25.01.2020r

Rafał Adamski
Michał Wypchło

Raport - Bazy Danych II

Projekt Sklep Internetowy

I. Charakterystyka projektu:

• Opis:

Za główny cel projektu obraliśmy stworzenie aplikacji bazodanowej do obsługi Sklepu. Aplikacja ma za zadanie dostarczenie takich funkcjonalności jak zakładanie konta, logowanie, zakup produktów. Baza danych przechowuje informacje o użytkownikach, produktach oraz zamówieniach. Użytkownik otrzymuje dostęp do zestawu funkcji realizowanych przez aplikację(GUI) napisaną w języku C#, która z kolei korzysta operacji dostępnych w bazie danych takich jak transakcje, procedury, triggery czy też widoki.

• Wykorzystane technologie:

- **.NET Framework** - platforma programistyczna opracowana przez Microsoft, obejmująca środowisko uruchomieniowe oraz biblioteki klas dostarczające standardowej funkcjonalności dla aplikacji. Technologia ta nie jest związana z żadnym konkretnym językiem programowania, a programy mogą być pisane w wielu językach – w naszym projekcie jest to C#. Zadaniem platformy .NET Framework jest zarządzanie różnymi elementami systemu: kodem aplikacji, pamięcią i zabezpieczeniami.

- **Visual Studio 2019** – ze względu na przyjazne środowisko programistyczne. Oferuje ono także łatwe tworzenie aplikacji okienkowych w języku C#, który wykorzystaliśmy w projekcie.

- **Microsoft SQL** - System Zarządzania Bazą Danych dostarczany przez firmę Microsoft.Dostarcza zaawansowane środowisko zarządzania bazami danych przeznaczone do profesjonalnych zastosowań. Jest to platforma, z którą mieliśmy najwięcej do czynienia podczas nauki oraz w ramach naszej kariery zawodowej, dlatego też zdecydowaliśmy się ją wybrać.

• Architektura oprogramowania

Oprogramowanie działa w architekturze klient-serwer.

Baza jest stworzona z myślą o przechowywaniu danych o użytkownikach, produktach i zamówieniach. Zawarte są w niej również procedury, triggery i widoki umożliwiające bardziej przystępne korzystanie z niej.

Aplikacja kliencka pozwala użytkownikowi na interakcję z bazą danych. Jest mostem do tej bazy i umożliwia łatwe wykonywanie operacji na bazie, czyli oferuje typowe usługi jak każdy sklep internetowy.

