POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

Katedra
Informatyki i Automatyki
Politechnika Rzeszowska

Bazy danych

Sprawozdanie nr 4

pt.: „Baza danych sklepu Clother”

Data wykonania: 22.11.2020

Grupa: L2

Jakub Stanisławczyk

Spis treści

1. Cel pracy

Moim projektem będzie baza przykładowego sklepu internetowego o nazwie Clother. Będzie on umożliwiał między innymi utworzenie pracownika, klienta, złożenie zamówienia, dodania nowych produktów itd.

2. Przebieg Pracy

1.1. Etap 1

Określenie funkcji bazy danych i ich priorytetu

Baza danych to jeden z najważniejszych elementów współczesnych aplikacji i serwisów. W tym projekcie baza będzie służyć do przechowywania danych klientów, pracowników, produktów i ich zamówień. Dodatkowo będzie również posiadała procedury i triggery. Baza będzie znormalizowana i będzie posiadać indeksy tak aby maksymalnie zoptymalizować jej działanie.

Wybór technologii i typu bazy danych do zrealizowania projektu

Projekt zostanie utworzony z użyciem relacyjnej bazy danych. Do tego zadania wybrałem PostgreSQL. Jego wybór został podyktowany jego dużą popularnością, wydajnością i wsparciem, a także tym, że jest to darmowy system bazodanowy.

Wybór narzędzi do zrealizowania projektu

zarządzania bazą wykorzystam pgAdmin 4, który jest najpopularniejszą platformą do graficznego zarządzania bazami Postgresa. Dodatkowo zostanie wykorzystane narzędzie VisualParadigm do wstępnego zaplanowania diagramu ERD

Prezentacja przygotowanego repozytorium z opisem

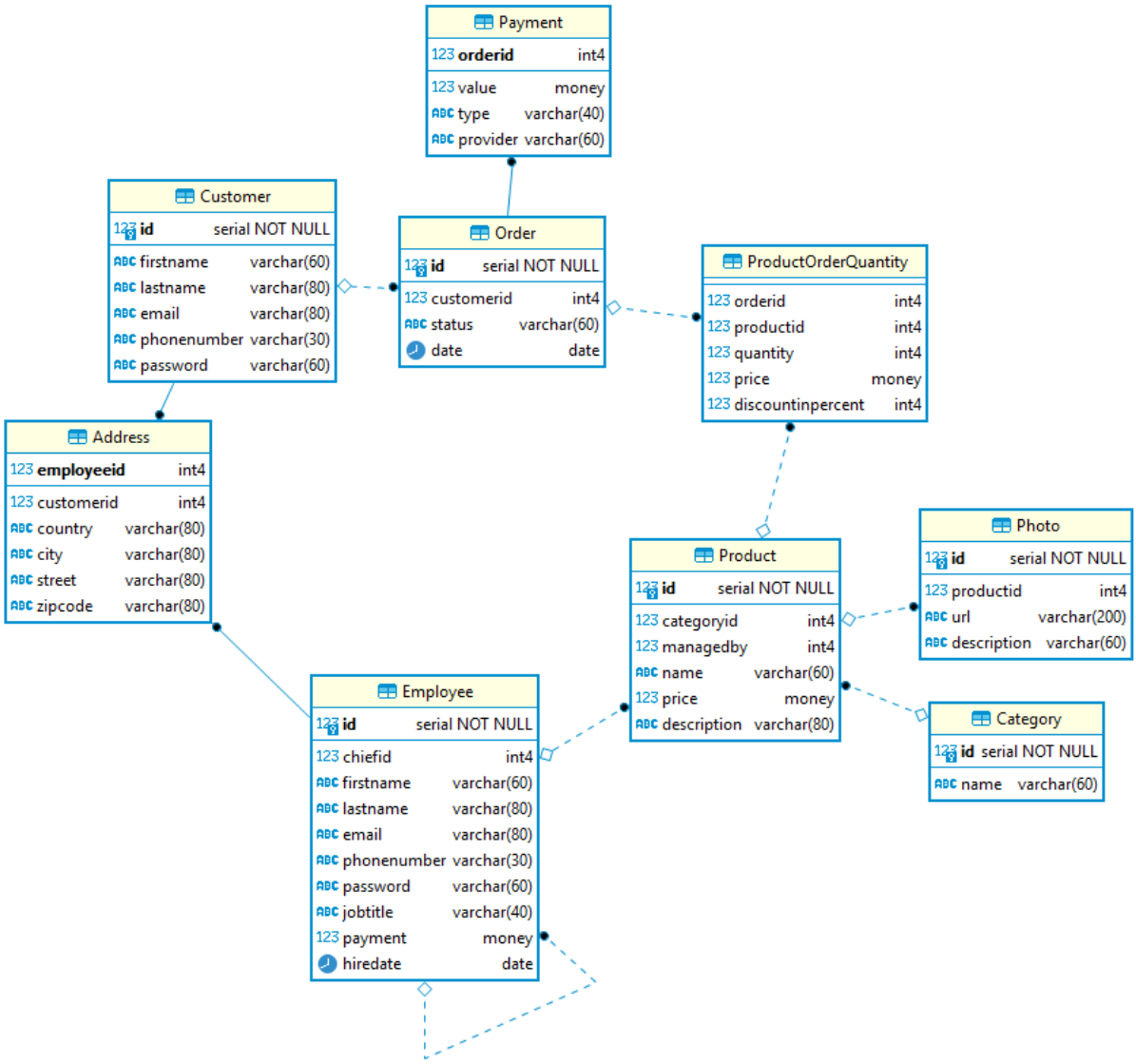
Repozytorium projektu zostało utworzone na platformie GitHub
https://github.com/jstanislawczyk/ClotherDB

