Baza Danych Strony Społecznościowej

Kamil Brzozowski,
Grzegorz Brzozowski

Spis Treści

1 Założenia i Opis Projektu

Baza Danych powstała w celu przechowywania informacji dotyczących społeczności, grup, wydarzeń na portalu społecznościowym.

Głównymi założeniami projektu były informacje o:

- **Osobach, ich danych logowania oraz danych osobowych**
- **Znajomych czyli połączenie dwoch osób jakąś relacją**
- **Postach, gdzie posty można umieszczać na swoim publicznym profilu lub w grupie**
- **Grupach które zrzeszają jakąś społeczność interesującą się jakimś tematem (kategorią)**
- **Wydarzeniach oraz miejscu ich wystąpienia aby przy ich pomocy dotrzeć do większej audiencji oraz ponownie zrzeszać jakąś społeczność**

2 Skrypt Tworzący Bazę Danych

```
CREATE TABLE Adresy (
    Id INT PRIMARY KEY,
    Województwo VARCHAR(19) NOT NULL,
    Miasto VARCHAR(30) NOT NULL,
    Ulica VARCHAR(30) NOT NULL,
    Kod_Pocztowy VARCHAR(6) NOT NULL,
)

CREATE TABLE Konta (
    Id INT NOT NULL PRIMARY KEY IDENTITY (1,1),
    Login VARCHAR(20) NOT NULL UNIQUE CHECK (LEN(Login) > 5 AND LEN(Login) < 21),
    Hasło VARCHAR(255) NOT NULL,
    Email VARCHAR(30) NOT NULL UNIQUE CHECK (EMAIL LIKE '%_@__%.__%'),
    Liczba_Znajomych INT DEFAULT 0,
);

CREATE TABLE Dane_Osobowe (
    Pesel VARCHAR (11) CHECK (LEN(Pesel) = 11) PRIMARY KEY,
    Imię VARCHAR(50),
    Nazwisko VARCHAR(50),
    Nr_Telefonu VARCHAR(9) CHECK (LEN(Nr_Telefonu) = 9),
    Urodziny DATETIME,
    Id_Adresu INT FOREIGN KEY REFERENCES Adresy(Id),
    Id_Konta INT FOREIGN KEY REFERENCES Konta(Id)
)
```

Tabela konta posiada informacje potrzebne między innymi do logowania. Każde konto może (ale nie musi) mieć dane osobowe oraz adres.

```
CREATE TABLE Zdjęcia (
    Id INT NOT NULL PRIMARY KEY IDENTITY (1,1),
```

```
    Ścieżka VARCHAR(255) NOT NULL,
    Id_Konta INT NOT NULL REFERENCES Konta(Id),
    Podpis VARCHAR(30),
    Wymiary VARCHAR(11) NOT NULL CHECK (Wymiary LIKE '%x%')
)

CREATE TABLE Zdjęcia_Profilowe (
    Id_Konta INT NOT NULL FOREIGN KEY REFERENCES Konta(Id) PRIMARY KEY,
    Id_Zdjęcia INT NOT NULL REFERENCES Zdjęcia(Id)
)
```

Następnie tworzymy tabelki na zdjęcia użytkowników dodawane do kont (zdjęcie dodane przez użytkownika może być później użyte do np. wydarzeń) oraz tabelkę trzymającą aktualne zdjęcie profilowe każdego użytkownika które może zostać wybrane sposród wszystkich wrzuconych przez niego zdjęć.

```
CREATE TABLE Znajomi (
    Id1 INT NOT NULL REFERENCES Konta(Id),
    Id2 INT NOT NULL REFERENCES Konta(Id),
    PRIMARY KEY(Id1, Id2)
)

CREATE TABLE Kategorie (
    Nazwa VARCHAR(50) NOT NULL PRIMARY KEY
)
```

Kolejne tabelki to znajomi oraz kategorie. Znajomi posiadają id dwóch kont którzy są ze sobą znajomymi, oba id tworzą klucz główny tabeli. Tabela kategorie trzyma nazwy różnych kategorii wykorzystywanych później do grup i wydarzeń.

```
CREATE TABLE Grupy (
    Id INT NOT NULL PRIMARY KEY IDENTITY (1,1),
    Id_Założyciela INT NOT NULL FOREIGN KEY REFERENCES Konta(Id),
    Nazwa VARCHAR(50) NOT NULL UNIQUE,
    Opis VARCHAR(255)
)

CREATE TABLE Moderatorzy_Grup (
    Id_Grupy INT NOT NULL FOREIGN KEY REFERENCES Grupy(Id),
    Id_Moderatora INT NOT NULL FOREIGN KEY REFERENCES Konta(Id),
    Uprawnienia INT NOT NULL CHECK (Uprawnienia >= 1 AND Uprawnienia <= 3)
    PRIMARY KEY(Id_Grupy, Id_Moderatora)
)

CREATE TABLE Grupy_Kategorie (
    Id_Grupy INT NOT NULL REFERENCES Grupy(Id),
    Nazwa_Kategorii VARCHAR(50) NOT NULL REFERENCES Kategorie(Nazwa),
    PRIMARY KEY(Id_Grupy, Nazwa_Kategorii)
)

CREATE TABLE Grupy_Członkowie (
    Id_Grupy INT NOT NULL REFERENCES Grupy(Id),
    Id_Konta INT NOT NULL REFERENCES Konta(Id),
    Najlepszy_Poster BIT NOT NULL DEFAULT 0
    PRIMARY KEY(Id_Grupy, Id_Konta)
)
```