II. Baza danych:

• Skrypty tworzące strukturę SZBD

```
CREATE TABLE [dbo].[Address](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[ZipCode] [nchar](12) NULL,
	[City] [nchar](100) NULL,
	[Street] [nchar](100) NULL,
CONSTRAINT [PK_Address] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[Categories](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[CategoryName] [nvarchar](max) NULL,
CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

CREATE TABLE [dbo].[OrderList](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[OrderID] [int] NOT NULL,
	[ProductID] [int] NOT NULL,
	[Amount] [int] NOT NULL,
CONSTRAINT [PK_OrderList] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[Orders](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[OrderStatus] [int] NOT NULL,
	[OrderTotal] [decimal](18, 2) NULL,
	[OrderDate] [datetime2](7) NULL,
	[UserID] [int] NOT NULL,
CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[OrdersStatus](
	[ID] [int] IDENTITY(1,1) NOT NULL,
```

```
	[Status] [nchar](10) NOT NULL,
CONSTRAINT [PK_OrderStates] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[ProdCat](
	[IDProd] [int] NOT NULL,
	[IDCat] [int] NOT NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[Products](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[ProductName] [nvarchar](max) NOT NULL,
	[ProductDesc] [nvarchar](max) NULL,
	[ProductPrice] [decimal](18, 2) NOT NULL,
	[ProductStock] [float] NOT NULL,
	[ProductReleaseDate] [date] NULL,
	[VendorID] [int] NOT NULL,
	[ProductImage] [image] NULL,
CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

CREATE TABLE [dbo].[Roles](
	[ID] [int] NOT NULL,
	[Name] [nchar](50) NOT NULL,
	[Description] [nchar](250) NULL,
PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[UserRole](
	[UserID] [int] NOT NULL,
	[RoleID] [int] NOT NULL,
CONSTRAINT [PK_UserRole] PRIMARY KEY CLUSTERED
(
	[UserID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[Users](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Username] [nchar](50) NOT NULL,
	[Password] [nchar](50) NOT NULL,
	[Email] [nchar](50) NOT NULL,
	[FirstName] [nchar](50) NULL,
	[LastName] [nchar](50) NULL,
	[TelephoneNumber] [int] NULL,
	[AddressID] [int] NULL,
CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
CREATE TABLE [dbo].[Vendors](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Vendor] [nchar](20) NULL,
CONSTRAINT [PK_Vendors] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[OrderList] WITH CHECK ADD CONSTRAINT [FK_OrderList_Orders]
FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([ID])
GO
ALTER TABLE [dbo].[OrderList] CHECK CONSTRAINT [FK_OrderList_Orders]
GO
ALTER TABLE [dbo].[OrderList] WITH CHECK ADD CONSTRAINT [FK_OrderList_Products]
FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ID])
GO
ALTER TABLE [dbo].[OrderList] CHECK CONSTRAINT [FK_OrderList_Products]
GO
ALTER TABLE [dbo].[Orders] WITH CHECK ADD CONSTRAINT [FK_Orders_OrdersStatus]
FOREIGN KEY([OrderStatus])
REFERENCES [dbo].[OrdersStatus] ([ID])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_OrdersStatus]
GO
ALTER TABLE [dbo].[Orders] WITH CHECK ADD CONSTRAINT [FK_Orders_Users] FOREIGN
KEY([UserID])
REFERENCES [dbo].[Users] ([ID])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Users]
GO
ALTER TABLE [dbo].[ProdCat] WITH NOCHECK ADD CONSTRAINT [FK_ProdCat_Categories]
FOREIGN KEY([IDCat])
REFERENCES [dbo].[Categories] ([ID])
GO
ALTER TABLE [dbo].[ProdCat] CHECK CONSTRAINT [FK_ProdCat_Categories]
GO
ALTER TABLE [dbo].[ProdCat] WITH NOCHECK ADD CONSTRAINT [ProdToProd] FOREIGN
KEY([IDProd])
REFERENCES [dbo].[Products] ([ID])
GO
ALTER TABLE [dbo].[ProdCat] CHECK CONSTRAINT [ProdToProd]
GO
ALTER TABLE [dbo].[Products] WITH CHECK ADD CONSTRAINT [FK_Products_Vendors]
FOREIGN KEY([VendorID])
REFERENCES [dbo].[Vendors] ([ID])
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_Vendors]
GO
ALTER TABLE [dbo].[UserRole] WITH CHECK ADD CONSTRAINT [FK_UserRole_Roles] FOREIGN
KEY([RoleID])
REFERENCES [dbo].[Roles] ([ID])
GO
ALTER TABLE [dbo].[UserRole] CHECK CONSTRAINT [FK_UserRole_Roles]
GO
ALTER TABLE [dbo].[UserRole] WITH CHECK ADD CONSTRAINT [FK_UserRole_Users] FOREIGN
KEY([UserID])
REFERENCES [dbo].[Users] ([ID])
GO
ALTER TABLE [dbo].[UserRole] CHECK CONSTRAINT [FK_UserRole_Users]
GO
ALTER TABLE [dbo].[Users] WITH CHECK ADD CONSTRAINT [FK_User_Address] FOREIGN
KEY([AddressID])
```

```
REFERENCES [dbo].[Address] ([Id])
GO
ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [FK_User_Address]
GO
```

