**Projektowanie i implementacja systemów bazodanowych**

Stworzenie systemu do zarządzania biblioteką, umożliwiającego wyszukiwanie książek, wypożyczanie oraz oddawanie

Tomasz Dydacki 48015

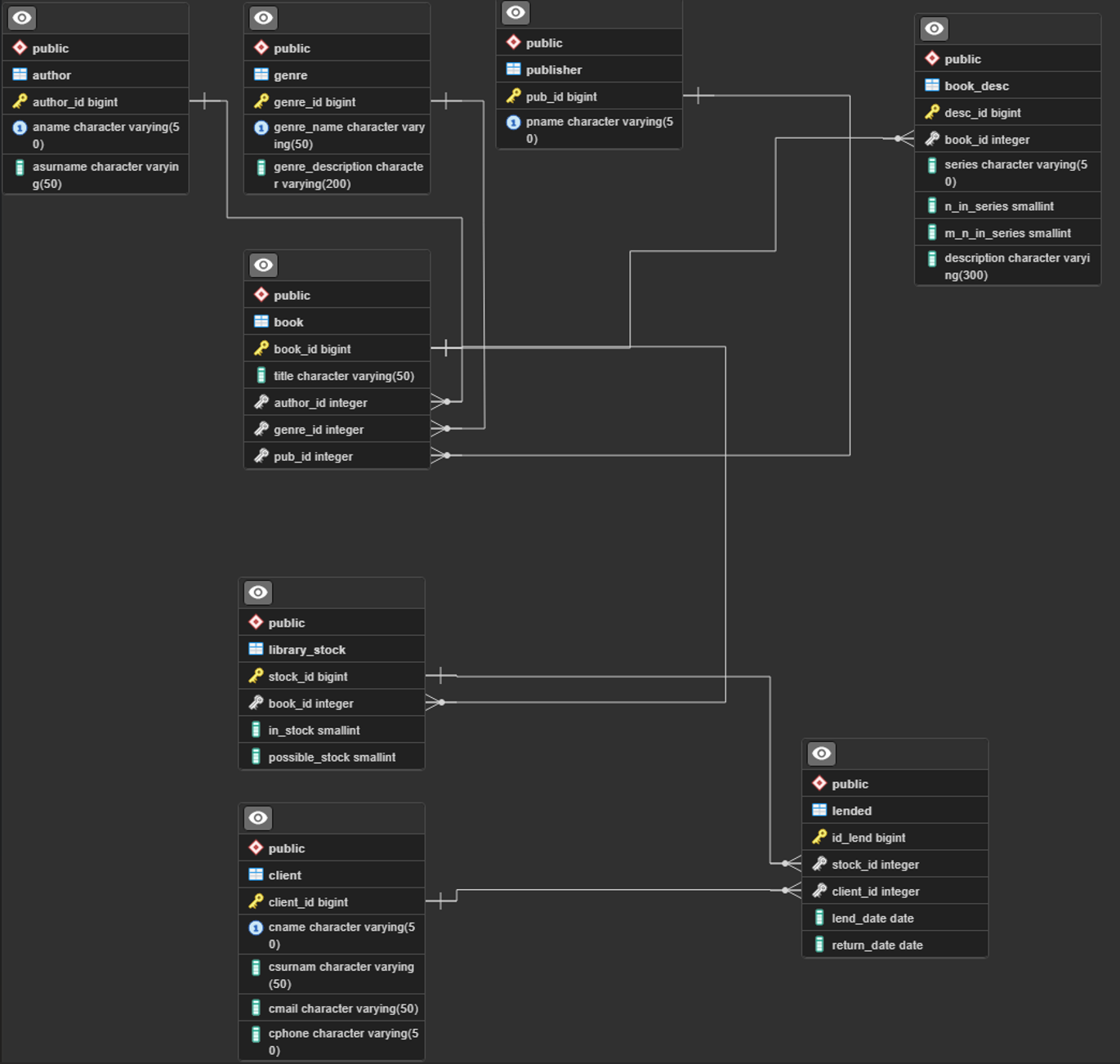
[BashiQz/Library (github.com)](https://github.com/BashiQz/Library)

https://github.com/BashiQz/Library

1. Wstęp

Celem jest stworzenie bazy danych biblioteki, która pozwoli nam sprawdzać informację o posiadanych książkach i użytkownikach. Najważniejsza jest możliwość kontrolowania działalności, czyli śledzenie wypożyczonych książek i tych aktualnie dostępnych.

