения общей суммы продаж по каждой кассе (DeskID) с учетом скидок для клиентов. Оно вычисляет сумму всех продаж, уменьшенную на процент скидки клиента. Для каждой кассы, на которой были совершены продажи, будет выводиться итоговая сумма продаж с учетом скидок.

```
CREATE VIEW vDesksSalesSummary AS

SELECT Sales.DeskID, SUM(Sales.SaleTotalDue * (1 - Customer.Discount / 100)) AS TotalSales

FROM Sales

JOIN Customer ON Customer.CustomerID = Sales.CustomerID

GROUP BY Sales.DeskID;
```

2. Представление **vSectionsSalesSummary** предназначено для отображения общей суммы продаж по отделам с учетом скидок для клиентов. Оно объединяет таблицы Sales, Customer, SalesDetails, Products и Sections, что позволяет анализировать доходность каждой секции.

```
CREATE VIEW vSectionsSalesSummary AS

SELECT sec.SectionID, sec.SectionName, SUM(s.SaleTotalDue * (1 - Customer.Discount / 100)) AS TotalSales

FROM Sales as s

JOIN Customer ON Customer.CustomerID = s.CustomerID

JOIN SalesDetails as sd ON sd.SaleID = s.SaleID

JOIN Products as pr ON pr.ProductID = sd.ProductID

JOIN Sections as sec ON pr.SectionID = sec.SectionID

GROUP BY sec.SectionID, sec.SectionName;
```

3. Представление **vSalesWithDiscounts** объединяет таблицы Customer и Sales, чтобы отобразить информацию о клиентах, их скидках и суммах покупок. Оно вычисляет итоговую сумму покупки с учетом предоставленной клиенту скидки, позволяя получить данные о скидках и реальной стоимости каждой продажи. Предоставляет удобный набор данных, описывающих покупателей и совершенные ими покупки.

```
CREATE VIEW vSalesWithDiscounts AS
```

```
SELECT c.CustomerID, c.Discount, s.SaleDate,
     s.SaleTotalDue,(s.SaleTotalDue * (1 - c.Discount / 100)) AS
DiscountedAmount
FROM Customer AS c
JOIN Sales s ON c.CustomerID = s.CustomerID
GROUP BY c.CustomerID, c.Discount, s.SaleDate, s.SaleTotalDue;
```

4. Обновляемое представление, **vEmployee,** предназначено для получения списка сотрудников магазина. Это обновляемое представление (updatable view) содержит в себе краткую информацию о сотрудниках магазина, и имеет функции удаления, внесения и обновления информации в данной таблице и связанной с ней таблицей Employee.

```
 CREATE VIEW vEmployee AS
SELECT EmployeeID, FullName, BirthDate
FROM Employees
```

5. Описание функций

1. Табличная функция **GetSalesByDateRange** предназначена для получения информации о продажах за определенный период. Она принимает два параметра: @StartDate и @EndDate, которые задают диапазон дат. Возвращаемая таблица включает идентификатор продажи (SaleID), дату продажи (SaleDate), общую сумму продажи (SaleTotalDue) и имя клиента, совершившего покупку (CustomerName).

```
CREATE FUNCTION GetSalesByDateRange (@StartDate DATE, @EndDate DATE)
RETURNS TABLE
AS
RETURN (
    SELECT s.SaleID, s.SaleDate, s.SaleTotalDue, c.FirstName + ' ' +
c.LastName AS CustomerName
    FROM Sales AS s
    JOIN Customer AS c ON s.CustomerID = c.CustomerID
    WHERE s.SaleDate BETWEEN @StartDate AND @EndDate
```

```
);
```

