Bazy Danych - projekt
Sklep Internetowy

Karolina Prygiel Mateusz Gwóźdź

Rozdział 1

Podstawowe założenia projektu

1.1 Cel i główne założenia projektu

Celem projektu jest implementacja bazdy danych do obsługi sklepu internetowego z kosmetykami.
Baza danych przechowuje informacje o **klientach**, aktualnej zawartości ich **koszyków**, a także o dokonanych przez nich **zamówieniach**. Każdy klient po dokonaniu zamówienia może wystawić recenzję zakupionym produktom lub ocenić istniejący już komentarz.
Przechowywane są również informacje o **produktach** dostępnych w sklepie. Ich **kategoriach**, **podkategoriach**, **markach** oraz aktualnym stanie w magazynie. Ponadto baza zawiera informacje o **pracownikach** sklepu, którzy zatrudnieni są w pewnych **działach** a także o **dostawach** produktów i ich **dostawcach**.

1.2 Klienci

Klienci są dodawani do bazy za pomocą procedury Add Customer, która jednocześnie przypisuje im pusty koszyk na zakupy oraz dodaje ich do listy mailingowej, co umożliwia późniejsze wysyłanie newslettera. Po pomyślnym dodaniu do bazy, klient otrzymuje maila powitalnego - trigger Confirmation Email, przez który musi potwierdzić swoje konto.

1.3 Dodawanie produktów do koszyka oraz składanie zamówień

Produkty można zakupić wyłącznie poprzez uprzednie dodanie ich do koszyka. Służy do tego procedura Add To Cart. Można dodawać wiele sztuk tego samego produktu oraz usuwać wybrane artykuły z koszyka - procedura Delete From Cart. Zostały zaimplemeentowane zabezpieczenia chroniące przed dodaniem/usunięciem z koszyka nieprawidłowej ilości produktów.
Złożenia zamówienia realizuje procedura Buy, która przepisuje zawartość koszyka do tabeli Order Details, następnie usuwa go i przydziela klientowi nowy (pusty) koszyk. Losowo przydzielany jest również pracownik, który spakuje zamówienie.
Możliwe jest również wygenerowanie faktury - funkcja Generate Note, zawierającej informajce o zamawiającym, dacie i całkowitej wartości zamówienia oraz o ilości kupionych produktów.

1.4 Opinie o produktach

Klienci mogą wystawiać opinie oraz oceniać produkty w skali 1-5. Ponadto można oceniać recenzje innych użytkowników pozytywnie /negatywnie odpowiednio poprzez '+' oraz '-'. Za realizację czego odpowiadają procedury Add Review oraz Add Review Rating.
Zostały zaimplementowane funkcje służące do manipulowania opiniami. Można wyświetlić wszystkie recenzje na temat danego produktu - funkcja Get Reviews, wyświetlić jego średnią ocenę (w skali 1-5) - funkcja Avg. Product Rate, a także wyświetlić ilosć '+' i '-' poszczególnych recenzji - Get Review Ratings.

1.5 Produkty

Baza danych przechowuje informacje na temat produktów dostępnych w sklepie.
Każdy produkt posiada przypisaną kategorię, podkategorię oraz jest produkowany przez określoną markę. Ze względu na to, że jest to sklep z kosmetykami, pojęcie marki ma tu szczególnie duże znaczenie. Tabela **brands**

przechowuje informacje o dostępnych w sklepie markach. Wyzwalacz Assign Brand Id dla każdego nowego produktu, sprawdza czy jego marka istnieje już w bazie i jeśli tak nie jest - automatycznie dodaje ją do tabeli Brands. Również gdy w ofercie sklepu pojawi się zupełnie nowy produkt, do klientów sklepu zostaje wysłany mail informujący o dostępności nowych produktów - wyzwalacz New Product Notification.
Przechowujemy również informacje na temat **ilości** dostępnych produktów w magazynie. Widok Storage Status pozwala w przejrzysty sposób uzyskać informacje na temat stanu wszystkich produktów w bazie, a gdy któregoś z produktów zaczyna brakować, trigger Stock Alert zawiadamia personel o wyczerpaniu jego zapasów.
Przyjmujemy, że każdy produkt jest dostarczany przez jednego **dostawcę**. W bazie przechowujemy informacje o poszczególnych **dostawach** i ich **szczegółach**.

1.6 Wyszukiwanie produktów

Zaprejktowano 2 sposoby wyszukiwania produktów w bazie:

1. Funkcja Search pozwalająca określić:

   - Markę;
   - kategorię;
   - podkategorię;
   - przedział cenowy.

   Wszystkie parametry są opcjonalne, a niesprecyzowanie któregokolwiek z nich powoduje wyświetlenie produktów ze wszystkich dostępnych marek, kategorii, przedziałów cenowych itd. Dzięki temu możemy wyszukiwać produkty tylko z określonej markii, kategorii lub w pewnym przedziale cenowym.

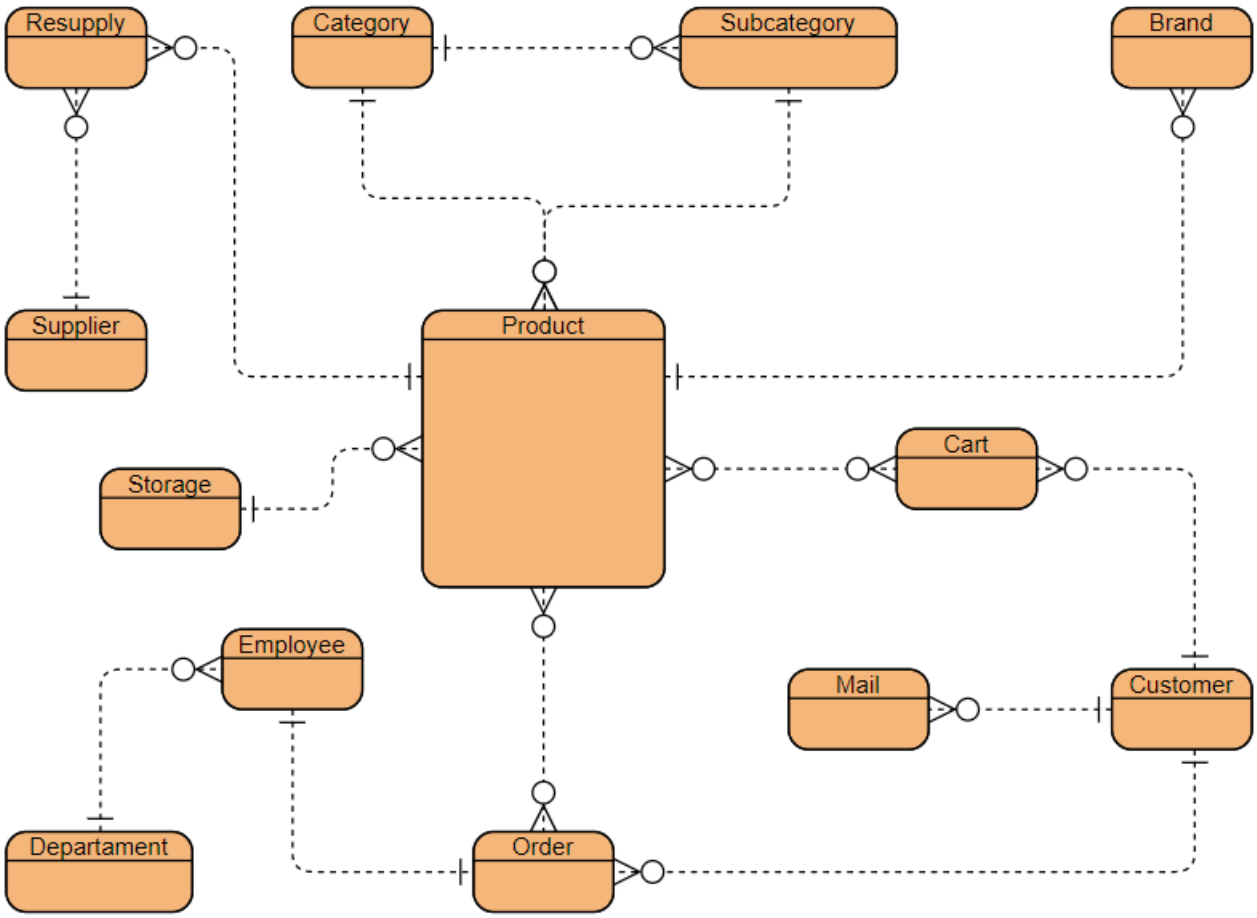
2. Funkcja Product Lookup pozwalająca wyszukiwać produkty po fragmencie jego nazwy lub nazwy jego marki.

