

Dokumentacja projektu - System Bazy Danych dla Kliniki

**Artur Saganowski,
IS, 3.rok**

I. Projekt koncepcji i założenia

1. Temat projektu

Tworzenie systemu bazy danych dla kliniki, umożliwiającego zarządzanie pacjentami, wizytami, specjalistami oraz placówkami.

Cele projektu:

- Umożliwienie umawiania wizyt dla pacjentów z konkretnymi specjalistami w wybranych placówkach.
- Dodawanie placówek do bazy.
- Optymalizacja zarządzania danymi o pracownikach i pacjentach.

II. Projekt diagramów (Konceptualny)

1. Główne tabele:

- Użytkownicy (z podziałem na role: pacjent, specjalista, admin).

```
CREATE TABLE IF NOT EXISTS klinika.uzyskownicy (  
    id_uzyskownika SERIAL PRIMARY KEY,  
    login VARCHAR(50) UNIQUE NOT NULL,  
    haslo VARCHAR(255) NOT NULL,  
    rola rola_typ NOT NULL,  
    specjalizacja specjalizacja_typ,  
    CHECK (  
        (rola = 'SPECJALISTA' AND specjalizacja IS NOT NULL) OR  
        (rola != 'SPECJALISTA' AND specjalizacja IS NULL)  
    )  
);  
  
ALTER TABLE klinika.uzyskownicy  
ADD COLUMN imie VARCHAR(50) NOT NULL,  
ADD COLUMN nazwisko VARCHAR(50) NOT NULL;
```

- Placówki (z informacją o specjalistach pracujących w placówkach).

```
CREATE TABLE IF NOT EXISTS klinika.placowki (  
    id_placowki SERIAL PRIMARY KEY,  
    nazwa VARCHAR(100) NOT NULL,  
    adres VARCHAR(200) NOT NULL,  
    telefon VARCHAR(20)  
);
```

- Wizyty (związane z pacjentem, specjalistą, i placówką)

```
CREATE TABLE IF NOT EXISTS klinika.wizyty (  
    id_wizyty SERIAL PRIMARY KEY,  
    data_wizyty TIMESTAMP NOT NULL,  
    id_placowki INT NOT NULL,  
    id_pacjenta INT NOT NULL,  
    id_specjalisty INT NOT NULL,  
    FOREIGN KEY (id_placowki) REFERENCES klinika.placowki (id_placowki)  
        ON DELETE CASCADE,  
    FOREIGN KEY (id_pacjenta) REFERENCES klinika.uzytownicy (id_uzytownika  
        ON DELETE CASCADE,  
    FOREIGN KEY (id_specjalisty) REFERENCES klinika.uzytownicy (id_uzytown  
        ON DELETE CASCADE,  
    CHECK (id_pacjenta != id_specjalisty)  
);
```

Wizyty są generowane poprzez funkcję, aby rozróżniać informacje dla specjalisty oraz dla pacjenta (wykorzystuje wbudowane funkcje oraz agregujące by określić za ile dni jest wizyta)

```

CREATE OR REPLACE FUNCTION klinika.get_wizyty(
    in_id_uzytkownika INT,
    in_rola rola_typ
)
RETURNS TABLE (
    id_wizyty INT,
    data_wizyty TIMESTAMP,
    nazwa_placowki varchar(100),
    adres_placowki VARCHAR(200),
    uczestnik TEXT,
    specjalizacja specjalizacja_typ,
    za_ile_dni INTEGER
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        w.id_wizyty,
        w.data_wizyty,
        p.nazwa AS nazwa_placowki,
        p.adres AS adres_placowki,
        CASE
            WHEN in_rola = 'PACJENT' THEN spec.imie || ' ' || spec.nazwisko
            WHEN in_rola = 'SPECJALISTA' THEN pac.imie || ' ' || pac.nazwisko
        END AS uczestnik,
        spec.specjalizacja,
        ROUND(DATE_PART('day', w.data_wizyty - CURRENT_DATE))::INTEGER AS za_ile_dni
    FROM
        klinika.wizyty w
    JOIN
        klinika.placowki p ON w.id_placowki = p.id_placowki
    JOIN
        klinika.uzytkownicy pac ON w.id_pacjenta = pac.id_uzytkownika
    JOIN
        klinika.uzytkownicy spec ON w.id_specjalisty = spec.id_uzytkownika
    WHERE
        (in_rola = 'PACJENT' AND w.id_pacjenta = in_id_uzytkownika) OR
        (in_rola = 'SPECJALISTA' AND w.id_specjalisty = in_id_uzytkownika)
    ORDER BY
        w.data_wizyty ASC;
END;
$$ LANGUAGE plpgsql;

```

2. Powiązania:

- Relacje jeden-do-wielu między użytkownikami i wizytami.
- Relacje wiele-do-wielu między specjalistami a placówkami, realizowane poprzez tabelę pośrednią.