W następnej kolejności tworzymy tabelki związane z grupami. Relacje w grupach wymagały utworzenia nowych tabelek - może być kilku moderatorów grup z których każdy ma uprawnienia od 1 do 3, każda grupa może mieć kilka kategorii oraz każda grupa może mieć wielu członków.

```
CREATE TABLE Posty (
    Id INT NOT NULL PRIMARY KEY IDENTITY (1,1),
    Treść VARCHAR(MAX) NOT NULL CHECK (LEN(Treść) > 5),
    Id_Autora INT NOT NULL REFERENCES Konta(Id),
    Id_Grupy INT REFERENCES Grupy(Id),
    Ilość_Polubień INT NOT NULL DEFAULT 0,
    Data_dodania DATETIME NOT NULL
)

CREATE TABLE Komentarze (
    Id INT NOT NULL PRIMARY KEY IDENTITY (1,1),
    Treść VARCHAR(MAX) NOT NULL CHECK (LEN(Treść) > 1),
    Id_Postu INT NOT NULL REFERENCES Posty(Id),
    Id_Autora INT NOT NULL REFERENCES Konta(Id),
    Ilość_Polubień INT NOT NULL DEFAULT 0,
    Data_dodania DATETIME NOT NULL,
)

CREATE TABLE Posty_Archiwum (
    Id INT NOT NULL PRIMARY KEY IDENTITY (1,1),
    Treść VARCHAR(MAX) NOT NULL CHECK (LEN(Treść) > 5),
    Id_Autora INT NOT NULL FOREIGN KEY REFERENCES Konta(Id),
    Id_Grupy INT REFERENCES Grupy(Id),
    Ilość_Polubień INT NOT NULL,
    Data_Dodania DATETIME NOT NULL,
    Data_Zmiany DATETIME NOT NULL,
    Stan VARCHAR(9) NOT NULL CHECK (Stan IN ('edycja', 'usunięcie'))
)
```

Tabelki dotyczą postów wrzucanych przez ludzi. Post może zostać wrzucony na grupę (wtedy posiada Id Grupy) lub może zostać udostępniony publicznie. Każdy post może zostać skomentowany, co zapamiętuje odpowiednia tabela Komentarze połączona kluczem obcym z Postami. Każdy post może zostać usunięty lub zedytowany, w takim wypadku zostaje dodany do Posty Archiwum i w każdej chwili może zostać przywrócony (jednak po usunięciu posta komentarze znikają bezpowrotnie).

```
CREATE TABLE Wiadomości (
    Id_Odbiorca INT NOT NULL FOREIGN KEY REFERENCES Konta(Id),
    Id_Nadawca INT NOT NULL FOREIGN KEY REFERENCES Konta(Id),
    Treść VARCHAR(MAX) NOT NULL,
    Data_Wysłania DATETIME NOT NULL,
)
```

Tabelka na wiadomości pomiędzy użytkownikami

```
CREATE TABLE Wydarzenia (
    Id INT PRIMARY KEY IDENTITY(1, 1),
    Id_Założyciela INT FOREIGN KEY REFERENCES Konta(Id),
```

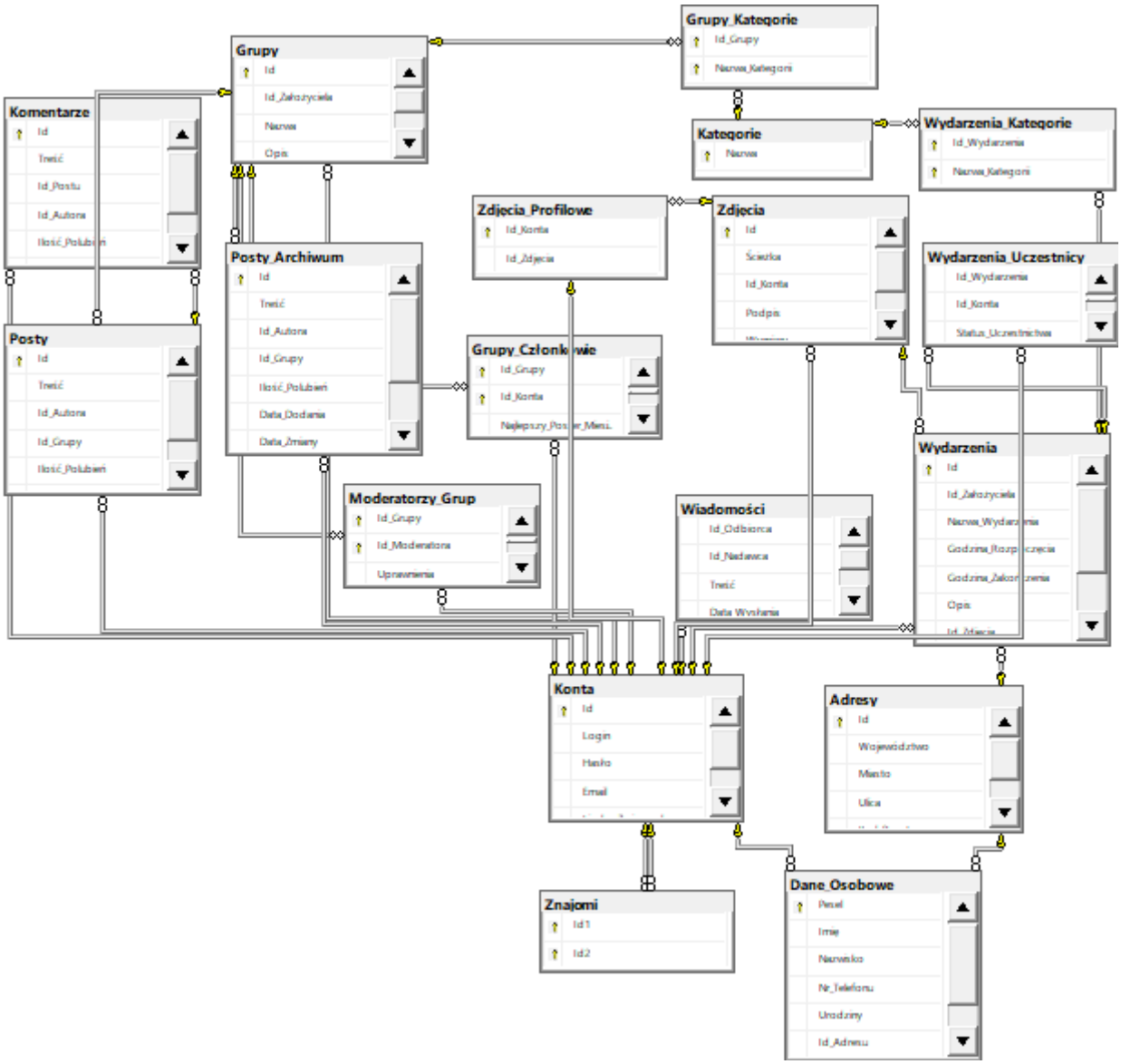
```
    Nazwa_Wydarzenia VARCHAR(100) NOT NULL CHECK(LEN(Nazwa_Wydarzenia) > 9),
    Godzina_Rozpoczęcia DATETIME NOT NULL,
    Godzina_Zakończenia DATETIME NOT NULL,
    Opis VARCHAR(MAX) NOT NULL CHECK(LEN(Opis) > 20),
    Id_Zdjęcia INT FOREIGN KEY REFERENCES Zdjęcia(Id),
    Id_Adresu INT FOREIGN KEY REFERENCES Adresy(Id)
)

CREATE TABLE Wydarzenia_Kategorie (
    Id_Wydarzenia INT NOT NULL FOREIGN KEY REFERENCES Wydarzenia(Id),
    Nazwa_Kategorii VARCHAR(50) NOT NULL FOREIGN KEY REFERENCES Kategorie(Nazwa),
    PRIMARY KEY(Id_Wydarzenia, Nazwa_Kategorii)
)


CREATE TABLE Wydarzenia_Uczestnicy (
    Id_Wydarzenia INT NOT NULL FOREIGN KEY REFERENCES Wydarzenia(Id),
    Id_Konta INT NOT NULL FOREIGN KEY REFERENCES Konta(Id),
    Status_Uczestnictwa VARCHAR(16) NOT NULL CHECK (Status_Uczestnictwa IN ('uczestniczy'
    , 'odmówił', 'nie odpowiedział', 'zainteresowany')) DEFAULT 'nie odpowiedział'
)
)
```