• **Diagramy ER (najciekawsze rozwiązania)**

Zmiana statusu zamówienia

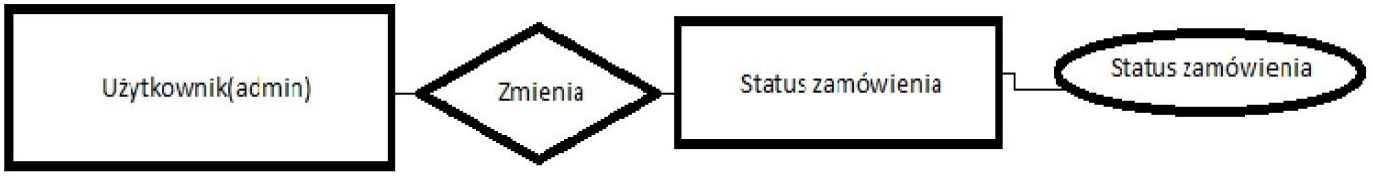


Diagram 1

Składanie zamówienia

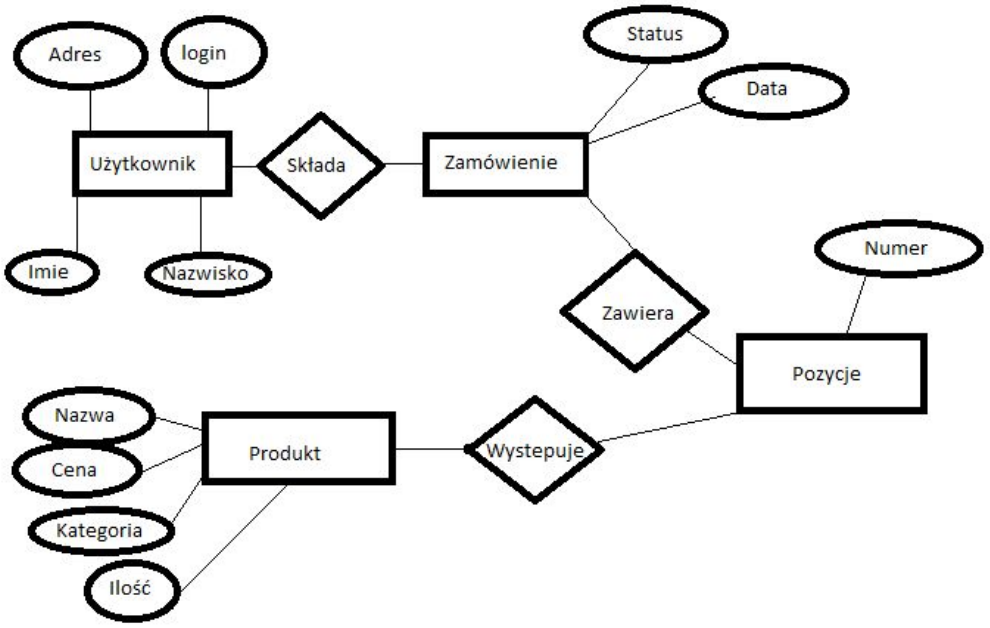


Diagram 2

Związek między produktem, a producentem:

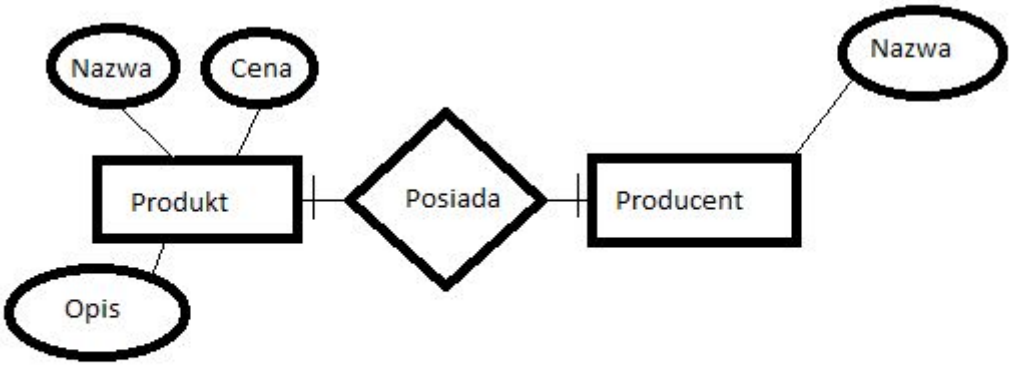
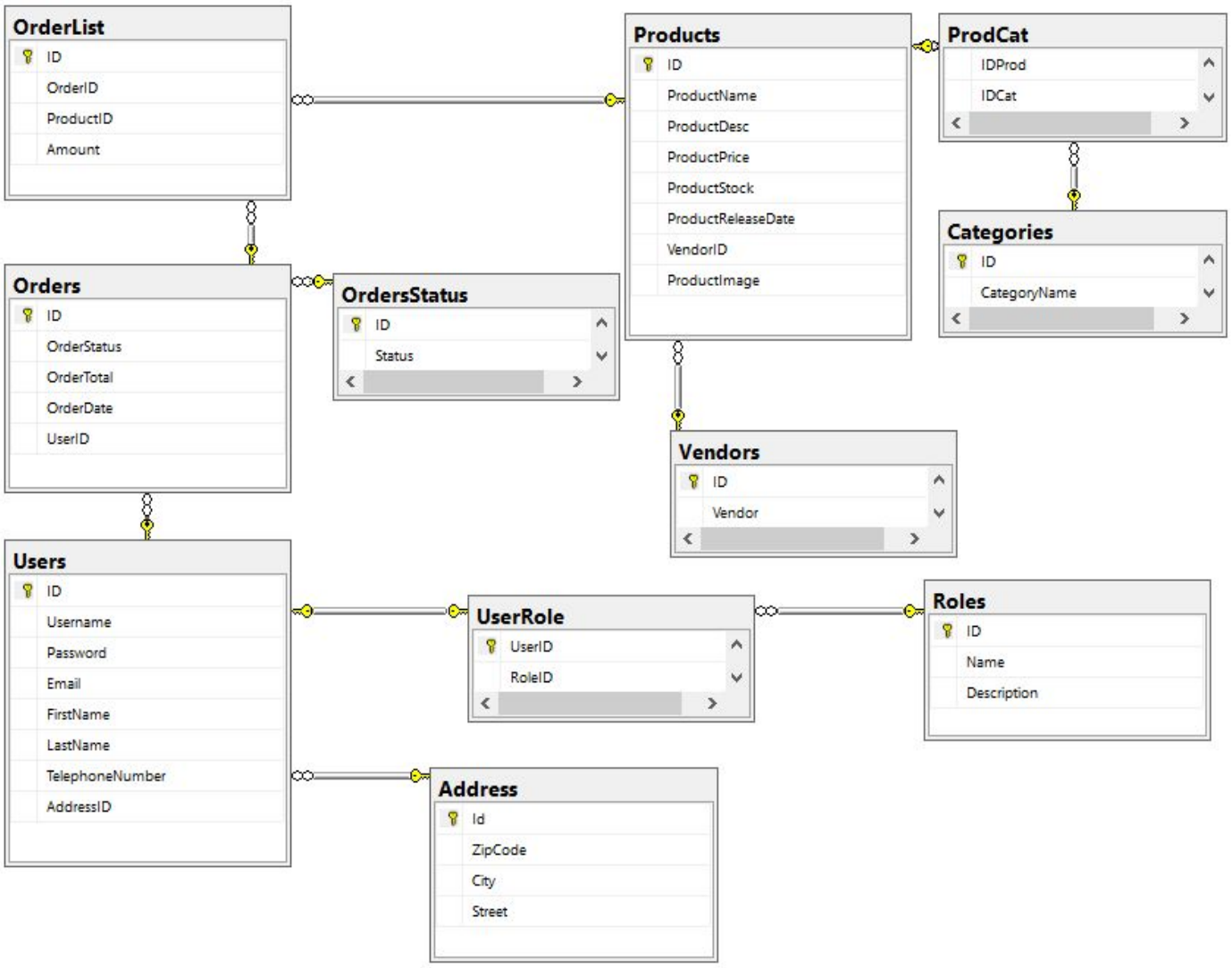