1.7 Przyjęte ograniczenia

Baza nie obsługuje zwrotów produktów, reklamacji a także informacji o sposobie realizacji dostaw i zamówień przez klientów. Ponaddto nie utworzono kont użytkowników, przechowujących hasła i umożlwiających m.in logowanie się do sklepu.

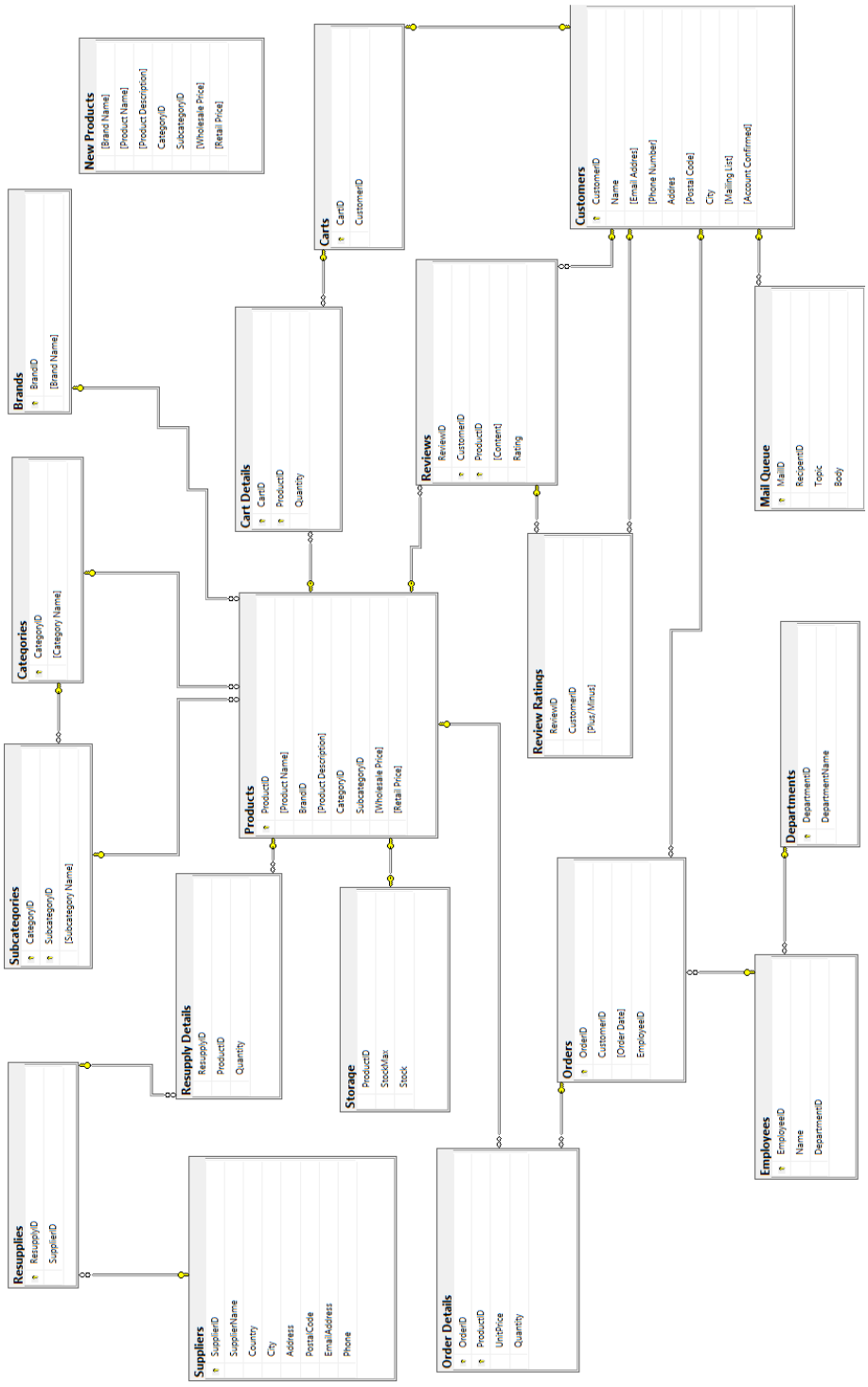
Rozdział 2

Diagram ER



Rozdział 3

Schemat bazy danych



Rozdział 4

Widoki

4.1 Storage Status

Widok przedstawiający informacje na temat aktualnego stanu magazynu.

	ProductID	Product Name	Stock
1	1	Colorstay	994 of 1000
2	2	X-CEPTIONAL WEAR	991 of 1000
3	3	GOLDEN FAIREST	893 of 1000

Widok *Storage Status* dla przykładowych danych

```
USE TestDB;
GO

DROP VIEW IF EXISTS [dbo].[Storage Status];
GO

CREATE VIEW [Storage Status]
AS
SELECT P.ProductID, P.[Product Name],
    CONVERT(VARCHAR(10), S.Stock) + ' of ' + CONVERT(VARCHAR(10),
    S.StockMax) AS [Stock]
FROM [dbo].[Storage] AS S
RIGHT JOIN [dbo].[Products] AS P ON S.ProductID = P.ProductID

GO
```

Kod generujący widok *Storage Status*

4.2 Prodcts Bought

Widok przedstawiający informacje na temat aktualnej ilości sumy zamówień wszystkich produktów.

	Brand	Product Name	Quantity
1	Anabelle Minerals	GOLDEN FAIREST	7
2	Revlon	Colorstay	5
3	Gosh	X-CEPTIONAL WEAR	4

Widok *Products Bought* dla przykładowych danych

```sql
USE TestDB;
GO

DROP VIEW IF EXISTS [dbo].[Products Bought];
GO

CREATE VIEW [Products Bought]
AS
SELECT B.[Brand Name] AS [Brand], P.[Product Name], Pr.[Quantity]
FROM Products AS P
JOIN (SELECT P.ProductID, SUM(OD.Quantity) AS [Quantity]
    FROM [dbo].[Products] AS P
    JOIN [dbo].[Order Details] AS OD ON P.ProductID = OD.ProductID
    GROUP BY P.ProductID) AS Pr ON P.ProductID = Pr.ProductID
JOIN [dbo].[Brands] AS B ON P.BrandID = B.BrandID

GO
```

Kod generujący widok *Prodcts Bought*

4.3 Hot Brands

Widok przedstawiający informacje na temat najchętniej kupowanych marek.

	Brand Name	Total Quantity
1	Anabelle Minerals	7
2	Revlon	5
3	Gosh	4

Widok *Hot Brands* dla przykładowych danych

```sql
USE TestDB
;
GO

DROP VIEW IF EXISTS [dbo].[Hot Brands]
;
GO

CREATE VIEW [Hot Brands]
AS
SELECT TOP 3 Brands.[Brand Name], SUM(Quantity) AS [Total Quantity] FROM [Order Details] OD
    JOIN Products ON OD.ProductID = Products.ProductID
JOIN Brands ON Products.BrandId = Brands.BrandId
GROUP BY ([Brand Name])
Order by [Total Quantity] DESC

;
GO
```

Kod generujący widok *Hot Brands*

4.4 Category Organisation

Widok przedstawiający strukturę kategorii i podkategorii będących podziałami produktów.