Ostanimi tabelkami są tabele związane z wydarzeniami. Zawierają one informacje o godzinach w których się odbywa, o miejscu czy zdjęciach związanych z wydarzeniem. Tak jak w przypadku grup, konieczne było dodanie nowych tabel - każde wydarzenie może mieć kilka kategorii oraz każde może mieć wielu uczestników.

3 Schemat Relacyjnej Bazy Danych



4 Opis Stworzonych Widoków

```
CREATE OR ALTER VIEW IlośćKomentarzyPost AS
SELECT P.Id, P.Treść, P.Ilość_Polubień, ISNULL(A.IlośćKomentarzy,0) AS IlośćKomentarzy
FROM Posty P
LEFT JOIN (
    SELECT K.Id_Postu, COUNT(*) IlośćKomentarzy
    FROM Komentarze K
    GROUP BY K.Id_Postu
) A ON A.Id_Postu = P.Id
GO
```

Widok IlośćKomentarzyPost wyświetla każdy post z tabeli Posty wraz z dodatkową kolumną z informacją o ilości komentarzy pod tym postem. Widok ten będzie wykorzystywany do wyświetlania postu, gdzie każdy post będzie zawierał taką właśnie informację.

```
CREATE OR ALTER VIEW TrwającеWydarzeniaSzczegóły AS
SELECT W.Nazwa_Wydarzenia, W.Opis, Z.Ścieżka, A.Miasto, A.Ulica
FROM Wydarzenia W
LEFT JOIN Zdjęcia Z ON Z.Id = W.Id_Zdjęcia
LEFT JOIN Adresy A ON A.Id = W.Id_Adresu
WHERE Godzina_Rozpoczęcia <= GETDATE() AND Godzina_Zakończenia >= GETDATE()
GO
```

Widok TrwającеWydarzeniaSzczegóły zawiera dokładnie informacje o wydarzeniach które się rozpoczęły oraz jeszcze trwają. Bierze on z tabeli zdjęcia oraz adresy wszystkie zdjęcia oraz adres mowiący o miejscu gdzie wydarzenie sie odbywa. Będzie on wykorzystywany w celu informowania użytkownika o tym gdzie i jak wydarzenie przebiega.

```
CREATE OR ALTER VIEW ŚredniaWiekuWGrupach AS
SELECT G.Nazwa, G.Opis, AV.ŚredniWiek
FROM Grupy G
INNER JOIN (
    SELECT GC.Id_Grupy, AVG(YEAR(GETDATE()) - YEAR(DO.Urodziny)) AS ŚredniWiek
    FROM Grupy_Członkowie GC
    INNER JOIN Dane_Osobowe DO ON DO.Id_Konta = GC.Id_Konta
    GROUP BY Id_Grupy
) AV ON AV.Id_Grupy = G.Id
GO
```

Widok ŚredniaWiekuWGrupach wylicza średnią wieku w grupie i razem z innymi danymi grupy wyświetla tą wartość. Użytkownik wyświetlając informację o wszystkich grupach będzie mógł porównać swój wiek ze średnim z każdej i dobrać odpowiednią dla siebie grupę.

```
CREATE OR ALTER VIEW ŚredniaWiekuWydarzenia AS
SELECT W.Nazwa_Wydarzenia, W.Opis, W.Godzina_Rozpoczęcia, W.Godzina_Zakończenia, AD.ŚredniWiek
FROM Wydarzenia W
INNER JOIN (
    SELECT WU.Id_Wydarzenia, AVG(YEAR(GETDATE()) - YEAR(DO.Urodziny)) AS ŚredniWiek
    FROM Wydarzenia_Uczestnicy WU
    LEFT JOIN Dane_Osobowe DO ON WU.Id_Konta =  DO.Id_Konta
    WHERE WU.Status_Uczestnictwa = 'uczestniczy'
    GROUP BY WU.Id_Wydarzenia
) AD ON AD.Id_Wydarzenia = W.Id
GO
```

Tak jak w powyższym widoku, zwracamy średnią wieku, tylko tym razem dla wydarzeń. Dzięki temu użytkownik będzie w stanie wybrać pasujące dla siebie wydarzenie z listy wszystkich wydarzeń.

```
CREATE OR ALTER VIEW IlośćGrupWKategorii AS
SELECT Nazwa_Kategorii, COUNT(*) AS Ilość_Grup
FROM Grupy_Kategorie
GROUP BY Nazwa_Kategorii
GO
```

Widok pokazuje ilość grup w każdej kategorii. Może być używany przez administratorów strony do porównywania i sprawdzania które kategorie są najbardziej popularne i żądane przez użytkowników.

```
CREATE OR ALTER VIEW IlośćOsóbPodJednymAdresem
AS
SELECT DO.Id_Adresu, a.Województwo, a.Miasto, a.Ulica, a.Kod_Pocztowy,
COUNT(*) OVER(PARTITION BY DO.Id_Adresu) AS Ilość_Osób
FROM Dane_Osobowe DO
INNER JOIN Adresy a ON a.Id = DO.Id_Adresu
GO
```

Dla każdego adresu zwracamy ilość osób mieszkających pod nim. Może być wykorzystany w celach np. reklamowych - im więcej osób mieszka pod danym adresem, tym lepiej wysłać pod niego reklamę, gdyż będzie miała większy odbiór.