1. Projektowanie systemu bazodanowego
   1. Diagram związków Encji



1. Implementacja systemu bazodanowego

Jako bazę danych użyto PostgreSQL. Do inicjalizacji środowiska został użyty Docker z kontenerem posiadającym obraz w najnowszej wersji. Nie był to krok konieczny, jednak znacznie ułatwił pracę deweloperom. PostgreSQL jako baza danych była najbardziej odpowiednim wyborem z powodu prostej integracji ze środowiskiem .NET. Jest to relacyjna baza danych, która najlepiej wpasowywała się w wymogi projektu, ponieważ logika projektu nie wymaga zastosowania bardziej zaawansowanych technologii niż bazy relacyjne. Do projektu mogła zostać wybrana prawdopodobnie jakakolwiek inna baza relacyjna typuSQL Server, SQLite czy OracleDB, jednak implementacja warstwy bazodanowej musiałaby zostać nieznacznie zmodyfikowana.

* 1. Tworzenie tabeli

--Tworzenie tabeli genre

CREATE TABLE genre (

genre\_id bigserial PRIMARY KEY,

genre\_name VARCHAR(50) UNIQUE NOT NULL,

genre\_description VARCHAR(200) NOT NULL

);

--Tworzenie tabeli author

CREATE TABLE Author (

author\_id bigserial PRIMARY KEY,

aname VARCHAR(50) UNIQUE NOT NULL,

asurname VARCHAR(50) NOT NULL

);

--Tworzenie tabeli publisher

CREATE TABLE Publisher (

pub\_id bigserial PRIMARY KEY,

pname VARCHAR(50) UNIQUE NOT NULL

);

--Tworzenie tabeli book

CREATE TABLE BOOK (

book\_id bigserial PRIMARY KEY,

title VARCHAR(50) NOT NULL,

author\_id INT,

genre\_id INT,

pub\_id INT,

FOREIGN KEY (author\_id) REFERENCES Author(author\_id),

FOREIGN KEY (genre\_id) REFERENCES genre(genre\_id),

FOREIGN KEY (pub\_id) REFERENCES Publisher(pub\_id)

);

--Tworzenie tabeli book\_desc

CREATE TABLE BOOK\_desc (

desc\_id bigserial PRIMARY KEY,

book\_id INT,

series VARCHAR(50),

n\_in\_series SMALLINT,

m\_n\_in\_series SMALLINT,

description VARCHAR(300),

FOREIGN KEY (book\_id) REFERENCES BOOK(book\_id)

);

--Tworzenie tabeli client

CREATE TABLE client (

client\_id bigserial PRIMARY KEY,

cname VARCHAR(50) UNIQUE NOT NULL,

csurnam VARCHAR(50) NOT NULL,

cmail VARCHAR(50) NOT NULL,

cphone VARCHAR(50) NOT NULL

);

--Tworzenie tabeli library\_stock

CREATE TABLE library\_stock (

stock\_id bigserial PRIMARY KEY,

book\_id INT,

in\_stock SMALLINT NOT NULL,

possible\_stock SMALLINT NOT NULL,

FOREIGN KEY (book\_id) REFERENCES BOOK(book\_id)

);

--Tworzenie tabeli lended

CREATE TABLE lended (

id\_lend bigserial PRIMARY KEY,

stock\_id INT,

client\_id INT,

lend\_date DATE,

return\_date DATE,

FOREIGN KEY (stock\_id) REFERENCES library\_stock(stock\_id),

FOREIGN KEY (client\_id) REFERENCES client(client\_id)

);

* 1. Wypełnianie tabeli

Tabele zostały wypełnione losowymi danymi, na których wykonywane były testy i tworzona struktura.

* 1. Tworzenie indeksów

-- Indeksy dla tabeli author

CREATE INDEX idx\_author\_id ON author(author\_id);

CREATE INDEX idx\_author\_name ON author(aname);

CREATE INDEX idx\_author\_surname ON author(asurname);

-- Indeksy dla tabeli book

CREATE INDEX idx\_book\_id ON book(book\_id);

CREATE INDEX idx\_book\_author\_id ON book(author\_id);

-- Indeksy dla tabeli book\_desc

CREATE INDEX idx\_book\_desc\_book\_id ON book\_desc(book\_id);

-- Indeksy dla tabeli publisher

CREATE INDEX idx\_publisher\_pub\_id ON publisher(pub\_id);

CREATE INDEX idx\_publisher\_pname ON publisher(pname);

-- Indeksy dla tabeli genre

CREATE INDEX idx\_genre\_genre\_id ON genre(genre\_id);

CREATE INDEX idx\_genre\_genre\_name ON genre(genre\_name);

-- Indeksy dla tabeli client

CREATE INDEX idx\_client\_client\_id ON client(client\_id);

CREATE INDEX idx\_client\_mail ON client(cmail);

CREATE INDEX idx\_client\_phone ON client(cphone);

-- Indeksy dla tabeli library\_stock

CREATE INDEX idx\_library\_stock\_stock\_id ON library\_stock(stock\_id);

CREATE INDEX idx\_library\_stock\_book\_id ON library\_stock(book\_id);

* 1. Tworzenie widoków

--Widok, który umożliwia wyświetlanie informacji o książce.