1.2. Etap 2

Prezentacja diagramu bazy danych

Utworzyłem diagram ERD mojej bazy danych. Różne narzędzia wykorzystują różne notacje. Mój diagram został wygenerowany za pomocą DBeaver z notacją Idef1x

Rysunek 1 Diagram ERD



Opis tabel bazy danych i ich funkcji

- **Address** - przechowuje adresy dla kont klienta lub pracownika
- **Customer** - przechowuje konto klienta
- **Employee** - przechowuje konta pracowników, posiada relację z samą sobą w celu stworzenia hierarchii pracowników
- **Order** - przechowuje dane odnośnie zamówienia takie jak jego status czy data
- **Payment** - przechowuje dane dotyczące płatności zamówienia
- **ProductOrderQuantity** - przechowuje dane dotyczące poszczególnych produktów w zamówieniu (ich zamówiona ilość czy cena w momencie zapisu)
- **Product** - przechowuje dane produktów
- **Photo** - zdjęcia produktów w postaci URL
- **Category** - kategorie produktów

Skrypt tworzący bazę danych

https://github.com/jstanislawczyk/ClotherDB/blob/main/init.sql

Jego część znajduje się poniżej

Wycinek 1. Fragment skryptu budującego baze

```
CREATE TABLE "Category" (
    id SERIAL PRIMARY KEY NOT NULL,
    name VARCHAR(60)
);

CREATE TABLE "Product" (
    id SERIAL PRIMARY KEY,
    categoryId INT REFERENCES "Category" (id),
    managedBy INT REFERENCES "Employee" (id),
    name VARCHAR(60),
    price MONEY,
    description VARCHAR(80)
);
```

Prezentacja problemów w realizacji

Nie napotkałem większych problemów przy tworzeniu bazy i skryptu. Jedyną wartą wspomnienia informacją jest to, że domyślną formą auto inkrementacji identyfikatorów jest sekwencja. Aby ułatwić sobie jej tworzenie dla poszczególnych tabel, warto użyć typu **SERIAL**, który automatycznie stworzy wymagane dla sekwencji kroki. Podsumowując:

Wycinek 2. Przykładowy kod sekwencji

```
CREATE TABLE table_name(
    id SERIAL
);

jest równoważne

CREATE SEQUENCE table_name_id_seq;

CREATE TABLE table_name (
    id integer NOT NULL DEFAULT nextval('table_name_id_seq')
);

ALTER SEQUENCE table_name_id_seq
OWNED BY table_name.id;
```

1.3. Etap 3

Skrypt dodający dane testowe

Stworzyłem również skrypt dodające testowe dane do tabel

Wycinek 3. Przykładowy kod do tworzenia testowych danych

```
DELETE FROM "Employee";


/* Employee */
ALTER SEQUENCE "Employee_id_seq" RESTART WITH 1;

INSERT INTO "Employee"(firstname, lastname, email, phonenumber, password, jobtitle,
payment, hiredate)
VALUES ('Eugeniusz', 'Olejarczyk', 'Olejarczyk@gmail.com', '+48 111111111', MD5('1q
azXSW@'), 'CEO', 10000, CURRENT_DATE);
```