5 Opis Stworzonych Funkcji

```
CREATE OR ALTER FUNCTION PostyZGrupyPoDacie (@Id_Grupy INT, @Data DATETIME)
RETURNS TABLE
AS
RETURN (
    SELECT P.Id, P.Treść, P.Ilość_Polubień, P.Data_dodania
    FROM Posty P
    WHERE P.Id_Grupy = @Id_Grupy AND P.Data_dodania >= @Data
)
GO
```

Bardzo prosta funkcja która ma za zadanie wyświetlać wszystkie posty z danej grupy od jakiejś daty. Może też zostać wykorzystana do pokazania wszystkich postów z grupy co może bardzo ułatwić projektowanie strony.

```
CREATE OR ALTER FUNCTION StatusyUczestnictwaWydarzenia (@Id_Wydarzenia INT)
RETURNS @WydarzeniaStatusy TABLE
(
    Id INT,
    Nazwa_Wydarzenia VARCHAR(100),
    Opis VARCHAR(MAX),
    Uczestnicy INT,
    Ludzie_którzy_odmówili INT,
    Brak_Odpowiedzi INT,
    Zainteresowani INT
)
AS
BEGIN

    DECLARE @Uczestnicy INT
    SET @Uczestnicy = (
        SELECT COUNT(*)
        FROM Wydarzenia_Uczestnicy
        WHERE Status_Uczestnictwa = 'uczestniczy' AND Id_Wydarzenia = @Id_Wydarzenia
    )

    DECLARE @Ludzie_którzy_odmówili INT
    SET @Ludzie_którzy_odmówili = (
        SELECT COUNT(*)
        FROM Wydarzenia_Uczestnicy
        WHERE Status_Uczestnictwa = 'odmówił' AND Id_Wydarzenia = @Id_Wydarzenia
    )

    DECLARE @Brak_Odpowiedzi INT
```

```
    SET @Brak_Odpowiedzi = (
        SELECT COUNT(*)
        FROM Wydarzenia_Uczestnicy
        WHERE Status_Uczestnictwa = 'nie odpowiedział' AND Id_Wydarzenia = @Id_Wydarzenia
    )

    DECLARE @Zainteresowani INT
    SET @Zainteresowani = (
        SELECT COUNT(*)
        FROM Wydarzenia_Uczestnicy
        WHERE Status_Uczestnictwa = 'zainteresowany' AND Id_Wydarzenia = @Id_Wydarzenia
    )

    INSERT INTO @WydarzeniaStatusy (Id, Nazwa_Wydarzenia, Opis, Uczestnicy,
    Ludzie_którzy_odmówili, Brak_Odpowiedzi, Zainteresowani)
    SELECT Id, Nazwa_Wydarzenia, Opis, @Uczestnicy, @Ludzie_którzy_odmówili,
    @Brak_Odpowiedzi, @Zainteresowani
    FROM Wydarzenia WHERE Id = @Id_Wydarzenia


    RETURN
END
GO
```

Funkcja dla danego wydarzenia zwraca ilość osób które w jakiś sposób są powiązane z danym wydarzeniem. Każda osoba ma do wyboru 4 opcje: 'uczestniczy', 'odmówił', 'nie odpowiedział' (w przypadku gdy np. został zaproszony przez inną osobę), 'zainteresowany'. Dzięki tej funkcji na stronie danego wydarzenia będzie dało się w prosty sposó wyświetlić kluczową informację o ilości uczestników.

```
CREATE OR ALTER FUNCTION NajlepsiPosterzyWGrupie (@IdGrupy INT)
RETURNS @NajlepsiPosterzy TABLE
(
    Id_Konta INT,
    Imię VARCHAR(50),
    Nazwisko VARCHAR(50),
    Ilość_Polubień INT
)
AS
BEGIN

    INSERT INTO @NajlepsiPosterzy (Id_Konta, Imię, Nazwisko, Ilość_Polubień)
    SELECT DO.Id_Konta, DO.Imię, DO.Nazwisko, T.Ilość_Polubień
    FROM Dane_Osobowe DO
    INNER JOIN (
        SELECT TOP 5 P.Id_Autora, SUM(Ilość_Polubień) Ilość_Polubień
        FROM Posty P
        GROUP BY Id_Autora
        ORDER BY SUM(Ilość_Polubień) DESC
    ) T ON DO.Id_Konta = T.Id_Autora
    ORDER BY T.Ilość_Polubień DESC

RETURN

END
GO
```