CategoryID	Category Name	SubcategoryID	Subcategory Name
3	Oczy	21	Pedzle do oczu
3	Oczy	22	Akcesoria
4	Brwi	23	Kredki do brwi
4	Brwi	24	Cienie
4	Brwi	25	Korektory do brwi
4	Brwi	26	Maskary, zele, woski, po...
4	Brwi	27	Pedzle do brwi
4	Brwi	28	Akcesoria
5	Paznokcie	29	Lakiery do paznokci
5	Paznokcie	30	Lakiery hybrydowe
5	Paznokcie	31	Pielegnacja
5	Paznokcie	32	Zmywacze

Widok *Category Organisation* dla przykładowych danych

```
USE TestDB;
GO

DROP VIEW IF EXISTS [dbo].[Category Organisation];
GO

CREATE VIEW [Category Organisation]
AS
SELECT C.CategoryID, C.[Category Name], S.SubcategoryID, S.[Subcategory Name]
FROM [dbo].[Categories] AS C
LEFT JOIN [dbo].[Subcategories] AS S ON C.CategoryID = S.CategoryID

GO
```

Kod generujący widok *Category Organisation*

Rozdział 5

Procedury składowane

5.1 Add Customer

Procedura pozwalająca dodać nowego klienta do tabeli *Customers*. Automatycznie zostaje mu przypisany nowy, pusty *koszyk*.

```
USE TestDB;
GO

DROP PROC IF EXISTS dbo.AddCustomer;
GO

CREATE PROC dbo.AddCustomer

@Name NVARCHAR(40) = NULL,
@Email NVARCHAR(40) = NULL,
@PhoneNumber NVARCHAR(5) = NULL,
@Addres NVARCHAR(40) = NULL,
@PostalCode NVARCHAR(15) = NULL,
@City NVARCHAR(25) = NULL

AS

DECLARE @error AS NVARCHAR(400);

IF @Name IS NULL OR @Email IS NULL
BEGIN
     SET @error = 'Error!';
     RAISERROR(@error, 16,1);
     RETURN;
END

INSERT INTO Customers([Name], [Email Addres], [Phone Number], [Addres], [Postal Code], City)
VALUES (@Name, @Email, @PhoneNumber, @Addres, @PostalCode, @City);


INSERT INTO Carts(CustomerID)
VALUES ((SELECT CustomerID FROM Customers WHERE [Email Addres] = @Email))

GO
```

Kod generujący procedurę *Add Customer*

5.2 Add Review

Procedura służąca do dodawania recenzji na temat produktu.

```
USE TestDB;
GO

DROP PROC IF EXISTS dbo.AddReview;
GO

CREATE PROC dbo.AddReview
@CustomerID INT = NULL,
@ProductID INT = NULL,
@Content NTEXT = NULL,
@Rating TINYINT = NULL

AS
  DECLARE @error AS NVARCHAR(50);

  IF @CustomerID IS NULL OR @ProductID IS NULL OR @Rating IS NULL
  BEGIN
    SET @error = 'Error! Rating lacking information.';
    RAISERROR(@error, 16,1);
    RETURN;
  END

  INSERT INTO dbo.[Reviews] VALUES
  (@CustomerID, @ProductID, @Content, @Rating)

GO
```

Kod generujący procedurę *Add Review*

5.3 Add Review Rating

Procedura służąca do ocenienia recenzji.

```
IF OBJECT_ID('dbo.AddReviewRating', 'P') IS NOT NULL
DROP PROC dbo.AddReviewRating
GO

CREATE PROC dbo.AddReviewRating

@ReviewID INT = NULL,
@CustomerID INT = NULL,
@PlusMinus CHAR(1) = NULL

AS

  DECLARE @error AS NVARCHAR(100);

  IF @ReviewID IS NULL OR @CustomerID IS NULL OR @PlusMinus IS NULL
  BEGIN
    SET @error = 'Error! Rating lacking information.';
    RAISERROR(@error, 16,1);
    RETURN;
  END

  INSERT INTO dbo.[Review Ratings] VALUES
  (@ReviewID, @CustomerID, @PlusMinus)


GO
```

Kod generujący procedurę *Add Review Rating*

5.4 New Mail

Procedura służąca do dodawania nowego maila do tabeli *Mail Queue*, będącej skrzynką nadawczą.

```
USE TestDB;
GO

DROP PROC IF EXISTS [dbo].[New Mail];
GO

CREATE PROC [dbo].[New Mail]
@CustomerID INT,
@Topic VARCHAR(50),
@Body VARCHAR(MAX)
AS
BEGIN

    IF @CustomerID IS NULL OR @Topic IS NULL OR @Body IS NULL
    BEGIN
        DECLARE @error NVARCHAR(50) = 'Error! Mail elements missing!';
        RAISERROR(@error, 16,1);
        RETURN;
    END

    INSERT INTO [dbo].[Mail Queue]
    ([RecipentID], [Topic], [Body])
    VALUES
    (@CustomerID, @Topic, @Body)

END

GO
```

Kod generujący procedurę *New Mail*

5.5 Confirm Account

Procedura służąca do potwierdzania konta użytkownika w systemie.

```
USE TestDB;
GO

DROP PROC IF EXISTS [dbo].[Confirm Account];
GO

CREATE PROC [dbo].[Confirm Account]
@CustomerID INT
AS
IF @CustomerID IS NULL
BEGIN
    DECLARE @error NVARCHAR(50) = 'Error! Account Information missing!';
    RAISERROR(@error, 16,1);
    RETURN;
END

UPDATE [Customers]
SET [Account Confirmed] = 1
WHERE [CustomerID] = @CustomerID

GO
```

Kod generujący procedurę *Confirm Account*

5.6 Add To Cart

Procedura służąca do dodawania przez użytkownika produktu do koszyka.

```
USE TestDB;
GO

DROP PROC IF EXISTS dbo.AddToCart;
GO


CREATE PROC dbo.AddToCart
@CustomerID INT = NULL,
@ProductID INT = NULL,
@Quantity INT = NULL



AS

DECLARE @error AS NVARCHAR(400);

IF @CustomerID IS NULL OR @ProductID IS NULL OR @Quantity IS NULL
BEGIN
     SET @error = 'Error!';
     RAISERROR(@error, 16,1);
     RETURN;
END

DECLARE @CartID AS INT;

SELECT @CartID =  CartID
FROM Carts
WHERE CustomerID = @CustomerID


DECLARE @RealQuantity AS INT

Select @RealQuantity = Stock FROM Storage WHERE ProductID = @ProductID;


IF @RealQuantity >= @Quantity
  BEGIN
  IF NOT EXISTS(Select * from [Cart Details] where CartID = @CartID and ProductID = @ProductID
    )
    BEGIN
    INSERT INTO [Cart details]
    VALUES (@CartID, @ProductId, @Quantity)
    END
  ELSE
    BEGIN
    UPDATE [Cart Details]
    SET Quantity = Quantity + @Quantity
    where CartID = @CartID and ProductID = @ProductID
    end



  UPDATE Storage
  SET Stock = @RealQuantity - @Quantity
  WHERE ProductID = @ProductID

  END

ELSE
BEGIN
     SET @error = 'Brak wystarczającej śiloć produktów w magazynie';
     RAISERROR(@error, 16,1);
     RETURN;
END

GO
```

Kod generujący procedurę *Add To Cart*

5.7 Delete From Cart

Procedura służąca do usuwania produktów z koszyka.

```
USE TestDB;
GO

DROP PROC IF EXISTS dbo.DeleteFromCart;
GO

CREATE PROC dbo.DeleteFromCart
@CustomerID INT = NULL,
@ProductID INT = NULL,
@Quantity INT = NULL

AS

DECLARE @error AS NVARCHAR(400);

IF @CustomerID IS NULL OR @ProductID IS NULL OR @Quantity IS NULL
BEGIN
     SET @error = 'Error!';
     RAISERROR(@error, 16,1);
     RETURN;
END

DECLARE @CartID AS INT;

SELECT @CartID =  CartID
FROM Carts
WHERE CustomerID = @CustomerID

IF NOT EXISTS (Select * From [cart details] where CartID = @CartID and ProductID = @ProductID)
BEGIN
     SET @error = 'Nie ma takiego produktu w koszyku!';
     RAISERROR(@error, 16,1);
     RETURN;
END

DECLARE @QuantityInCart AS INT


Select @QuantityInCart = Quantity FROM [cart details]
WHERE ProductID = @ProductID and CartID = @CartID;


IF @QuantityInCart >= @Quantity
    BEGIN
  IF @QuantityInCart > @Quantity
    BEGIN
    UPDATE [Cart Details]
    SET Quantity = Quantity - @Quantity
    WHERE ProductID = @ProductID and CartID = @CartID;
    end
  ELSE
    BEGIN
    DELETE FROM [Cart Details]
    WHERE ProductID = @ProductID and CartID = @CartID;
    END

  UPDATE Storage
  SET Stock = Stock + @Quantity
  WHERE ProductID = @ProductID
  END

ELSE
    BEGIN
    SET @error = 'Nie masz aż tylu produktów w koszyku!';
     RAISERROR(@error, 16,1);
     RETURN;
    END

GO
```