Diagram 3

- **Diagram BD**



III. Przykłady rozwiązań wykorzystujące:

• **Triggery**

Trigger dodający automatycznie datę do nowo dodawanego produktu:
Data dodania posłużyć może nam do wyświetlania listy Nowości spośród produktów dostępnych w sklepie.

```
CREATE TRIGGER ReleaseDate
ON Products
AFTER INSERT
AS
BEGIN
UPDATE Products SET ProductReleaseDate = GETDATE() where ProductReleaseDate is null
END
```

Trigger przypisujący rolę “brak” do nowo zarejestrowanego użytkownika:
Użytkownicy dzielą się na adminów, managerów oraz zwykłych klientów - trigger dodaje wpis do tabeli UserRole dla nowo utworzonego użytkownika(przypisuje mu rolę zwykłego użytkownika, specjalne uprawnienia może nadać tylko główny administrator).

```
CREATE TRIGGER [RoleTrigger]
ON [dbo].[Users]
FOR INSERT
AS
BEGIN
SET NOCOUNT ON
INSERT INTO UserRole(UserID,RoleID) Values((SELECT TOP 1 ID FROM Users ORDER BY ID
DESC),3)
END
```

• **Widoki**

Widok zwracający ID trzech ostatnio dodanych produktów

```
SELECT TOP (3) ID
FROM dbo.Products
ORDER BY ProductReleaseDate DESC
```

Widok zwracający ID najpopularniejszego produktu spośród ostatnich 5 zakupów
Innymi słowami najczęściej kupowany produkt z ostatnich pięciu zamówień

```
CREATE VIEW [dbo].[Top1OutOfRecent5]
AS SELECT TOP (1) IDprod, SUM(Amount) AS [Suma Zamowionych]
FROM
(SELECT TOP (5) dbo.OrderList.ID, dbo.Products.ID as IDprod, dbo.OrderList.Amount
FROM dbo.OrderList INNER JOIN
dbo.Orders ON dbo.OrderList.OrderID = dbo.Orders.ID INNER JOIN
dbo.Products ON dbo.OrderList.ProductID = dbo.Products.ID
ORDER BY dbo.OrderList.ID DESC)
AS tabela
GROUP BY IDprod
ORDER BY [Suma Zamowionych] DESC
```

• **Procedury**

Procedura dodająca nowego użytkownika do tabeli Users, gdy ten poprawnie przejdzie przez proces rejestracji.

```
CREATE PROCEDURE [dbo].[DodajUzytkownika](
@login nchar(50),
@password nchar(50),
@email nchar(50),
@imie nchar(50),
@nazwisko nchar(50),
@telefon int
)
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO dbo.Users (Username, Password, FirstName, LastName, Email,
TelephoneNumber) VALUES(@login, @password, @imie, @nazwisko, @email, @telefon)
END
```

Procedura sprawdzająca, czy login wprowadzony w trakcie rejestracji użytkownika jest zajęty.

```
CREATE PROCEDURE [dbo].[LoginTaken](@login nchar(50))
AS
BEGIN
SET NOCOUNT ON;
SELECT Username from dbo.Users WHERE Username=@login
END
GO
```

Procedura wyciągająca dane z bazy produktu o konkretnym ID

```
CREATE PROCEDURE [dbo].[DisplaySpecificProduct]
	@id int
AS
	SELECT
dbo.Products.ProductName, dbo.Products.ProductDesc, dbo.Products.ProductPrice,
dbo.Products.ProductImage, dbo.Products.ProductStock, dbo.Vendors.Vendor,
dbo.Categories.CategoryName, dbo.Products.ID
FROM
dbo.Categories INNER JOIN
dbo.ProdCat ON dbo.Categories.ID = dbo.ProdCat.IDCat INNER JOIN
dbo.Products ON dbo.ProdCat.IDProd = dbo.Products.ID INNER JOIN
dbo.Vendors ON dbo.Products.VendorID = dbo.Vendors.ID
WHERE
(
dbo.Products.ID = @id
)
```