Funkcja zwraca dane osób które są uznane jako "najlepsi posterzy" w danej grupie, przy czym jako najlepszy poster definiujemy osobę która zebrała największą liczbę polubień ze swoich wszystkich

wrzuconych postów (w obszarze tej grupy). Zawsze może być ich maksymalnie 5. Funkcja ta potrzebna jest do procedury ZaktualizujNajlepszychPosterówGrupy.

```
CREATE OR ALTER FUNCTION SzukajUżytkowników (@CiągZnakowy VARCHAR(255))
RETURNS @Użytkownicy TABLE
(
    Id_Konta INT,
    Imię VARCHAR(50),
    Nazwisko VARCHAR(50),
    Id_Zdjęcia_Profilowego INT
)
AS
BEGIN
DECLARE @PoczątekImienia VARCHAR(50)
DECLARE @PoczątekNazwiska VARCHAR(50)
SET @PoczątekImienia = (
    SELECT PARSENAME(REPLACE(@CiągZnakowy, ' ', '.'), 2)
)
SET @PoczątekNazwiska = (
    SELECT PARSENAME(REPLACE(@CiągZnakowy, ' ', '.'), 1)
)

IF(@PoczątekImienia IS NULL)
BEGIN
    SET @PoczątekImienia = @PoczątekNazwiska
    SET @PoczątekNazwiska = ''
END

INSERT INTO @Użytkownicy (Id_Konta, Imię, Nazwisko, Id_Zdjęcia_Profilowego)
SELECT d.Id_Konta, d.Imię, d.Nazwisko, z.Id_Zdjęcia
FROM Dane_Osobowe d
INNER JOIN Zdjęcia_Profilowe z ON z.Id_Konta = d.Id_Konta
WHERE d.Imię LIKE @PoczątekImienia + '%' AND d.Nazwisko LIKE @PoczątekNazwiska + '%'

RETURN

END
GO
```

Funkcja ta zwraca dane osób które pasują do wpisanego ciągu znakowego. Będzie ona wykorzystywana do wyszukiwarki, w której użytownik może wpisać imię i nazwisko osoby i otrzymać wszystkie takie osoby zarejestrowane w bazie danych, przy czym nie musi wpisywać pełnego imienia i/oraz nazwiska. Przykładowo, wpisanie "Ka" zwróci wszystkich użytkowników których imię zaczyna się na "Ka".

```
CREATE OR ALTER FUNCTION ListaZnajomych(@IdKonta INT)
RETURNS @Znajomi TABLE (
    Id_Znajomego INT,
    Imię VARCHAR(50),
    Nazwisko VARCHAR(50),
    Id_Zdjęcia_Profilowego INT
)
AS
BEGIN
    INSERT INTO @Znajomi (Id_Znajomego, Imię, Nazwisko, Id_Zdjęcia_Profilowego)
```

```
    SELECT z.Id1, d.Imię, d.Nazwisko, zp.Id_Zdjęcia
    FROM Znajomi z
    LEFT JOIN Dane_Osobowe d ON d.Id_Konta = z.Id1
    LEFT JOIN Zdjęcia_Profilowe zp ON zp.Id_Konta = d.Id_Konta
    WHERE z.Id2 = @IdKonta


    INSERT INTO @Znajomi (Id_Znajomego, Imię, Nazwisko, Id_Zdjęcia_Profilowego)
    SELECT z.Id2, d.Imię, d.Nazwisko, zp.Id_Zdjęcia
    FROM Znajomi z
    LEFT JOIN Dane_Osobowe d ON d.Id_Konta = z.Id2
    LEFT JOIN Zdjęcia_Profilowe zp ON zp.Id_Konta = d.Id_Konta
    WHERE z.Id1 = @IdKonta

    RETURN
END
GO
```

Funkcja ta zwraca listę znajomych danej osoby. Przez strukturę tabeli Znajomi proces ten jest trochę skomplikowany (nie wiemy czy szukana osoba będzie w Id1 czy w Id2), dlatego uznaliśmy że taka funkcja będzie przydatna. Wykorzystana zostanie w bardzo prostym celu - kiedy użytkownik wchodzi na swój profil, chce widzieć wszystkie osoby z którymi zawarł znajomość, co umożliwi mu ta funkcja. Zostanie ona też użyta do dwóch następnych funkcji związanych ze znajomymi.

```
CREATE OR ALTER FUNCTION ZnajomiWGrupie (@IdKonta INT, @IdGrupy INT)
RETURNS @ZnajomiWGrupie TABLE (
    Id_Znajomego INT,
    Imię VARCHAR(50),
    Nazwisko VARCHAR(50),
    Id_Zdjęcia_Profilowego INT
)
AS
BEGIN
    DECLARE @ListaZnajomych TABLE (
        Id_Znajomego INT,
        Imię VARCHAR(50),
        Nazwisko VARCHAR(50),
        Id_Zdjęcia_Profilowego INT
    )
    INSERT INTO @ListaZnajomych (Id_Znajomego, Imię, Nazwisko,Id_Zdjęcia_Profilowego)
    SELECT * FROM dbo.ListaZnajomych(@IdKonta)

    DECLARE @ListaCzłonkówGrupy TABLE (
        Id_Członka INT
    )

    INSERT INTO @ListaCzłonkówGrupy (Id_Członka)
    SELECT Id_Konta AS Id_Członka
    FROM Grupy_Członkowie
    WHERE Id_Grupy = @IdGrupy

    INSERT INTO @ZnajomiWGrupie (Id_Znajomego, Imię, Nazwisko, Id_Zdjęcia_Profilowego)
    SELECT lz.Id_Znajomego, lz.Imię, lz.Nazwisko, lz.Id_Zdjęcia_Profilowego
    FROM @ListaZnajomych lz
    WHERE lz.Id_Znajomego IN (SELECT Id_Członka FROM @ListaCzłonkówGrupy)

    RETURN
```

```
END
GO
```

Funkcja zwraca listę znajomych danej osoby które dzielą z nią tę samą grupę. Kiedy użytkownik wchodzi na stronę grupy i zastanawia się czy powinien do niej dołączyć, chce wiedzieć którzy z jego znajomych w niej są, co umożliwia ta funkcja.

```
CREATE OR ALTER FUNCTION UrodzinyZnajomych(@IdKonta INT)
RETURNS @ZnajomiZUrodzinami TABLE (
    Id_Znajomego INT,
    Imię VARCHAR(50),
    Nazwisko VARCHAR(50),
    Id_Zdjęcia_Profilowego INT
)
AS
BEGIN
    DECLARE @ListaZnajomych TABLE (
        Id_Znajomego INT,
        Imię VARCHAR(50),
        Nazwisko VARCHAR(50),
        Id_Zdjęcia_Profilowego INT
    )

    INSERT INTO @ListaZnajomych (Id_Znajomego, Imię, Nazwisko,Id_Zdjęcia_Profilowego)
    SELECT * FROM dbo.ListaZnajomych(@IdKonta)

    INSERT INTO @ZnajomiZUrodzinami (Id_Znajomego, Imię, Nazwisko, Id_Zdjęcia_Profilowego)
    SELECT lz.Id_Znajomego, lz.Imię, lz.Nazwisko, lz.Id_Zdjęcia_Profilowego
    FROM @ListaZnajomych lz
    LEFT JOIN Dane_Osobowe d ON d.Id_Konta = lz.Id_Znajomego
    WHERE MONTH(d.Urodziny) = MONTH(GETDATE()) AND DAY(d.Urodziny) = DAY(GETDATE())

    RETURN
END
GO
```

Funkcja zwraca listę znajomych danej osoby które mają urodziny. Będzie ona wykorzystana do powiadomień które zostaną wysłane do osoby aby przypomnieć o urodzinach jej znajomych.