Kod generujący procedurę *Delete From Cart*

5.8 Buy

Procedura pozwalająca zamówić produkty będące w koszyku.

```
USE TestDB;
GO

DROP PROC IF EXISTS dbo.Buy;
GO

CREATE PROC dbo.Buy
@CustomerID INT = NULL

AS
DECLARE @EmployeeID INT
SELECT TOP 1 @EmployeeID = EmployeeID FROM Employees
WHERE DepartmentID = 4
ORDER BY NEWID()


DECLARE @CartID AS INT;

SELECT @CartID =  CartID
FROM Carts
WHERE CustomerID = @CustomerID


DECLARE @OrderID INT


INSERT INTO Orders
Values(@CustomerID, GETDATE(), @EmployeeID)
SET @OrderID = @@IDENTITY


DECLARE @ProductID INT
DECLARE @Quantity INT

DECLARE Cart CURSOR
FOR SELECT ProductID, Quantity FROM [Cart Details]
WHERE CartID = @CartID

DECLARE @UnitPrice MONEY


OPEN Cart
Fetch Cart INTO @ProductID, @Quantity

WHILE @@FETCH_STATUS = 0
BEGIN

    SELECT @UnitPrice = [Retail Price] FROM Products WHERE ProductID = @ProductID

    INSERT INTO [Order Details]
    VALUES(@OrderID, @ProductID, @UnitPrice, @Quantity)
    Fetch Cart INTO @ProductID, @Quantity

END
CLOSE Cart
DEALLOCATE Cart


DELETE FROM [Cart Details]
WHERE CartID = @CartID

DELETE FROM Carts
WHERE CartID = @CartID

INSERT INTO Carts(CustomerID)
VALUES (@CustomerID)

GO
```

Kod generujący procedurę *Buy*

Rozdział 6

Funkcje

6.1 Generate Note

Funkcja przyjmująca jako argument *@OrderID* zamówienia, tworząca fakturę na podstawie tego zamówienia.

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[GenerateNote];
GO

CREATE FUNCTION dbo.GenerateNote (@OrderID INT)
RETURNS @Note TABLE(
    OrderID INT,
    ClientName  NVARCHAR(40),
    Amout MONEY,
    OrderDate Date,
    [Number of Products] INT
)
AS
BEGIN

DECLARE @CustomerName NVARCHAR(40)

SELECT @CustomerName = [Name] from Customers JOIN Orders
ON Orders.CustomerID  = Customers.CustomerID
WHERE Orders.OrderID = @OrderID

DECLARE @Amount MONEY
SELECT @Amount = SUM(UnitPrice * Quantity) FROM [Order Details]
WHERE OrderID  = @OrderID

DECLARE @OrderDate Date
SELECT @OrderDate = [Order Date] from Orders
WHERE OrderID = @OrderID

DECLARE @Quantity INT
SELECT @Quantity = SUM(Quantity) FROM [Order Details]
WHERE OrderID = @OrderID


INSERT INTO @Note
VALUES(@OrderID, @CustomerName, @Amount, @OrderDate, @Quantity)

RETURN
END

GO
```

Kod generujący funkcję *Generate Note*

6.2 Get Reviews

Funkcja przyjmująca jako argument *@ProductID* produktu, zwracająca recenzje na jego temat.

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[Get Reviews];
GO

CREATE FUNCTION [dbo].[Get Reviews] (@ProductID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT [Content] AS [Review], [Rating]
    FROM [dbo].[Reviews]
    WHERE ProductID = @ProductID
)

GO
```

Kod generujący funkcję *Get Reviews*

6.3 Get Review Ratings

Funkcja przyjmująca jako argument *@ReviewID* recencji, zwracająca jej oceny.

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[Get Review Ratings];
GO

CREATE FUNCTION [dbo].[Get Review Ratings] (@ReviewID INT)
RETURNS  @Raitings TABLE(
ReviewID INT,
Pluses INT,
Minuses INT
)
AS
BEGIN
DECLARE @Pluses INT
SELECT @Pluses = COUNT(*) FROM [Review Ratings]
WHERE ReviewID = @ReviewID AND [Plus/Minus] = '+'

DECLARE @Minuses INT
SELECT @Minuses = COUNT(*) FROM [Review Ratings]
WHERE ReviewID = @ReviewID AND [Plus/Minus] = '-'

INSERT INTO @Raitings VALUES(@ReviewID, @Pluses, @Minuses)

RETURN
END

GO
```

Kod generujący funkcję *Get Review Ratings*

6.4 Avg. Product Rate

Funkcja przyjmująca jako argument *@ProductID* produktu, zwracająca jego średnią ocenę.

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[AvgProductRate];
GO

CREATE FUNCTION dbo.AvgProductRate (@ProductID INT)
RETURNS FLOAT
AS
BEGIN

DECLARE @AvgRate FLOAT


SELECT @AvgRate =   CAST(AVG(cast(rating as float)) AS DECIMAL(3,2))from Reviews
WHERE ProductID = @ProductID




RETURN @AvgRate
END

GO
```

Kod generujący funkcję *Avg. Product Rate*

6.5 Product Bought

Funkcja przyjmująca jako argument *@ProductName* nazwę produktu, zwracająca informację o ilości sprzedanych sztuk produktu.

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[Product Bought];
GO

CREATE FUNCTION dbo.[Product Bought] (@ProductName NVARCHAR(40) = '%')
RETURNS TABLE
AS
RETURN
(
  SELECT *
  FROM dbo.[Products Bought] AS P
  WHERE (P.[Product Name] LIKE @ProductName)
)

GO
```

Kod generujący funkcję *Product Bought*

6.6 Search

Funkcja służąca do precyzyjnego wyszukiwania produktu przyjmująca jako argumenty:

1. Nazwę Marki Produktu *@brandName*
2. Nazwę Kategorii Produktu *@CategoryName*
3. Nazwę Podkategorii Produktu *@Subcategory*
4. Dolny Zakres Cenowy *@MinPrice*
5. Górny Zakres Cenowy *@MaxPrice*

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[Search];
GO

CREATE FUNCTION dbo.Search (@BrandName NVARCHAR(20) = '%', @CategoryName NVARCHAR(20) ='%',
    @Subcategory NVARCHAR(40) ='%',
                @MinPrice INT = 0, @MaxPrice INT = 10000)
RETURNS TABLE
AS
RETURN
(
  SELECT P.[Product Name], P.[Product Description], B.[Brand Name], C.[Category Name], S.[
    Subcategory Name], P.[Retail Price] AS Price
  FROM Products AS P
  JOIN Brands AS B
  ON P.BrandID = B.BrandID
  JOIN Categories AS C
  ON P.CategoryID = C.CategoryID
  JOIN Subcategories AS S
  ON P.SubcategoryID = S.SubcategoryID
  WHERE ( [Brand Name] LIKE @BrandName AND
      [Category Name] LIKE @CategoryName AND
      [Subcategory Name] LIKE @Subcategory AND
      [Retail Price] >= @MinPrice AND
      [Retail Price] <=@MaxPrice
      )
)

GO

-- SELECT * FROM Search(DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT )
```

Kod generujący funkcję *Search*

6.7 Product Lookup

Funkcja służąca do wyszukiwania produktu na podstawie małej ilości informacji.
Przyjmuje argument *@ProductName* który może być prefiksem Nazwy Produktu, lub Marki.

```
USE TestDB;
GO

DROP FUNCTION IF EXISTS [dbo].[Product Lookup];
GO

CREATE FUNCTION dbo.[Product Lookup] (@ProductName NVARCHAR(40) = '%')
RETURNS TABLE
AS
RETURN
(
  SELECT Pr.[Product Name], Br.[Brand Name] AS [Brand], Ss.[Stock] AS [Availability]
  FROM dbo.[Storage Status] AS Ss
  JOIN dbo.[Products] AS Pr
    ON SS.[ProductID] = Pr.[ProductID]
  JOIN dbo.[Brands] AS Br
    ON Pr.[BrandID] = Br.[BrandID]
  WHERE Ss.[Product Name] LIKE @ProductName + '%'
    OR Br.[Brand Name] LIKE @ProductName + '%'
)

GO
```