III. Projekt logiczny

W projekcie posłużyłem się listami enumerowanymi:

```
CREATE TYPE rola_typ AS ENUM ('ADMIN', 'PACJENT', 'SPECJALISTA');
CREATE TYPE specjalizacja_typ AS ENUM (
    'Kardiologia',
    'Pediatria',
    'Dermatologia',
    'Chirurgia',
    'Onkologia',
    'Neurologia'
);
```

Triggery walidujące czy

- dany specjalista nie ma zaplanowanej innej wizyty w podanym czasie
- nie próbujemy przypisać specjalizacji użytkownikowi innemu niż specjalista.

```
CREATE OR REPLACE FUNCTION klinika.sprawdz_konflikty_wizyt()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM klinika.wizyty
        WHERE id_specjalisty = NEW.id_specjalisty
        AND NEW.data_wizyty BETWEEN
            data_wizyty - INTERVAL '15 minutes' AND data_wizyty + INTERVAL '15
    ) THEN
        RAISE EXCEPTION 'Specjalista ma już wizytę zaplanowaną na ten czas';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER sprawdz_konflikty_wizyt_trigger
BEFORE INSERT OR UPDATE ON klinika.wizyty
FOR EACH ROW
EXECUTE FUNCTION klinika.sprawdz_konflikty_wizyt();
```

W bazie mamy również widoki:

- selecta łączący tabelę placówek z funkcją agregującą liczącą ilość specjalistów w każdej z nich
- selecta zwracający użytkowników bez admina (bo Admini nie muszą siebie widzieć :)) wraz ze specjalizacją i placówkami, do których należą specjaliści

Oba wykorzystują GROUP BY oraz HAVING

```
CREATE OR REPLACE VIEW klinika.placowki_z_iloscia_specjalistow AS
SELECT
  p.id_placowki,
  p.nazwa,
  p.adres,
  p.telefon,
  COUNT(sp.id_specjalisty) AS ilosc_specjalistow
FROM
  klinika.placowki p
LEFT JOIN
  klinika.specjalista_placowki sp ON p.id_placowki = sp.id_placowki
GROUP BY
  p.id_placowki
ORDER BY
  p.nazwa ASC;
```



```

CREATE OR REPLACE VIEW klinika.uzytownicy_z_placowkami AS
SELECT
    u.id_uzytownika,
    u.login,
    u.rola,
    u.specjalizacja,
    u.imie,
    u.nazwisko,
    CASE
        WHEN u.rola = 'SPECJALISTA' THEN STRING_AGG(p.nazwa, ', ')
        ELSE NULL
    END AS placowki
FROM
    klinika.uzytownicy u
LEFT JOIN
    klinika.specjalista_placowki sp 1<->0..n: ON u.id_uzytownika = sp.id_specjalisty
LEFT JOIN
    klinika.placowki p 1..n<->1: ON sp.id_placowki = p.id_placowki
GROUP BY
    u.id_uzytownika, u.login, u.rola, u.specjalizacja, u.imie, u.nazwisko
HAVING
    u.rola != 'ADMIN';

```

IV. Projekt funkcjonalny

1. Formularz dodawania użytkowników oraz raport z niego dostępny dla roli ADMIN:
 - Pola: dane użytkownika takie jak np. login, imię, rola, ale również dla specjalistów - specjalizacja oraz przypisane placówki.
2. Formularz dodawania wizyt (dostępny dla Pacjenta) oraz raport dla specjalisty oraz pacjenta:
 - Pola: data wizyty, placówka, specjalista z walidacją na date i godzinę (opisaną w punkcie III).
3. Formularz dodawania oraz usuwania placówki oraz raport z nich plus dane statystyczne takie jak ilość specjalistów w placówce:
 - Pola takie jak: nazwa, adres, telefon

V. Dokumentacja

Wprowadzanie danych ręczne z poziomu psql oraz aplikacji webowej.

Dla niezalogowanych użytkowników dostępny jest panel z listą placówek.

Możliwość zalogowania na jedną z trzech ról (ADMIN, PACJENT, SPECJALISTA) oraz w zależności od roli istnieją uprawnienia do edycji poszczególnych tabel.

Stack technologiczny, w jakim była pisana aplikacja to

- Angular
- Node.js
- PostgreSQL

Wykaz literatury:

- Wykłady z zajęć
- Dokumentacja Angulara