2. Скалярная функция **GetTotalAmountSpentByCustomer** предназначена для вычисления общей суммы, потраченной клиентом, идентификатор которого передается в качестве параметра @CustomerID. Она использует курсор для перебора всех записей о покупках клиента из представления vSalesWithDiscounts, извлекая поле DiscountedAmount и суммируя значения в переменной @Sum. В конце функция возвращает итоговую сумму в формате DECIMAL(10,2).

```
CREATE FUNCTION GetTotalAmountSpentByCustomer(@CustomerID INT)
RETURNS DECIMAL(10,2)
BEGIN

    DECLARE @Sum DECIMAL(10, 2) = 0;
    DECLARE @CursorSum  DECIMAL(10, 2);

    DECLARE MYCURSOR CURSOR FAST_FORWARD FOR
    SELECT DiscountedAmount
    FROM vSalesWithDiscounts
    WHERE vSalesWithDiscounts.CustomerID = @CustomerID;

    OPEN MYCURSOR;
    FETCH NEXT FROM MYCURSOR INTO @CursorSum;
    WHILE @@FETCH_STATUS = 0
    BEGIN
    SET @Sum=@Sum+@CursorSum
    FETCH NEXT FROM MYCURSOR INTO @CursorSum;
    END
    CLOSE MYCURSOR;
    DEALLOCATE MYCURSOR;
```

```
      RETURN @sum
END
```

3. Табличная функция **GetProductStock** предназначена для вычисления текущего остатка товаров на складе. Она возвращает таблицу с информацией о каждом товаре, включая его идентификатор (ProductID), название (ProductName) и расчетный текущий остаток (CurrentStock). Остаток вычисляется как разница между общим количеством поступившего товара (SupplyDetails.Amount) и проданного товара (SalesDetails.Amount).

```
CREATE FUNCTION GetProductStock()
RETURNS TABLE
AS
RETURN
(
    SELECT p.ProductID, p.ProductName, ISNULL(SUM(sd.Amount), 0) -
ISNULL(SUM(saled.Amount), 0) AS CurrentStock
    FROM Products p
    LEFT JOIN SupplyDetails sd ON p.ProductID = sd.ProductID
    LEFT JOIN SalesDetails saled ON p.ProductID = saled.ProductID
    GROUP BY p.ProductID, p.ProductName
);
```

6. Описание хранимых процедур

1. Это 2 хранимых процедур pr_InsertIntoEmpView и pr_DeleteFromvEmployee предназначена для добавления и удаление нового сотрудника магазина. Процедуры созданы с использованием транзакции для обеспечения атомарности операций и управления целостностью данных в случае возникновения ошибок.

```
CREATE PROCEDURE pr_InsertIntovEmployee (@FullName
NVARCHAR(100),@BirthDate DATE)
AS
```

```
BEGIN
      BEGIN TRANSACTION
      INSERT INTO vEmployee(FullName,BirthDate)
      VALUES (@FullName, @BirthDate)

      IF @@ERROR = 0
      BEGIN
            COMMIT TRANSACTION;
            PRINT 'Новый сотрудник магазина успешно добавлен'
      END
      ELSE
      BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Произошла ошибка. Транзакция отменена'
      END
END
CREATE PROCEDURE pr_DeleteFromvEmployee (@EmployeeID INT)
AS
BEGIN
    BEGIN TRANSACTION;

    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID =
@EmployeeID)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Ошибка: Сотрудник с таким ID не найден.';
            RETURN;
```

```
        END

        DELETE FROM Employees

        WHERE EmployeeID = @EmployeeID;

        COMMIT TRANSACTION;

        PRINT 'Сотрудник успешно удален';

    END TRY

    BEGIN CATCH

        ROLLBACK TRANSACTION;

        PRINT 'Произошла ошибка при удалении сотрудника: ' +
ERROR_MESSAGE();

    END CATCH;

END;
```