Kod generujący funkcję *Product Lookup*

Rozdział 7

Wyzwalacze

7.1 Assign Brand ID

Wyzwalacz automatycznie dodający nowe Marki do Bazy Danych, oraz przypisujący je dodawanym produktom.

```
USE TestDB;
GO

DROP TRIGGER IF EXISTS [dbo].[TR_AssignBrandID];
GO

CREATE TRIGGER [dbo].[TR_AssignBrandID]
    ON [dbo].[New Products]
    INSTEAD OF INSERT
AS
BEGIN

    -- add new Brands to Database
    INSERT INTO [dbo].[Brands] ([Brand Name])
  SELECT DISTINCT I.[Brand Name]
  FROM INSERTED AS I
  LEFT JOIN [dbo].[Brands] AS B ON I.[Brand Name] = B.[Brand Name]
  WHERE B.[Brand Name] IS NULL

    -- add new Products with according linking to Brands
    INSERT INTO [dbo].[Products] ([Product Name], [BrandID]
        , [Product Description], [CategoryID], [SubcategoryID]
        , [Wholesale Price], [Retail Price])
    SELECT I.[Product Name], B.[BrandID], I.[Product Description], I.[CategoryID], I.[
    SubcategoryID], I.[Wholesale Price], I.[Retail Price]
    FROM INSERTED AS I
  JOIN [dbo].[Brands] AS B ON I.[Brand Name] = B.[Brand Name]

END

GO
```

Kod generujący wyzwalacz *Assign Brand ID*

7.2 Confirmation Email

Wyzwalacz automatycznie wysyłający nowym użytkownikom Maila z prośbą o potwierdzenia aktywacji konta.

```
USE TestDB;
GO

DROP TRIGGER IF EXISTS [dbo].[Confirmation Email];
GO

CREATE TRIGGER [dbo].[Confirmation Email]
ON [dbo].[Customers]
AFTER INSERT
AS
IF (@@ROWCOUNT = 0)
RETURN;

    DECLARE @TopicContent VARCHAR(50) = 'Witamy w Sklepie!';
    DECLARE @BodyContent VARCHAR(MAX) =
    'Jeszcze tylko jeden krok i możesz cieszyć ęsi zakupami!' + CHAR(13)
    + 'Wystarczy że potwierdzisz swój adres email.' + CHAR(13)
    + 'Pozdrawiamy,' + CHAR(13)
    + 'Ekipa Twojego Sklepu';

    DECLARE c CURSOR
    FOR SELECT [CustomerID] FROM INSERTED
    WHERE [Mailing List] = 1 AND [Account Confirmed] = 0

    DECLARE @cID INT

    OPEN c
    FETCH c INTO @cID

    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [dbo].[New Mail] @cID, @TopicContent, @BodyContent

        FETCH c INTO @cID
    END

    CLOSE c
    DEALLOCATE c


GO
```

Kod generujący wyzwalacz *Confirmation Email*

7.3 Guard Categories

Wyzwalacz powstrzymujący modyfikacje lub usunięcie kluczowych elementów Bazy Danych.

```
USE TestDB;
GO

DROP TRIGGER IF EXISTS [dbo].[Guard Categories];
GO

CREATE TRIGGER [dbo].[Guard Categories] ON [dbo].[Categories]
INSTEAD OF DELETE, UPDATE
AS
IF (@@ROWCOUNT = 0)
RETURN;
  RAISERROR ('Action prohibited. This incident will be reported.', 16, 0)
  ROLLBACK
GO
```

Kod generujący wyzwalacz *Guard Categories*

7.4 Stock Alert

Wyzwalacz zawiadamiający personel o wyczerpaniu zapasów w magazynie.

```
USE TestDB;
GO

DROP TRIGGER IF EXISTS [dbo].[Storage Alert];
GO

CREATE TRIGGER [dbo].[Storage Alert]
ON [dbo].[Storage]
AFTER UPDATE
AS

IF NOT EXISTS (
SELECT *
FROM [dbo].[Storage]
WHERE [Stock] = 0)
    RETURN;

SELECT FORMATMESSAGE('Alert! Some products are out of stock! Notify Logistics Dep.')
    AS [System Notification];

GO
```

Kod generujący wyzwalacz *Stock Alert*

7.5 New Product Notification

Wyzwalacz zawiadamiający subskrybujących użytkowników o nowych produktach dostępnych w sklepie.

```
USE TestDB;
GO

DROP TRIGGER IF EXISTS [dbo].[New Product Notification];
GO

CREATE TRIGGER [dbo].[New Product Notification]
ON [dbo].[Products]
AFTER INSERT
AS
IF (@@ROWCOUNT = 0)
RETURN;

    DECLARE @TopicContent VARCHAR(50) = 'Newsletter sklepu'
    DECLARE @BodyContent VARCHAR(MAX) = 'Nowe produkty ędostpne już ździ! Sprawdź!'

    DECLARE c CURSOR FOR
    SELECT [CustomerID] FROM [dbo].[Customers]
    WHERE [Mailing List] = 1 AND [Account Confirmed] = 1

    DECLARE @cID INT

    OPEN c
    FETCH c INTO @cID

    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [dbo].[New Mail] @cID, @TopicContent, @BodyContent

        FETCH c INTO @cID
    END

    CLOSE c
    DEALLOCATE c

GO
```

Kod generujący wyzwalacz *New Product Notification*

Rozdział 8

Skrypt tworzący bazę danych

Kod generujący bazę danych z oprogramowaniem:

```
-- create DB
PRINT 'Creating Database...'
:r <path>\DB-Project\CreateDB.sql
GO
PRINT 'Database created.
  '


PRINT 'Deploying Views...'
:r <path>\DB-Project\VIEW_Products_Bought.sql
:r <path>\DB-Project\VIEW_Storage_Status.sql
:r <path>\DB-Project\VIEW_Hot_Brands.sql
:r <path>\DB-Project\VIEW_Category_Organisation.sql
GO
PRINT 'Views deployed.
  '


PRINT 'Deploying Procedures...'
:r <path>\DB-Project\PROC_Add_Customer.sql
:r <path>\DB-Project\PROC_Add_Review.sql
:r <path>\DB-Project\PROC_Add_Review_Rating.sql
:r <path>\DB-Project\PROC_Add_To_Cart.sql
:r <path>\DB-Project\PROC_Buy.sql
:r <path>\DB-Project\PROC_Delete_From_Cart.sql
:r <path>\DB-Project\PROC_New_Mail.sql
:r <path>\DB-Project\PROC_Confirm_Account.sql
GO
PRINT 'Procedures deployed.
  '


PRINT 'Deploying Functions...'
:r <path>\DB-Project\FUNC_Generate_Note.sql
:r <path>\DB-Project\FUNC_Product_Bought.sql
:r <path>\DB-Project\FUNC_Product_Lookup.sql
:r <path>\DB-Project\FUNC_Get_Review_Ratings.sql
:r <path>\DB-Project\FUNC_Get_Reviews.sql
GO
PRINT 'Functions deployed.
  '


PRINT 'Deploying Triggers...'
:r <path>\DB-Project\TRIG_AssignBrandID.sql
:r <path>\DB-Project\TRIG_Guard_Categories.sql
:r <path>\DB-Project\TRIG_New_Product_Notification.sql
:r <path>\DB-Project\TRIG_Confirmation_Email.sql
:r <path>\DB-Project\TRIG_Stock_Alert.sql
GO
PRINT 'Triggers deployed.
  '


PRINT 'Filling Tables...'
```

```
:r <path>\DB-Project\FillDB.sql
GO
PRINT '
Tables Filled.
  '
GO
```