6 Opis Stworzonych Procedur

```
CREATE OR ALTER PROCEDURE DodajAdresOsobie(
    @IdOsoby INT,
    @Województwo VARCHAR(19),
    @Miasto VARCHAR(30),
    @Ulica VARCHAR(30),
    @KodPocztowy VARCHAR(6))
AS

DECLARE @IdAdresu INT
SET @IdAdresu = (SELECT Id FROM Adresy WHERE Województwo = @Województwo AND
Miasto = @Miasto AND Ulica = @Ulica)

IF(@IdAdresu IS NULL)
BEGIN
    DECLARE @LastId INT
    SET @LastId = (SELECT MAX(id) FROM Adresy)
```

```
    IF(@LastId IS NULL)
    BEGIN
        SET @LastId = 0
    END
    ELSE
    BEGIN
        SET @LastId = @LastId + 1
    END

    INSERT INTO Adresy VALUES(@LastId, @Województwo, @Miasto, @Ulica, @KodPocztowy)

    UPDATE d
    SET d.Id_Adresu = @LastId
    FROM Dane_Osobowe d
END
ELSE
BEGIN
    UPDATE d
    SET d.Id_Adresu = @IdAdresu
    FROM Dane_Osobowe d
END
GO
```

Procedura DodajAdresOsobie przypusje jakiejś osobie adres. Jeśli ten adres znajduje się już w tabeli Adresy, poprostu ustawiamy w tabeli Dane Osobowe id tego adresu, w przeciwnym razie dodajemy i go robimy to samo. Procedura ta pozwala na swobodne dodawanie adresu osobie, bez konieczności spradzania czy dany adres znajduje się w bazie danych.

```
CREATE OR ALTER PROCEDURE DodajAdresWydarzeniu(
    @IdWydarzenia INT,
    @Województwo VARCHAR(19),
    @Miasto VARCHAR(30),
    @Ulica VARCHAR(30),
    @KodPocztowy VARCHAR(6))
AS

DECLARE @IdAdresu INT
SET @IdAdresu = (SELECT Id FROM Adresy WHERE Województwo = @Województwo AND
Miasto = @Miasto AND Ulica = @Ulica)

IF(@IdAdresu IS NULL)
BEGIN
    DECLARE @LastId INT
    SET @LastId = (SELECT MAX(id) FROM Adresy)
    IF(@LastId IS NULL)
    BEGIN
        SET @LastId = 0
    END
    ELSE
    BEGIN
        SET @LastId = @LastId + 1
    END

    INSERT INTO Adresy VALUES(@LastId, @Województwo, @Miasto, @Ulica, @KodPocztowy)

    UPDATE w
    SET w.Id_Adresu = @LastId
    FROM Wydarzenia w
```

```
END
ELSE
BEGIN
    UPDATE w
    SET w.Id_Adresu = @IdAdresu
    FROM Wydarzenia w
END
GO
```

Procedura DodajAdresWydarzeniu działa podobnie do poprzedniej procedury, lecz tym razem dodajemy adres do wydarzenia, nie do osoby.

```
CREATE OR ALTER PROCEDURE PrzywróćUsuniętyPost(@IdPostu INT)
AS
BEGIN TRANSACTION

    DECLARE @Stan VARCHAR(9)
    SET @Stan = (
        SELECT Stan FROM Posty_Archiwum
        WHERE Id = @IdPostu
    )

    IF(@Stan <> 'usunięcie')
    BEGIN
        ROLLBACK
        RAISERROR('Post nigdy nie został usunięty!', 16, 1)
    END

    INSERT INTO Posty (Treść, Id_Autora, Id_Grupy, Ilość_Polubień, Data_Dodania)
    SELECT Treść, Id_Autora, Id_Grupy, Ilość_Polubień, Data_Dodania
    FROM Posty_Archiwum
    WHERE Id = @IdPostu

    DELETE FROM Posty_Archiwum
    WHERE Id = @IdPostu

COMMIT
GO
```

Procedura PrzywróćUsuniętyPost jest używana do przywracania postu, który wcześniej został usunięty oraz przeniesiony do archiwum. Procedure można stosować np. jeśli post został przypadkowo usunięty.

```
CREATE OR ALTER PROCEDURE UsuńPost(@IdUżytkownika INT, @IdPostu INT)
AS
BEGIN TRANSACTION

DECLARE @IdAutora INT
SET @IdAutora = (
    SELECT Id_Autora
    FROM Posty
    WHERE Id = @IdPostu
)

IF (@IdAutora = @IdUżytkownika)
```

```
BEGIN
    DELETE FROM Posty
    WHERE Id = @IdPostu

    DELETE FROM Komentarze
    WHERE Id_Postu = @IdPostu
END
ELSE
BEGIN
    DECLARE @IdGrupy INT
    SET @IdGrupy = (
        SELECT Id_Grupy FROM Posty
        WHERE Id = @IdPostu
    )

    DECLARE @Uprawnienia INT
    SET @Uprawnienia =  (
        SELECT Uprawnienia
        FROM Moderatorzy_Grup
        WHERE Id_Grupy = @IdGrupy AND Id_Moderatora = @IdUżytkownika
        )
    IF(@Uprawnienia <> 2 AND @Uprawnienia <> 3)
    BEGIN
        ROLLBACK
        RAISERROR('Brak uprawnień użytkownika do usunięcia postu!', 16, 1)
    END
    ELSE
    BEGIN
        DELETE FROM Posty
        WHERE Id = @IdPostu

        DELETE FROM Komentarze
        WHERE Id_Postu = @IdPostu
    END
END
COMMIT
GO
```