2. Процедура AddNewSupply добавляет новую поставку в базу данных, проверяя существование поставщика и всех товаров из списка поставки. Она принимает параметры: идентификатор поставщика, дату поставки и таблицу с деталями поставки, которая включает идентификаторы товаров, количество и цену. После проверки данных, процедура создает запись о поставке в таблице Supplies, вставляет детали поставки в SupplyDetails, рассчитывает общую сумму поставки, суммируя стоимость товаров с учетом их количества и цены, и обновляет запись о поставке с вычисленной общей суммой. Я создала пользовательский тип данных SupplyDetailsType, представляющий собой таблицу с тремя колонками: ProductID (идентификатор продукта), Amount (количество) и UnitPrice (цена за единицу) для передачи данных о товаре, поставляемом в рамках поставки.

```
CREATE TYPE SupplyDetailsType AS TABLE
(
    ProductID INT,
    Amount INT,
    UnitPrice DECIMAL(10, 2)
);
GO
```

```
CREATE PROCEDURE AddNewSupply
    @SupplierID INT,
    @DateSupply DATE,
    @SupplyDetails SupplyDetailsType READONLY
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Suppliers WHERE SupplierID =
@SupplierID)
    BEGIN
        RAISERROR('Указанный поставщик не существует!', 16, 1);
        RETURN;
    END;
    IF EXISTS (SELECT 1 FROM @SupplyDetails sd WHERE NOT EXISTS
(SELECT 1 FROM Products p WHERE p.ProductID = sd.ProductID))
    BEGIN
        RAISERROR('Один или несколько продуктов не существуют!', 16,
1);
        RETURN;
    END;
    DECLARE @SupplyID INT;
    INSERT INTO Supplies (SupplierID, DateSupply, SupplyTotalDue)
    VALUES (@SupplierID, @DateSupply, 0);
    SET @SupplyID = SCOPE_IDENTITY();
    DECLARE @TotalDue DECIMAL(10, 2) = 0;

    INSERT INTO SupplyDetails (SupplyID, ProductID, Amount, UnitPrice)
    SELECT @SupplyID, ProductID, Amount, UnitPrice
```

```
    FROM @SupplyDetails;

    DECLARE cur_supply CURSOR FOR
    SELECT ProductID, Amount, UnitPrice
    FROM SupplyDetails
    WHERE SupplyID = @SupplyID;
    DECLARE @ProductID INT, @Amount INT, @UnitPrice DECIMAL(10, 2);
    OPEN cur_supply;
    FETCH NEXT FROM cur_supply INTO @ProductID, @Amount, @UnitPrice;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @TotalDue = @TotalDue + (@Amount * @UnitPrice);
        FETCH NEXT FROM cur_supply INTO @ProductID, @Amount,
@UnitPrice;
    END;
    CLOSE cur_supply;
    DEALLOCATE cur_supply;

    UPDATE Supplies
    SET SupplyTotalDue = @TotalDue
    WHERE SupplyID = @SupplyID;
END;
```

3. Хранимая процедура AddNewSale добавляет новую запись о продаже в таблицы Sales и SalesDetails, учитывая переданные данные о покупателе и товарах. Если покупатель не существует, он добавляется в таблицу Customer. Процедура рассчитывает общую сумму продажи и обновляет поле SaleTotalDue в таблице Sales. Созданный тип данных SalesDetailsType представляет собой таблицу с колонками ProductID, Amount и UnitPrice, используемую для передачи информации о товарах в процессе продажи.

```
CREATE TYPE SalesDetailsType AS TABLE
(
    ProductID INT,
    Amount INT,
    UnitPrice DECIMAL(10, 2)
);
GO

CREATE PROCEDURE AddNewSale
    @CustomerID INT,
    @DeskID INT,
    @SaleDate DATE,
    @SalesDetails SalesDetailsType READONLY
AS
BEGIN

    IF NOT EXISTS (SELECT 1 FROM Customer WHERE CustomerID =
@CustomerID)
    BEGIN
           SET IDENTITY_INSERT Customer ON;
        INSERT INTO Customer (CustomerID, FirstName, LastName,
Discount)
        VALUES (@CustomerID, 'Unknown', 'Unknown', 0);
    END;

    DECLARE @SaleID INT;
    INSERT INTO Sales (SaleDate, DeskID, CustomerID, SaleTotalDue)
    VALUES (@SaleDate, @DeskID, @CustomerID, 0);
```

```
    SET @SaleID = SCOPE_IDENTITY();

    DECLARE @SaleTotalDue DECIMAL(10, 2) = 0;
    DECLARE @ProductID INT, @Amount INT, @UnitPrice DECIMAL(10, 2);

    DECLARE sales_cursor CURSOR FOR
        SELECT ProductID, Amount, UnitPrice
        FROM @SalesDetails;

    OPEN sales_cursor;
    FETCH NEXT FROM sales_cursor INTO @ProductID, @Amount, @UnitPrice;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO SalesDetails (SaleID, ProductID, Amount,
UnitPrice)
        VALUES (@SaleID, @ProductID, @Amount, @UnitPrice);
        SET @SaleTotalDue = @SaleTotalDue + (@Amount * @UnitPrice);
        FETCH NEXT FROM sales_cursor INTO @ProductID, @Amount,
@UnitPrice;
    END;
    CLOSE sales_cursor;
    DEALLOCATE sales_cursor;
    UPDATE Sales
    SET SaleTotalDue = @SaleTotalDue
    WHERE SaleID = @SaleID;
END;
```