Kod generujący bazę danych z oprogramowaniem

Kod generujący tabele:

```
/* CREATE DATABASE */
USE master;
GO

IF DB_ID('TestDB') IS NOT NULL
    DROP DATABASE TestDB;
GO

CREATE DATABASE TestDB;
GO

USE TestDB;
GO

/* /CREATE DATABASE */
-----------------------------------------------------------------
/* CREATE TABLES */

-- -- /* FIRST DROP TABLES*/
PRINT 'Performing cleanup...'
GO

DROP TABLE IF EXISTS [dbo].[Brands] ;
GO

DROP TABLE IF EXISTS [dbo].[Cart Details] ;
GO

DROP TABLE IF EXISTS [dbo].[Carts] ;
GO

DROP TABLE IF EXISTS [dbo].[Customers] ;
GO

DROP TABLE IF EXISTS [dbo].[Categories] ;
GO

DROP TABLE IF EXISTS [dbo].[Subcategories] ;
GO

DROP TABLE IF EXISTS [dbo].[Products] ;
GO

DROP TABLE IF EXISTS [dbo].[Suppliers] ;
GO

DROP TABLE IF EXISTS [dbo].[Resupplies] ;
GO

DROP TABLE IF EXISTS [dbo].[Resupply Details] ;
GO

DROP TABLE IF EXISTS [dbo].[Orders] ;
GO

DROP TABLE IF EXISTS dbo.[Order Details] ;
GO

DROP TABLE IF EXISTS [dbo].[Reviews] ;
GO

DROP TABLE IF EXISTS [dbo].[Review Ratings] ;
GO
```

```
64
65  DROP TABLE IF EXISTS [dbo].[Departments] ;
66  GO
67
68  DROP TABLE IF EXISTS [dbo].[Employees] ;
69  GO
70
71  DROP TABLE IF EXISTS [dbo].[Storage] ;
72  GO
73
74  DROP TABLE IF EXISTS [dbo].[New Products] ;
75  GO
76
77  DROP TABLE IF EXISTS [dbo].[Mail Queue] ;
78  GO
79
80
81
82  PRINT 'Done.
83      '
84  GO
85  -- -- /* /FIRST DROP TABLES*/
86
87  PRINT 'Creating tables...'
88  GO
89
90  -- Customers
91  CREATE TABLE [dbo].[Customers]
92      (
93          [CustomerID] INT IDENTITY(1,1) NOT NULL
94          , CONSTRAINT [PK_Customers] PRIMARY KEY (CustomerID)
95          , [Name] NVARCHAR(40) NOT NULL
96          , [Email Addres] NVARCHAR(40) NOT NULL UNIQUE
97          , [Phone Number] NVARCHAR(15)
98          , [Addres] NVARCHAR(40)
99          , [Postal Code] NVARCHAR(8)
100         , [City] NVARCHAR(25)
101         , [Mailing List] BIT NOT NULL DEFAULT 1
102         , [Account Confirmed] BIT NOT NULL DEFAULT 0
103     )
104 ;
105 GO
106
107 -- Mail Queue
108 CREATE TABLE [dbo].[Mail Queue]
109     (
110         [MailID] INT IDENTITY(1,1) NOT NULL
111         , CONSTRAINT [PK_Mail_Queue] PRIMARY KEY (MailID)
112         , [RecipentID] INT NOT NULL
113         , CONSTRAINT [FK_Mail_Customer] FOREIGN KEY (RecipentID)
114           REFERENCES [dbo].[Customers] (CustomerID)
115           ON DELETE CASCADE
116           ON UPDATE CASCADE
117         , [Topic] VARCHAR(50) NOT NULL
118         , [Body] VARCHAR(MAX) NOT NULL
119     )
120 ;
121 GO
122
123 -- Categories
124 CREATE TABLE [dbo].[Categories]
125     (
126         [CategoryID] TINYINT NOT NULL
127         , CONSTRAINT [PK_Categories] PRIMARY KEY (CategoryID)
128         , [Category Name] NVARCHAR(20) NOT NULL
129     )
130 ;
131 GO
132
133 -- Subcategories
134 CREATE TABLE [dbo].[Subcategories]
135     (
136         [CategoryID] TINYINT NOT NULL
137         , CONSTRAINT [FK_Subcategories_Categories] FOREIGN KEY (CategoryID)
138           REFERENCES [dbo].[Categories] (CategoryID)
139           ON DELETE CASCADE
```

```
          ON UPDATE CASCADE
        , [SubcategoryID] TINYINT IDENTITY(1,1) NOT NULL UNIQUE
        , [Subcategory Name] NVARCHAR(40) NOT NULL
        , CONSTRAINT [PK_Subcategories] PRIMARY KEY (CategoryID, SubcategoryID)
    )
;
GO

-- Brands
CREATE TABLE [dbo].[Brands]
    (
        [BrandID] INT IDENTITY(1,1)
        , CONSTRAINT [PK_Brands] PRIMARY KEY (BrandID)
        , [Brand Name] NVARCHAR(20) NOT NULL
    )
;
GO



-- Products
CREATE TABLE [dbo].[Products]
    (
        [ProductID] INT IDENTITY(1,1) NOT NULL
        , CONSTRAINT [PK_Products] PRIMARY KEY (ProductID)
        , [Product Name] NVARCHAR(40) NOT NULL
        , [BrandID] INT NOT NULL
        , [Product Description] NVARCHAR(100)
        , [CategoryID] TINYINT
        , CONSTRAINT [FK_Products_Categories] FOREIGN KEY (CategoryID)
          REFERENCES [dbo].[Categories] (CategoryID)
          ON DELETE SET NULL
          ON UPDATE CASCADE
        , [SubcategoryID] TINYINT
        , CONSTRAINT [FK_Products_Subcategories] FOREIGN KEY (SubcategoryID)
          REFERENCES [dbo].[Subcategories] (SubcategoryID)
        , [Wholesale Price] MONEY NOT NULL  -- TODO trigger: price >=0
        , [Retail Price] MONEY NOT NULL  -- TODO trigger
    )
;
GO

-- temporary table for adding new products
CREATE TABLE [dbo].[New Products]
(
    [Brand Name] NVARCHAR(20)
    , [Product Name] NVARCHAR(40) NOT NULL
    , [Product Description] NVARCHAR(100)
    , [CategoryID] TINYINT
    , [SubcategoryID] TINYINT
    , [Wholesale Price] MONEY NOT NULL
    , [Retail Price] MONEY NOT NULL
)
;
GO


-- Suppliers
CREATE TABLE [dbo].[Suppliers]
    (
        [SupplierID] INT IDENTITY(1,1) NOT NULL
        , CONSTRAINT [PK_Suppliers] PRIMARY KEY (SupplierID)
        , [SupplierName] NVARCHAR(40) NOT NULL
        , [Country] NVARCHAR(20)
        , [City] NVARCHAR(20)
        , [Address] NVARCHAR(40)
        , [PostalCode] NVARCHAR(6)
        , [EmailAddress] NVARCHAR(30)
        , [Phone] NVARCHAR(20)
    )
;
GO

-- Resupplies
CREATE TABLE [dbo].[Resupplies]
    (
```

```
        [ResupplyID] INT IDENTITY(1,1) NOT NULL
        , CONSTRAINT [PK_Resupplies] PRIMARY KEY (ResupplyID)
        , [SupplierID] INT NOT NULL
        , CONSTRAINT [FK_Resupplies_Suppliers] FOREIGN KEY (SupplierID)
          REFERENCEs [dbo].[Suppliers] (SupplierID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
    )
;
GO

-- ResupplyDetails
CREATE TABLE [dbo].[Resupply Details]
    (
        [ResupplyID] INT NOT NULL
        , CONSTRAINT [FK_ResupplyDetails_Resupplies] FOREIGN KEY (ResupplyID)
          REFERENCEs [dbo].[Resupplies] (ResupplyID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [ProductID] INT NOT NULL
        , CONSTRAINT [FK_ResupplyDetails_Products] FOREIGN KEY (ProductID)
          REFERENCEs [dbo].[Products] (ProductID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [Quantity] SMALLINT NOT NULL
    )
;
GO

-- Departments
-- Employee can be assigned to a department
CREATE TABLE [dbo].[Departments]
    (
        [DepartmentID] TINYINT NOT NULL
        , CONSTRAINT [PK_Departments] PRIMARY KEY (DepartmentID)
        , [DepartmentName] VARCHAR(20) NOT NULL
    )
;
GO

-- Employees
CREATE TABLE [dbo].[Employees]
    (
        [EmployeeID] INT IDENTITY(1,1) NOT NULL
        , CONSTRAINT [PK_Employees] PRIMARY KEY (EmployeeID)
        , [Name] NVARCHAR(40) NOT NULL UNIQUE
        , [DepartmentID] TINYINT DEFAULT NULL
        , CONSTRAINT [FK_Employees_Departments] FOREIGN KEY (DepartmentID)
          REFERENCES [dbo].[Departments] (DepartmentID)
          ON DELETE SET NULL
          ON UPDATE CASCADE
    )
;
GO

-- Orders
CREATE TABLE [dbo].[Orders]
    (
        [OrderID] INT IDENTITY(1,1) NOT NULL
        , CONSTRAINT [PK_Orders] PRIMARY KEY (OrderID)
        , [CustomerID] INT
        , CONSTRAINT [FK_Orders_Customers] FOREIGN KEY (CustomerID)
          REFERENCES [dbo].[Customers] (CustomerID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [Order Date] DATETIME
        , [EmployeeID] INT
        , CONSTRAINT [FK_Orders_Employees] FOREIGN KEY (EmployeeID)
          REFERENCES [dbo].[Employees] (EmployeeID)
          ON DELETE CASCADE
          ON UPDATE CASCADE

    )
;
GO