Procedura UsuńPost pozwala usunąć niepotrzebny post. Post może być usunięty tylko przez autora lub ewentualnie moderatora grupy (z poziomem uprawnień 2 lub 3) jeśli post się w takowej znajduje. Procedura może być wykorzystywana w sytuacji gdy będziemy chcieli usunąć np. post który jest spamem.

```
CREATE OR ALTER PROCEDURE ZmieńZdjęcieProfilowe (@IdKonta INT,
@Ścieżka VARCHAR(255), @IdNowegoProfilowego INT = NULL,
@Podpis VARCHAR(30) = NULL, @Wymiary VARCHAR(11) = NULL)
AS
BEGIN TRANSACTION

    IF ((SELECT COUNT (*) FROM Konta WHERE Id = @IdKonta) = 0) BEGIN
        ROLLBACK
        RAISERROR('Brak konta któremu probowano zmienić zdjęcie profilowe', 16, 1)
    END

    IF ((SELECT COUNT(*) FROM Zdjęcia WHERE Id = @IdNowegoProfilowego) = 1)
    BEGIN
        UPDATE Zdjęcia_Profilowe
        SET Id_Zdjęcia = @IdNowegoProfilowego
```

```
        WHERE Id_Konta = @IdKonta
    END
    ELSE
    BEGIN
        INSERT INTO Zdjęcia (Ścieżka, Id_Konta, Podpis, Wymiary)
        VALUES (@Ścieżka, @IdKonta, @Podpis, @Wymiary)

        SET @IdNowegoProfilowego = (
            SELECT Id
            FROM Zdjęcia
            WHERE Ścieżka = @Ścieżka
        )

        UPDATE Zdjęcia_Profilowe
        SET Id_Zdjęcia = @IdNowegoProfilowego
        WHERE Id_Konta = @IdKonta
    END
COMMIT
GO
```

Procedura ZmieńZdjęcieProfilowe pozwala na zmianę aktualnego zdjęcia profilowego. Możemy wybrać istniejące już zdjęcie które było przypisane do profilu, lecz nie było zdjęciem profilowym lub dodać nowe zdjęcie oraz od razu ustawić je na zdjęcie profilowe.

```
CREATE OR ALTER PROCEDURE ZaktualizujNajlepszychPosterówGrupy(@IdGrupy INT)
AS
BEGIN TRANSACTION

    DECLARE @NajlepsiPosterzy TABLE (
        IdPostera INT
    )

    INSERT INTO @NajlepsiPosterzy (IdPostera)
    SELECT Id_Konta FROM dbo.NajlepsiPosterzyWGrupie(@IdGrupy)

    UPDATE gc
    SET Najlepszy_Poster = 0
    FROM Grupy_Członkowie gc
    WHERE Id_Grupy = @IdGrupy

    UPDATE gc
    SET Najlepszy_Poster = 1
    FROM Grupy_Członkowie gc
    WHERE Id_Konta IN (SELECT * FROM @NajlepsiPosterzy) AND Id_Grupy = @IdGrupy

    COMMIT
GO
```

Procedura ta będzie wykorzystywana do aktualizowania informacji kto jest najlepszym posterem w danej grupie (ma największą liczbę polubień we wszystkich wrzuconych postach w danej grupie). Każda taka osoba ma przypisaną 1 w atrybucie Najlepszy Poster (będzie ich zawsze maksymalnie pięciu), reszta ma 0.

7 Opis Stworzonych Wyzwalaczy

```
CREATE OR ALTER TRIGGER ArchiwizujPostyUpdate
ON Posty
AFTER UPDATE
AS
IF UPDATE(Treść) BEGIN
    INSERT INTO Posty_Archiwum (Treść, Id_Autora, Id_Grupy, Ilość_Polubień, Data_Dodania,
    Data_Zmiany, Stan)
    SELECT Treść, Id_Autora, Id_Grupy, Ilość_Polubień, Data_Dodania, GETDATE(), 'edycja'
    FROM Inserted
END
GO
```

Wyzwalacz włącza się po edycji postu. Ma on za zadanie wrzucić starą wersję posta (sprzed aktualizacji jego treści) do tabeli Posty Archiwum. Dzięki temu mamy w bazie danych system kontroli wersji i zawsze będzie można powrócić do starszej wersji w przypadku jakiegoś błędu.

```
CREATE OR ALTER TRIGGER ArchiwizujPostyDelete
ON Posty
INSTEAD OF DELETE
AS

IF ((SELECT COUNT (*) FROM Posty P INNER JOIN Deleted ON P.Id = Deleted.Id) = 0) BEGIN
    ROLLBACK
    RAISERROR('Nie istnieje post który próbowano usunąć', 16, 1)
END

IF ((SELECT COUNT (*) FROM Posty P INNER JOIN Deleted ON P.Id = Deleted.Id) > 1) BEGIN
    ROLLBACK
    RAISERROR('Można usunąć tylko jeden post naraz', 16, 1)
END

DECLARE @DeletedID INT
SET @DeletedID = (
    SELECT Id FROM Deleted
)

DELETE FROM Posty
WHERE Id = @DeletedID

INSERT INTO Posty_Archiwum (Treść, Id_Autora, Id_Grupy, Ilość_Polubień, Data_Dodania, Data_Zmiany,
SELECT Treść, Id_Autora, Id_Grupy, Ilość_Polubień, Data_Dodania, GETDATE(), 'usunięcie'
FROM Deleted

DELETE FROM Komentarze
WHERE Id_Postu = @DeletedID

GO
```