Procedura logowania - sprawdza poprawność danych w bazie:

```
CREATE PROCEDURE [dbo].[LoginProcedure](@login nchar(50),@password nchar(50))
AS
BEGIN
	SET NOCOUNT ON;
SELECT Username, Password from dbo.Users WHERE Username=@login AND
Password=@password
END
```

Procedura ładująca dostępne dane o użytkowniku do formularza “Edytuj dane”:

```
CREATE PROCEDURE [dbo].[GetUserData](@login nchar(50))
AS
BEGIN
	SET NOCOUNT ON;
SELECT [FirstName], [LastName], [ZipCode], [City], [Street], [TelephoneNumber] FROM
Users WHERE Username=@login
END
```

Procedura zmieniająca hasło do konta użytkownika:

```
CREATE PROCEDURE [dbo].[PasswordChange](@login nchar(50), @password nchar(50))
AS
BEGIN
	SET NOCOUNT ON;
UPDATE Users SET Password=@password WHERE Username=@login
END
```

Procedura weryfikująca poprawność hasła przed zmianą hasła:

```
CREATE PROCEDURE [dbo].[VerifyPassword](@login nchar(50))
AS
BEGIN
SET NOCOUNT ON;
SELECT Password from dbo.Users WHERE Username=@login
END
```

Procedura odpowiadająca za wyszukanie przedmiotu w sklepie

```
CREATE PROCEDURE [dbo].[Search]
	@id_kat int ,
	@id_ven int,
	@prod_name nvarchar(50) = ''

AS

DECLARE @querry nvarchar(max) = 'SELECT dbo.Products.ProductName, dbo.Products.ID
FROM dbo.Products INNER JOIN
dbo.Vendors ON dbo.Products.VendorID = dbo.Vendors.ID INNER JOIN
dbo.ProdCat ON dbo.Products.ID = dbo.ProdCat.IDProd
                                          WHERE Products.ProductName LIKE ' +
char(39)+ char(37) + @prod_name + char(37)+ char(39);

IF @id_kat IS NOT NULL
SET @querry += N' AND ProdCat.IDCat = ' + CONVERT(VARCHAR(12), @id_kat);

IF @id_ven IS NOT NULL
SET @querry += N' AND Products.VendorID = ' + CONVERT(VARCHAR(12), @id_ven);

SET @querry += N' GROUP BY Products.ProductName, Products.ID';

EXEC (@querry)
```

Procedura dodająca zamówienie do bazy

```
 CREATE PROCEDURE [dbo].[DodajZamowienie](
@userID int
)
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO
dbo.Orders (OrderStatus, userID, OrderTotal)
VALUES
(1, @userID, 0)
END
```

Procedura dodająca szczegóły do zamówienia

```
CREATE PROCEDURE [dbo].[DodajSzczegolyZamowienia](
@user_id int,
@prod_id int,
@amount int
)
AS
BEGIN
SET NOCOUNT ON;

DECLARE @order_ID int
SELECT TOP 1
@order_ID = ID
FROM
Orders
WHERE Orders.UserID = @user_id
ORDER BY ID DESC

DECLARE @prod_price decimal(18,2)
SELECT
@prod_price = ProductPrice
FROM
Products
WHERE Products.ID = @prod_id

INSERT INTO OrderList (OrderID, ProductID, Amount)
VALUES (@order_ID, @prod_id, @amount)

UPDATE Orders
SET OrderTotal = OrderTotal + (@prod_price * @amount)
WHERE ID = @order_ID

exec UsunZMagazynu @prod_id, @amount

END
```

(Procedura dodająca zamówienie i dodająca szczegóły do niej są wywoływane zaraz po sobie)