```

```
-- Order Details
CREATE TABLE [dbo].[Order Details]
    (
        [OrderID] INT NOT NULL
        , CONSTRAINT [FK_OrderDetails_Orders] FOREIGN KEY (OrderID)
          REFERENCES [dbo].[Orders] (OrderID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [ProductID] INT NOT NULL
        , CONSTRAINT [FK_OrderDetails_Products] FOREIGN KEY (ProductID)
          REFERENCES [dbo].[Products] (ProductID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [UnitPrice] MONEY
        , [Quantity] SMALLINT NOT NULL
        , CONSTRAINT [PK_OrderDetails] PRIMARY KEY (OrderID, ProductID)
    )
;
GO

-- Reviews
CREATE TABLE [dbo].[Reviews]
    (
        [ReviewID] INT IDENTITY(1,1) NOT NULL UNIQUE
        , [CustomerID] INT NOT NULL
        , CONSTRAINT [FK_Reviews_Customers] FOREIGN KEY (CustomerID)
          REFERENCES [dbo].[Customers] (CustomerID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [ProductID] INT NOT NULL
        , CONSTRAINT [FK_Reviews_Products] FOREIGN KEY (ProductID)
          REFERENCES [dbo].[Products] (ProductID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [Content] NTEXT
        , [Rating] TINYINT NOT NULL
        , CONSTRAINT [CK_Reviews_ValidRating] CHECK
          (Rating >= 1 AND Rating <= 5)
        , CONSTRAINT [PK_Reviews] PRIMARY KEY (CustomerID, ProductID)
    )
;
GO

-- Review Ratings
-- customer can 'thumbUp' a review to mark it as helpful
CREATE TABLE [dbo].[Review Ratings]
    (
        [ReviewID] INT NOT NULL
        , CONSTRAINT [FK_ReviewRatings_Reviews] FOREIGN KEY (ReviewID)
          REFERENCES [dbo].[Reviews] (ReviewID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [CustomerID] INT NOT NULL
        , CONSTRAINT [FK_ReviewRatings_Customers] FOREIGN KEY (CustomerID)
          REFERENCES [dbo].[Customers] (CustomerID),
          [Plus/Minus] CHAR(1)
    )
;
GO


-- Storage
-- every item has to be stored in Storage
CREATE TABLE [dbo].[Storage]
    (
        -- every product has only one spot in storage
        [ProductID] INT NOT NULL UNIQUE
        , CONSTRAINT [FK_Storage_Products] FOREIGN KEY (ProductID)
          REFERENCES [dbo].[Products] (ProductID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [StockMax] INT NOT NULL
        , CONSTRAINT [CK_Storage_StockMaxValid] CHECK ( StockMax > 0 )  -- TODO trigger
        , [Stock] INT NOT NULL
        , CONSTRAINT [CK_Storage_StockValid] CHECK ( Stock BETWEEN 0 AND StockMax ) -- TODO
    trigger
```

```
    )

;
GO

-- Carts
-- every user, while shopping, adds items to Cart.
-- Orders are made by ordering what is currently in a Cart
CREATE TABLE [dbo].[Carts]
    (
        [CartID] INT IDENTITY(1,1)
        , CONSTRAINT [PK_Carts] PRIMARY KEY (CartID)
        -- every customer can have only one cart at the time
        , [CustomerID] INT NOT NULL UNIQUE
        , CONSTRAINT [FK_Carts_Customers] FOREIGN KEY (CustomerID)
          REFERENCES [dbo].[Customers] (CustomerID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
    )
;
GO

-- Cart Details
CREATE TABLE [dbo].[Cart Details]
    (
        [CartID] INT NOT NULL
        , CONSTRAINT [FK_CartDetails_Carts] FOREIGN KEY (CartID)
          REFERENCES [dbo].[Carts] (CartID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , [ProductID] INT NOT NULL
        , CONSTRAINT [FK_CartDetails_Products] FOREIGN KEY (ProductID)
          REFERENCES [dbo].[Products] (ProductID)
          ON DELETE CASCADE
          ON UPDATE CASCADE
        , CONSTRAINT [PK_CartDetails] PRIMARY KEY (CartID, ProductID)
        , [Quantity] INT NOT NULL
    )
;
GO

PRINT 'Done.
    '
GO
/* /CREATE TABLES */
-------------------------------------------------------------------------
```

Kod generujący tabele

Kod wypełniający tabele:

```
/* FILL TABLES */

SET NOCOUNT ON

PRINT 'Adding Customers...'
    EXEC dbo.AddCustomer @Name = 'Celina Johansen', @Email = 'celinajohansen@gmail.com',
    @PhoneNumber = '123456789',
@Addres = 'Skarbowa 16', @PostalCode =  '30-056', @City = 'Krakow'
GO

EXEC dbo.AddCustomer @Name = 'Cyryl Jensen', @Email = 'cJensen@gmail.com', @PhoneNumber = '
    222333444',
@Addres = 'Lipowa 18/2', @PostalCode = '12-819', @City = 'Warszawa'
GO

EXEC dbo.AddCustomer @Name = 'Cameron Moller', @Email = 'mollercameron@wp.pl', @PhoneNumber =
     '367235980',
@Addres = 'Lea 13', @PostalCode = '26-600', @City = 'Radom'
GO

PRINT 'Done.'
GO