4. Хранимая процедура CustomerDiscountUpdate обновляет скидку для клиента на основе общей суммы, которую он потратил. Она использует функцию GetTotalAmountSpentByCustomer, чтобы получить сумму покупок клиента. Таким образом, процедура автоматически регулирует размер скидки в зависимости от уровня потраченных средств клиентом.

```
 CREATE PROCEDURE CustomerDiscountUpdate(@CustomerID INT)
AS
BEGIN
    IF (dbo.GetTotalAmountSpentByCustomer(@CustomerID) > 50.00)
    BEGIN
        UPDATE Customer
        SET Discount = 7
        FROM Customer
        WHERE Customer.CustomerID = @CustomerID
    END

    IF (dbo.GetTotalAmountSpentByCustomer(@CustomerID) > 100.00 )
    BEGIN
        UPDATE Customer
        SET Discount = 10
        FROM Customer
        WHERE Customer.CustomerID = @CustomerID
    END

    IF (dbo.GetTotalAmountSpentByCustomer(@CustomerID) > 150.00)
    BEGIN
        UPDATE Customer
        SET Discount = 20
```

```
        FROM Customer
        WHERE Customer.CustomerID = @CustomerID
    END

END;
```

7. Описание триггеров

1. Триггер **trg_DeletingSuppliesAndDetails** срабатывает при попытке удаления записей из таблицы Supplies. Вместо простого удаления из Supplies, он удаляет связанные записи в таблице SupplyDetails, а затем удаляет запись из Supplies. Это предотвращает оставление "осиротевших" данных в таблице SupplyDetails при удалении поставки.

```
CREATE TRIGGER trg_DeletingSuppliesAndDetails
ON Supplies INSTEAD OF DELETE
AS
BEGIN
    DECLARE @SupplyID INT;
    DECLARE cur_for_delete CURSOR FOR
    SELECT SupplyID FROM deleted;

    OPEN cur_for_delete;

    FETCH NEXT FROM cur_for_delete INTO @SupplyID;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        DELETE FROM SupplyDetails WHERE SupplyID = @SupplyID;
        DELETE FROM Supplies WHERE SupplyID = @SupplyID;
        FETCH NEXT FROM cur_for_delete INTO @SupplyID;
    END;
    CLOSE cur_for_delete;
```

```
    DEALLOCATE cur_for_delete;
END;
```

2. Триггер **trg_CheckStockOnInsert** срабатывает после вставки новых данных в таблицу SalesDetails. Он проверяет, достаточно ли товара на складе для выполнения нового заказа. Если товара недостаточно, триггер вызывает ошибку и отменяет вставку, чтобы предотвратить создание заказа с недостаточным количеством товара.

```
CREATE TRIGGER trg_CheckStockOnInsert
ON SalesDetails AFTER INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted
        JOIN GetProductStock()  ON inserted.ProductID = GetProductStock.ProductID
        WHERE GetProductStock.CurrentStock < inserted.Amount
    )
    BEGIN
        RAISERROR (N'Недостаточно товаров на складе для добавления в заказ!', 16, 1);
        ROLLBACK;
    END;
END;
```