Wyzwalacz działa na podobnej zasadzie jak ten powyżej, tylko dodaje on do Posty Archiwum posty które zostały usunięte. Ponadto nie pozwala on na usunięcie więcej niż jednego postu na raz.

```
CREATE OR ALTER TRIGGER DodajKonto
ON Konta
```

```
INSTEAD OF INSERT
AS

DECLARE @Count INT
SET @Count = (
    SELECT COUNT(*) FROM inserted
)

IF(@Count > 1)
BEGIN
    ROLLBACK
    RAISERROR('Można dodać tylko jednego użytkownika na raz', 16, 1)
END

DECLARE @Id INT
DECLARE @Login VARCHAR(20)

SET @Login = (
    SELECT Login FROM inserted
)

DECLARE @Hasło VARCHAR(255)
SET @Hasło = (
    SELECT Hasło FROM inserted
)

SET @Hasło = (SELECT HASHBYTES('SHA2_256', @Hasło))

INSERT INTO Konta (Login, Hasło, Email)
SELECT Login, @Hasło, Email FROM inserted

SET @Id = (
    SELECT Id FROM Konta WHERE Login = @Login
)

INSERT INTO Zdjęcia_Profilowe (Id_Konta, Id_Zdjęcia) VALUES(@Id, null)

GO
```

Wyzwalacz dodaje nowe konto do bazy danych. Jednak trzymanie hasła jako zwykły tekst nie byłoby bezpieczne, dlatego na poziomie bazy danych używamy systemu hashowania tak aby nawet jeśli ktoś dostał się do bazy danych nie był w stanie z łatwością przechwycić dostępu do kont. Oprócz tego nie pozwalamy na dodanie dwóch lub więcej użytkowników za jednym razem.

```
CREATE OR ALTER TRIGGER EdytujKonto
ON Konta
INSTEAD OF UPDATE
AS

IF ((SELECT COUNT(*) FROM inserted) > 1)
BEGIN
    ROLLBACK
    RAISERROR('Można zrobić tylko jedną aktualizację konta na raz!', 16, 1)
END

IF (UPDATE(Hasło) AND UPDATE(Email))
BEGIN
    ROLLBACK
```

```
    RAISERROR('Nie można aktualizować hasła i emailu jednocześnie!', 16, 1)
END
IF (UPDATE(Login))
BEGIN
    ROLLBACK
    RAISERROR('Nie można aktualizować loginu konta!', 16, 1)
END

DECLARE @Id INT
SET @Id = (
    SELECT Id FROM inserted
)

DECLARE @LiczbaZnajomychWcześniej INT
DECLARE @LiczbaZnajomychPóźniej INT
SET @LiczbaZnajomychWcześniej = (
    SELECT Liczba_Znajomych
    FROM Konta
    WHERE Id = @Id
)
SET @LiczbaZnajomychPóźniej = (
    SELECT Liczba_Znajomych
    FROM inserted
)

IF(@LiczbaZnajomychPóźniej - @LiczbaZnajomychWcześniej <> 0 AND
    @LiczbaZnajomychPóźniej - @LiczbaZnajomychWcześniej <> 1 AND
    @LiczbaZnajomychPóźniej - @LiczbaZnajomychWcześniej <> -1)
BEGIN
    ROLLBACK
    RAISERROR('Liczba znajomych może się zmienić co najwyżej o 1!', 16, 1)
END

IF (UPDATE(Hasło))
BEGIN
    DECLARE @Hasło VARCHAR(255)
    SET @Hasło = (
        SELECT Hasło FROM inserted
    )

    SET @Hasło = (SELECT HASHBYTES('SHA2_256', @Hasło))

    UPDATE k
    SET Hasło = @Hasło
    FROM Konta k
END

IF (UPDATE(Email))
BEGIN
    DECLARE @Email VARCHAR(30)
    SET @Email = (
        SELECT Email FROM inserted
    )

    UPDATE k
    SET Email = @Email
    FROM Konta k
END

IF (UPDATE(Liczba_Znajomych))
```

```
BEGIN
    UPDATE k
    SET Liczba_Znajomych = @LiczbaZnajomychPóźniej
    FROM Konta k
    WHERE k.Id = @Id
END

GO
```

Wyzwalacz odpowiada za edycję konta. Tak jak poprzednio, musimy dbać o bezpieczeństwo bazy danych i używamy systemu hashowania do nowego hasła. Oprócz tego wyzwalacz dba o to, żeby dało się zaktualizować tylko jedno konto na raz i żeby zmiana znajomości była pojedyńcza. Nie pozwala on na zmianę loginu (chcemy żeby każdy użytownik miał zawsze login który z łatwoscią go zidentyfikuje). Nie można też zmienić jednocześnie emaila i hasła gdyż mogłoby to prowadzić do wielu problemów - przy zmianie hasła użytownik musi potwierdzić nowe hasło przy pomocy emailu, mogłoby się zdarzyć że nowe hasło będzie wysłane na stary email.

```
CREATE OR ALTER TRIGGER DodajZnajomych
ON Znajomi
AFTER INSERT
AS

DECLARE @Id1 INT
DECLARE @Id2 INT
SET @Id1 = (
    SELECT Id1 FROM inserted
)

SET @Id2 = (
    SELECT Id2 FROM inserted
)

UPDATE k
SET k.Liczba_znajomych = k.Liczba_znajomych + 1
FROM Konta k
WHERE k.Id = @Id1

UPDATE k
SET k.Liczba_znajomych = k.Liczba_znajomych + 1
FROM Konta k
WHERE k.Id = @Id2

GO


-----------------------------------------------------
CREATE OR ALTER TRIGGER UsuńZnajomych
ON Znajomi
AFTER DELETE
AS

DECLARE @Id1 INT
DECLARE @Id2 INT
SET @Id1 = (
    SELECT Id1 FROM deleted
)

SET @Id2 = (
```

```
    SELECT Id2 FROM deleted
)

UPDATE k
SET k.Liczba_znajomych = k.Liczba_znajomych - 1
FROM Konta k
WHERE k.Id = @Id1

UPDATE k
SET k.Liczba_znajomych = k.Liczba_znajomych - 1
FROM Konta k
WHERE k.Id = @Id2

GO
```

Oba wyzwalacze działają w taki sam sposób. W tabeli Konta mamy atrybut który zmienia się w czasie (Liczba znajomych). Musi on być atualizowany za każdym razem, kiedy jakieś dwie osoby albo dodadzą się do znajomych, albo usuną się ze znajomych. Wyzwalacze te dbają o to, aby atrybut ten zawsze zmieniał się poprawnie.