```

```
21
22 PRINT 'Confirming some Customers...'
23 EXEC dbo.[Confirm Account] 1
24 PRINT 'Done.'
25 GO
26
27
28
29
30    -- INSERT INTO [dbo].[Customers] --name, addresmail, phone, addres, postal code, city
31    -- VALUES
32      -- ('Celina Johansen', 'celinajohansen@gmail.com', '123456789', 'Skarbowa 16', '30-056',
     'Krakow')
33     --, ('Cyryl Jensen', 'cJensen@gmail.com', '222333444', 'Lipowa 18/2', '12-819', 'Warszawa'
      )
34     --, ('Cameron Moller','mollercameron@wp.pl', '367235980', 'Lea 13', '26-600', 'Radom' )
35     --, ('Carolina Moller', 'carolinaMoller@gmail.pl', '379215396', 'Szwedzka 22', '40-432', '
     Poznan' )
36     --, ('Cyryl Andersen', 'CyrylAndersen@gmail.com', '92349089', 'Kapelanka 166', '30-252', '
     Krakow')
37     --, ('Cataleya Pedersen', 'Cataleya@gmail.com', '23433444', 'Miodowa 44', '12-800', '
     Warszawa' )
38     --, ('Chanell Moller', 'MollerChannel@gmail.com', '23412454', 'Kwiatowa 19', '12-600', '
     Warszawa' )
39     --, ('Carmen Moller', 'CarmenMoller@gmail.com', '19896764', 'Polna 1', '12-880', 'Warszawa
     ' )
40     --, ('Carla Christiansen', 'carlachris@gmail.com', '334226557', 'Czysta 112', '55-220', '
     Gdansk')
41     --, ('Cyprian Moller', 'cyprianMoll@wp.pl', '346747292', 'Kijowska 99','40-222', 'Poznan')
42    -- ;
43     --
44
45 PRINT 'Adding Categories...'
46 INSERT INTO [dbo].[Categories] VALUES
47   (1, 'Twarz')
48 , (2, 'Usta')
49 , (3, 'Oczy')
50 , (4, 'Brwi')
51 , (5, 'Paznokcie')
52 , (6, 'Akcesoria')
53 , (7, 'Dermokosmetyki')
54 , (8, 'Zestawy')
55 ;
56 PRINT 'Done.
57 '
58 GO
59
60 PRINT 'Adding Subcategories...'
61 INSERT INTO [dbo].[Subcategories] VALUES
62 -- Twarz
63   (1, 'łPodkady')
64 , (1, 'Pudry')
65 , (1, 'Korektory')
66 , (1, 'śRozwietlacze i bronzery')
67 , (1, 'Róże')
68 , (1, 'Kremy BB i CC')
69 , (1, 'Bazy pod makijaż')
70 , (1, 'Spraye utrwalające makijaż')
71 , (1, 'ęPdzle do makijażu')
72 , (1, 'Akcesoria')
73 -- Usta
74 , (2, 'łByszczyki')
75 , (2, 'Pomadki do ust')
76 , (2, 'Konturówki do ust')
77 , (2, 'ęPdzle do ust')
78 -- Oczy
79 , (3, 'Tusze do ęrzs')
80 , (3, 'Bazy i cienie do powiek')
81 , (3, 'Paletki cieni')
82 , (3, 'Kredki do oczu i eyelinery')
83 , (3, 'Sztuczne ęrzsy')
84 , (3, 'Odżywki do ęrzs')
85 , (3, 'ęPdzle do oczu')
86 , (3, 'Akcesoria')
87 -- Brwi
88 , (4, 'Kredki do brwi')
```

```
, (4, 'Cienie')
, (4, 'Korektory do brwi')
, (4, 'Maskary, żele, woski, pomady, henna')
, (4, 'ęPdzle do brwi')
, (4, 'Akcesoria')
-- Paznokcie
, (5, 'Lakiery do paznokci')
, (5, 'Lakiery hybrydowe')
, (5, 'ęPielgnacja')
, (5, 'Zmywacze')
, (5, 'Zdobienie paznokci')
, (5, 'Bazy i utwardzanie, top coat')
, (5, 'Sztuczne paznokcie')
, (5, 'Akcesoria')
-- Akcesoria
, (6, 'Do twarzy')
, (6, 'Do ust')
, (6, 'Do oczu')
, (6, 'Do paznokci')
, (6, 'ęPdzle do makijażu')
, (6, 'Kosmetyczki')
-- Dermokosmetyki nie mają podkategorii
-- Zestawy też nie mają podkategorii
;
PRINT 'Done.
'
GO

PRINT 'Adding Brands...'
INSERT INTO [dbo].[Brands] VALUES
  ('Hello Kitty') -- test
PRINT 'Done.
'
GO

PRINT 'Adding Products...'
INSERT INTO [dbo].[New Products] VALUES
  ('Revlon', 'Colorstay', 'łpodkad z pompką do cery łtustej i mieszanej, 30 ml', 1, 1, 24,
    48.99)
, ('Gosh', 'X-CEPTIONAL WEAR', 'kryjący łpodkad do twarzy w kremie do twarzy, 35 ml', 1, 1, 21,
     41.99)
, ('Anabelle Minerals', 'GOLDEN FAIREST', 'łpodkad matujący do twarzy, 4 g', 1, 1, 22, 44.99)
, ('Catrice','HD LIQUID COVERAGE', 'łpodkad do twarzy, 30 ml', 1, 1, 10, 23.19)
;
PRINT 'Done.
'
GO

PRINT 'Adding Departments...'
INSERT INTO [dbo].[Departments] VALUES
  (1, 'Management')
, (2, 'IT')
, (3, 'Sales')
, (4, 'Packaging')
;
PRINT 'Done.
'
GO

PRINT 'Adding Employees...'
INSERT INTO [dbo].[Employees] VALUES
  ('Eryk Sorensen', 4)
, ('Elif Jensen', 3)
, ('Eliza Moller', 2)
, ('Elisabeth Christiansen', 1)
, ('Edward Larsen', 1)
, ('Emanuel Poulsen', 3)
, ('Eugeniusz Johansen', 3)
, ('Eryk Pedersen', 4)
, ('Elif Olsen', 4)
, ('Elzna Jorgensen', 4)
;
PRINT 'Done.
'
GO

```

```
PRINT 'Adding Storage...'
INSERT INTO [dbo].[Storage] VALUES
  (1, 1000, 1000)
, (2, 1000, 1000)
, (3, 1000, 1000)
;
PRINT 'Done.
'
GO


PRINT 'Adding some products to carts'

EXEC dbo.AddToCart @CustomerID = 1, @ProductID = 1, @Quantity = 2
GO

EXEC dbo.AddToCart @CustomerID = 1, @ProductID = 2, @Quantity = 3
GO

EXEC dbo.AddToCart @CustomerID = 3, @ProductID = 2, @Quantity = 5
GO

EXEC dbo.AddToCart @CustomerID = 2, @ProductID = 2, @Quantity = 1
GO

EXEC dbo.AddToCart @CustomerID = 1, @ProductID = 3, @Quantity = 7
GO

EXEC dbo.AddToCart @CustomerID = 2, @ProductID = 1, @Quantity = 3
GO

EXEC dbo.AddToCart @CustomerID = 3, @ProductID = 1, @Quantity = 1
GO

EXEC dbo.AddToCart @CustomerID = 3, @ProductID = 3, @Quantity = 100
GO

PRINT 'Done.'


PRINT 'Buying products...'
EXEC dbo.Buy @CustomerID = 1
GO
EXEC dbo.Buy @CustomerID = 2
GO
/*EXEC dbo.Buy @CustomerID = 3
GO*/

PRINT 'Done.'

PRINT 'Adding Reviews ...'
EXEC dbo.AddReview @CustomerID = 2, @ProductID = 1, @Content = 'Super, Polecam!', @Rating = 5
GO
EXEC dbo.AddReview @CustomerID = 3, @ProductID = 2, @Content = 'Może być.', @Rating = 3
GO
EXEC dbo.AddReview @CustomerID = 3, @ProductID = 1, @Content = 'Nawet, nawet', @Rating = 3
GO

PRINT 'Done.'

PRINT 'Adding Reviews Ratings ...'
EXEC dbo.AddReviewRating @CustomerID = 1, @ReviewID = 1, @PlusMinus = '+'
GO
EXEC dbo.AddReviewRating @CustomerID = 3, @ReviewID = 1, @PlusMinus = '-'

PRINT 'Done.'

SET NOCOUNT OFF


GO
/* /FILL TABLES */
-----------------------------------------------------------------------
```

Kod wypełniający tabele

Rozdział 9

Typowe zapytania

```
-- proste wyszukiwanie produktu przez użytkownika na podstawie ęśczci nazwy
EXEC [dbo].[Product Lookup] 'Oriflame'
EXEC [dbo].[Product Lookup] 'Orif'
EXEC [dbo].[Product Lookup] 'Av'
EXEC [dbo].[Product Lookup] 'Revlon'


-- łdokadniejsze wyszukiwanie na podstawie śścilejszych parametrów
EXEC [dbo].[Search](DEFAULT, 'Paznokcie', DEFAULT, DEFAULT, 20.00)
EXEC [dbo].[Search](DEFAULT, 'Maseczki', 'Peelingi', 20.00, 40.00)


-- sprawdzenie popularnych w sklepie
SELECT * FROM [dbo].[Hot Brands]


-- dodawanie do koszyka
EXEC [dbo].[Add To Cart] 1 1 1
EXEC [dbo].[Add To Cart] 1 2 2
EXEC [dbo].[Add To Cart] 1 3 4


-- zakup
EXEC [dbo].[Buy] 1
EXEC [dbo].[Buy] 14


-- generowanie faktury dla danego zamówienia
EXEC [dbo].[Generate Note] 1
EXEC [dbo].[Generate Note] 3
EXEC [dbo].[Generate Note] 4


-- pobranie przez ęaplikacj webową recenzji na temat produktu oraz gwiazdek dla recenzji do
    zrenderowania strony
SELECT * FROM [dbo].[Get Reviews] 1
SELECT * FROM [dbo].[Get Review Ratings] 1

SELECT * FROM [dbo].[Get Reviews] 2
SELECT * FROM [dbo].[Get Review Ratings] 4

SELECT * FROM [dbo].[Get Reviews] 3
SELECT * FROM [dbo].[Get Review Ratings] 2

SELECT * FROM [dbo].[Get Reviews] 5
SELECT * FROM [dbo].[Get Review Ratings] 3


-- sprawdzanie przez pracowników stanu magazynu
SELECT * FROM [dbo].[Storage Status]
SELECT * FROM [dbo].[Storage Status] WHERE STOCK LIKE '0%'
```

Przykładowe zapytania