Procedura wypisująca wszystkie zamówienia użytkownika

```
CREATE PROCEDURE [dbo].[ListUserOrders]
```

```
	@userID int
AS
	SELECT TOP (100) PERCENT
dbo.Orders.ID, dbo.Orders.OrderTotal, dbo.OrdersStatus.Status, dbo.Orders.OrderDate
FROM
dbo.Orders INNER JOIN
dbo.OrderList ON dbo.Orders.ID = dbo.OrderList.OrderID INNER JOIN
dbo.OrdersStatus ON dbo.Orders.OrderStatus = dbo.OrdersStatus.ID INNER JOIN
dbo.Products ON dbo.OrderList.ProductID = dbo.Products.ID INNER JOIN
dbo.Users ON dbo.Orders.UserID = dbo.Users.ID
WHERE
(dbo.Users.ID = @userID)
GROUP BY
dbo.Orders.ID, dbo.Orders.OrderTotal, dbo.OrdersStatus.Status, dbo.Orders.OrderDate
ORDER BY
dbo.Orders.OrderDate DESC
```

Procedura wypisująca dane zamówienie

```
CREATE PROCEDURE [dbo].[DisplayOrder]
	@orderID int
AS
SELECT
dbo.OrderList.OrderID, dbo.OrderList.Amount, dbo.Products.ProductName,
dbo.Products.ProductPrice
FROM
dbo.OrderList INNER JOIN
dbo.Products ON dbo.OrderList.ProductID = dbo.Products.ID
WHERE
(dbo.OrderList.OrderID = @orderID)
```

- **Transakcje**

Zdejmująca towar z magazynu w momencie zakupu

```
CREATE PROCEDURE [dbo].[UsunZMagazynu]
@id int,
@amount int
AS
BEGIN TRANSACTION
UPDATE
[dbo].[Products]
SET
[dbo].[Products].ProductStock = [dbo].[Products].ProductStock - @amount
WHERE
([dbo].[Products].ID = @id)
COMMIT TRANSACTION
```

Dodająca nowy produkt w momencie, gdy administrator chce dodać produkt do systemu:

```
CREATE PROCEDURE [dbo].[AddNewProductTransaction](
@nazwa nchar(50),
@opis nchar(250),
@cena decimal,
@ilosc float,
@kategoria int,
@vendor INT
)
AS
BEGIN TRANSACTION
INSERT INTO Products(ProductName, ProductDesc, ProductPrice, ProductStock,
ProductCategoryID, VendorID) Values(@nazwa, @opis,@cena,@ilosc,@kategoria,@vendor)
            INSERT INTO ProdCat(IDProd, IDCat) VALUES((SELECT
IDENT_CURRENT('Products')), @kategoria)
COMMIT
```

Usuwająca konto użytkownika wraz z wpisem o jego uprawnieniach:

```
CREATE PROCEDURE [dbo].[DeleteAccount](@login nchar(50))
AS
BEGIN Transaction
DELETE from dbo.UserRole WHERE UserID=(SELECT ID FROM dbo.Users WHERE
Username=@login)
DELETE from dbo.Users WHERE Username=@login
COMMIT TRANSACTION
```

IV. Ciekawe rozwiązania

Jednym z najciekawszych rozwiązań w naszej aplikacji jest połączenie trigerra, który wpisuje aktualną datę dla dodawanego produktu wraz z widokiem nowości dodanych do sklepu, jest to proste rozwiązanie, gdzie na podstawie daty produktu SELECTem, a dokładnie procedurą opartą na SELECTcie wybieramy najnowsze produkty dodane do bazy.

Innym rozwiązaniem było zastosowanie tabeli “przejściowej” w bazie danych, tak byśmy byli w stanie utworzyć relację wiele do wielu. Niestety w Microsoft SQL Server nie da się stworzyć takiej relacji bezpośrednio i dlatego potrzebny był taki właśnie chwyt.

W jednej z naszych procedur (wyszukiwanie towaru) został użyty tzw. *Dynamic SQL*. Jest to technika pozwalająca na na dynamiczne dopasowanie kwerendy. W tym przypadku od podanych danych zależy jak będziemy filtrować wyszukiwany produkt. Pozwoliło to na uogólnienie procedury, przez co nie musieliśmy tworzyć kilku z nich.