3. Триггер **trg_DeleteEmployee** срабатывает при попытке удаления записи из таблицы Employees и использует курсор для перебора всех удаляемых сотрудников. Для каждого сотрудника, который будет удален, сначала удаляются связанные записи из таблицы WorkSchedule, а затем сам сотрудник удаляется из таблицы Employees. Триггер также выводит сообщения о том, какие записи были удалены. Это необходимо для обеспечения целостности данных, чтобы не оставались "осиротевшие" записи в других таблицах, связанных с удаляемым сотрудником(WorkSchedule).

```
CREATE TRIGGER trg_DeleteEmployee
ON Employees INSTEAD OF DELETE
AS
BEGIN
    DECLARE @EmployeeID INT;

    DECLARE cur_employee CURSOR FOR
    SELECT EmployeeID
    FROM DELETED;
    OPEN cur_employee;
    FETCH NEXT FROM cur_employee INTO @EmployeeID;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        DELETE FROM WorkSchedule WHERE EmployeeID = @EmployeeID;
        PRINT 'Удалены записи из WorkSchedule для сотрудника с
EmployeeID = ' + CAST(@EmployeeID AS VARCHAR(10));
        DELETE FROM Employees WHERE EmployeeID = @EmployeeID;
        PRINT 'Удален сотрудник с EmployeeID = ' + CAST(@EmployeeID
AS VARCHAR(10));
        FETCH NEXT FROM cur_employee INTO @EmployeeID;
    END;
    CLOSE cur_employee;
    DEALLOCATE cur_employee;
END;
```

8. Примеры работы с БД с использованием созданных объектов

1.Запрос для представления vSectionsSalesSummary

```
SELECT * FROM vSectionsSalesSummary;
```

Результаты Сообщения

	SectionID	SectionName	TotalSales
1	1	Fruits and Vegetables	5964.50800000
2	2	Dairy Products	1107.57900000
3	3	Meat and Fish	793.60600000
4	4	Bakery	798.26400000
5	5	Beverages	770.23000000
6	6	Confectionery	867.70500000

2. Проверка выполнение функции GetProductsStock()

```
SELECT * FROM GetProductStock();
```

Результаты Сообщения

	ProductID	ProductName	CurrentStock
1	1	Apples	1430
2	2	Bananas	1564
3	3	Potatoes	1868
4	4	Carrots	1626
5	5	Milk	3274
6	6	Cheese	2690
7	7	Chicken Breast	602
8	8	Beef	458
9	9	Bread	655
10	10	Croissants	317
11	11	Mineral Water	1757
12	12	Orange Juice	1169
13	13	Chocolate	869
14	14	Cookies	569

3. Выполнение процедуры InsertIntoEmpView

```
EXEC pr_InsertIntovEmployee @FullName='Nikoleva Polina', @BirthDate='2005-12-20';
```

```
(затронута одна строка)
Новый сотрудник магазина успешно добавлен
```

4. Проверка работы триггера **trg_DeleteEmployee.** Для каждого сотрудника, сначала были удалены связанные записи из таблицы WorkSchedule, а затем сам сотрудник удален из таблицы Employees.

```
BEGIN TRANSACTION
SELECT * FROM Employees WHERE EmployeeID in (9,10);
SELECT * FROM WorkSchedule WHERE EmployeeID in (9,10);
DELETE FROM Employees WHERE EmployeeID in (9,10);
SELECT * FROM Employees WHERE EmployeeID in (9,10);
SELECT * FROM WorkSchedule WHERE EmployeeID in (9,10);
ROLLBACK TRANSACTION
```

80 %

Результаты | Сообщения

	EmployeeID	FullName	BirthDate
1	9	William Lee	1986-01-20
2	10	Isabella Lopez	1991-10-05

	WorkScheduleID	DeskID	EmployeeID	WorkDate
1	15	3	9	2024-12-03
2	16	4	10	2024-12-03
3	25	1	9	2024-12-05
4	26	2	10	2024-12-05

	EmployeeID	FullName	BirthDate

	WorkScheduleID	DeskID	EmployeeID	WorkDate

Результаты | Сообщения

```
(затронуто строк: 2)

(затронуто строк: 4)

(затронуто строк: 2)
Удалены записи из WorkSchedule для сотрудника с EmployeeID = 9

(затронута одна строка)
Удален сотрудник с EmployeeID = 9

(затронуто строк: 2)
Удалены записи из WorkSchedule для сотрудника с EmployeeID = 10

(затронута одна строка)
Удален сотрудник с EmployeeID = 10

(затронуто строк: 2)

(затронуто строк: 0)

(затронуто строк: 0)

Время выполнения: 2024-12-23T00:36:06.6451854+03:00
```

9.Список литературы

- Документация Microsoft SQL

Документация по Microsoft SQL - SQL Server | Microsoft Learn

- Сайт Metanit (Руководство по MS SQL Server 2022)

  MS SQL Server 2022 и T-SQL - METANIT.COM