CREATE OR REPLACE VIEW vw\_book\_info AS

SELECT

b.book\_id,

b.title AS book\_title,

a.aname AS author\_name,

a.asurname AS author\_surname,

bd.series,

bd.n\_in\_series,

bd.m\_n\_in\_series,

bd.description AS book\_description,

p.pname AS publisher\_name,

g.genre\_name

FROM

book b

JOIN

author a ON b.author\_id = a.author\_id

JOIN

book\_desc bd ON b.book\_id = bd.book\_id

JOIN

publisher p ON b.pub\_id = p.pub\_id

JOIN

genre g ON b.genre\_id = g.genre\_id;

-- Widok używany do uproszczonego użytku poprzedniego

CREATE OR REPLACE VIEW vw\_books\_by AS

SELECT

book\_id,

book\_title,

author\_name,

author\_surname,

series,

n\_in\_series,

m\_n\_in\_series,

book\_description,

publisher\_name,

genre\_name

FROM

vw\_book\_info;

--Widok, który umożliwia wyświetlanie informacji o wypożyczonych pozycjach.

CREATE OR REPLACE VIEW vw\_lenders AS

SELECT

b.book\_id,

b.title AS book\_title,

a.aname AS author\_name,

a.asurname AS author\_surname,

l.client\_id,

c.cname AS client\_name,

c.csurnam AS client\_surname,

c.cmail AS client\_mail,

c.cphone AS client\_phone,

l.lend\_date,

l.return\_date

FROM

lended l

JOIN

library\_stock ls ON l.stock\_id = ls.stock\_id

JOIN

book b ON ls.book\_id = b.book\_id

JOIN

author a ON b.author\_id = a.author\_id

JOIN

client c ON l.client\_id = c.client\_id;

--

CREATE OR REPLACE VIEW vw\_lenders\_by AS

SELECT

book\_id,

book\_title,

author\_name,

author\_surname,

client\_id,

client\_name,

client\_surname,

client\_mail,

client\_phone,

lend\_date,

return\_date

FROM

vw\_lenders;

* 1. Tworzenie procedur

--Tworzenie procedury automatycznego zmianu stanu posiadanych tytułów - zmniejszanie

CREATE OR REPLACE PROCEDURE decrement\_stock(p\_stock\_id INT)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE library\_stock

SET in\_stock = in\_stock - 1

WHERE stock\_id = p\_stock\_id;

END;

$$;

--Tworzenie procedury automatycznego zmianu stanu posiadanych tytułów - zwiększanie

CREATE OR REPLACE PROCEDURE increment\_stock(p\_stock\_id INT)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE library\_stock

SET in\_stock = in\_stock + 1

WHERE stock\_id = p\_stock\_id;

END;

$$;

-- Tworzenie funkcji dla Triggera zmniejszania stanu

CREATE OR REPLACE FUNCTION trigger\_decrement\_stock()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

CALL decrement\_stock(NEW.stock\_id);

RETURN NEW;

END;

$$;

-- Tworzenie funkcji dla Triggera zwiększania stanu

CREATE OR REPLACE FUNCTION trigger\_increment\_stock\_on\_return()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

IF NEW.return\_date IS NOT NULL AND OLD.return\_date IS NULL THEN

CALL increment\_stock(NEW.stock\_id);

END IF;

RETURN NEW;

END;

$$;

-- Trigger dla operacji INSERT

CREATE TRIGGER before\_insert\_lended

BEFORE INSERT ON lended

FOR EACH ROW

EXECUTE FUNCTION trigger\_decrement\_stock();

-- Trigger dla zmiany w kolumnie return\_date

CREATE TRIGGER after\_update\_return\_date

AFTER UPDATE OF return\_date ON lended

FOR EACH ROW

WHEN (OLD.return\_date IS NULL AND NEW.return\_date IS NOT NULL)

EXECUTE FUNCTION trigger\_increment\_stock\_on\_return();

-- Funkcja zapobiegająca wypożyczeniu książki, której nie posiadamy

CREATE OR REPLACE FUNCTION check\_stock\_before\_lend()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

-- Sprawdza, czy książka jest na stanie

IF (SELECT in\_stock FROM library\_stock WHERE stock\_id = NEW.stock\_id) <= 0 THEN

RAISE EXCEPTION 'Cannot lend a book that is out of stock';

END IF;

RETURN NEW;

END;

$$;

-- Trigger przy operacji INSERT do sprawdzenia stanu

CREATE TRIGGER before\_insert\_check\_stock

BEFORE INSERT ON lended

FOR EACH ROW

EXECUTE FUNCTION check\_stock\_before\_lend();

1. Przykładowe zapytania

--Zwykłe zapytanie

SELECT

b.title AS book\_title,

a.aname AS author\_name,

a.asurname AS author\_surname,

bd.description AS book\_description

FROM

book b

JOIN

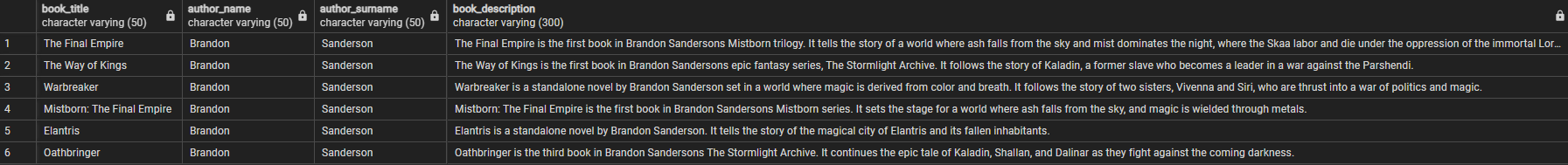
author a ON b.author\_id = a.author\_id

JOIN

book\_desc bd ON b.book\_id = bd.book\_id

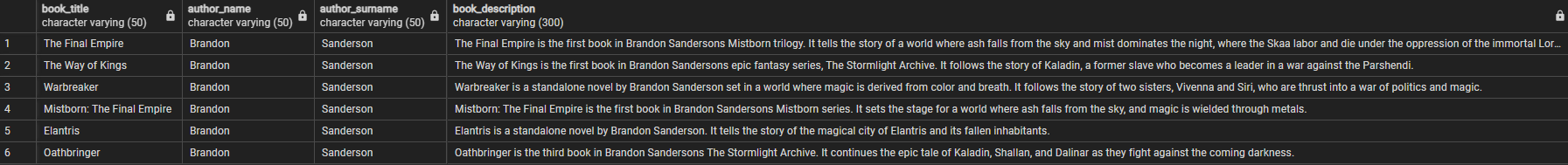
WHERE

a.aname = 'Brandon';



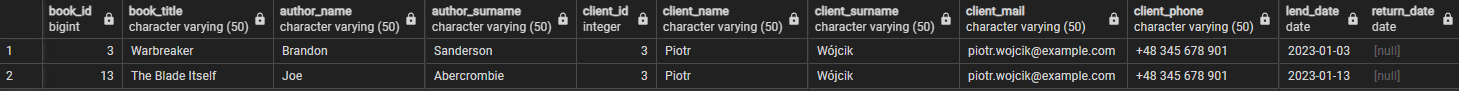
--Ten sam wynik z użyciek widoku

SELECT book\_title, author\_name, author\_surname, book\_description FROM vw\_books\_by WHERE author\_name = 'Brandon';

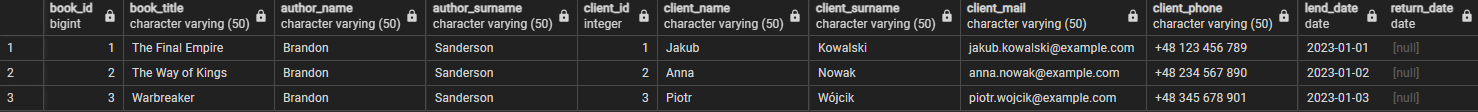


--Użycie innego widoku

SELECT \* FROM vw\_lenders\_by WHERE client\_name = 'Piotr';



SELECT \* FROM vw\_lenders\_by WHERE author\_name = 'Brandon';



--Dodawanie wypożyczenia

INSERT INTO lended (stock\_id,client\_id,lend\_date,return\_date) VALUES

(35,9,CURRENT\_DATE,NULL);

--Rejestracja zwrotu, wymaga dopracowania by móc to robić po tytule i kliencie

UPDATE lended

SET return\_date = CURRENT\_DATE

WHERE id\_lend = 21;