Całość można znaleźć pod adresem
https://github.com/jstanislawczyk/ClotherDB/blob/main/init-entities.sql

Przykładowe selecty

Stworzyłem przykładowe selecty aby móc łatwo przejrzeć zależne od siebie dane
https://github.com/jstanislawczyk/ClotherDB/blob/main/sample-selects.sql

Wycinek 4. Pobranie danych klienta

```
SELECT * FROM "Customer" AS customer

WHERE customer.id = 2;
```

Wycinek 5. Pobranie danych pracownika

```
SELECT * FROM "Employee" AS employee

WHERE employee.id = 2;
```

Wycinek 6. Pobranie danych produktu

```
SELECT * FROM "Product" AS product

WHERE product.id = 3;
```

Wycinek 7. Pobranie szefa pracownika

```
SELECT chief.* FROM "Employee" AS employee

INNER JOIN "Employee" AS chief
ON employee.chiefId = chief.id
WHERE employee.id = 5;
```

Wycinek 8. Pobranie podwładnych

```
SELECT product.* FROM "Product" AS product

INNER JOIN "Category" AS category
ON product.categoryId = category.id
WHERE category.name = 'Buty';
```

Wycinek 9. Pobranie produktów w kategorii

```
SELECT employee.* FROM "Employee" AS chief

INNER JOIN "Employee" AS employee
ON chief.id = employee.chiefId
WHERE chief.id = 2;
```

Wycinek 10. Pobranie produktów zarządzanych przez pracownika

```
SELECT product.* FROM "Product" AS product

INNER JOIN "Employee" AS employee
ON employee.id = product.managedBy
WHERE employee.id = 4;
```

Wycinek 11. Pobranie zamówień klienta

```
SELECT orders.* FROM "Customer" AS customer

INNER JOIN "Order" AS orders
ON customer.id = orders.customerId
WHERE customer.id = 3;
```

Wycinek 12. Pobranie płatności klienta

```
SELECT payment.* FROM "Customer" AS customer

INNER JOIN "Order" AS orders
ON customer.id = orders.customerId
INNER JOIN "Payment" AS payment
ON payment.orderId = orders.id
WHERE customer.id = 3;
```

Wycinek 13. Pobranie elementów zamówienia

```
SELECT product.name, productOrderQuantity.quantity,
productOrderQuantity.price FROM "Order" AS orders

INNER JOIN "ProductOrderQuantity" AS productOrderQuantity
ON orders.id = productOrderQuantity.orderId
INNER JOIN "Product" AS product
ON product.id = productOrderQuantity.productId
WHERE orders.id = 2;
```

Funkcje bazodanowe

Dodałem również funkcje i procedury składowane

https://github.com/jstanislawczyk/ClotherDB/blob/main/procedures.sql

Poniżej zaprezentuje wybrane z nich.

Wycinek 14. Stworzenie pracownika

```
CREATE OR REPLACE PROCEDURE createEmployee(
   firstname varchar, lastname varchar, email varchar, phonenumber varchar, password
varchar, jobtitle varchar, payment bigint
)
AS $$
    BEGIN
        INSERT INTO "Employee"(firstname, lastname, email, phonenumber, password,
               jobtitle, payment, hiredate)
        VALUES (firstname, lastname, email, phonenumber, MD5(password), jobtitle,
               payment, CURRENT_DATE);
    END
$$ LANGUAGE plpgsql;

call createEmployee('Eugeniusz', 'Olejarczyk', 'Olejarczyk@gmail.com', '123456789',
'1qazXSW@', 'CEO', 3000);
```

Powyższa procedura jako argumenty przyjmuje poszczególne pola użytkownika. Następnie przekazuje je do INSERT, który dodaje na ich podstawie pracownika do tabeli Employee

Wycinek 15. Zmienienie danych pracownika

```
CREATE OR REPLACE PROCEDURE updateEmployee(
   employeeId int, newFirstname varchar, newLastname varchar, newPhoneNumber varchar,
newJobTitle varchar, newPayment bigint
)
AS $$
    BEGIN
        UPDATE "Employee"
        SET firstname = newFirstname, lastname = newLastname,
            phonenumber = newPhoneNumber,
            jobTitle = newJobTitle, payment = newPayment
        WHERE id = employeeId;
    END
$$ LANGUAGE plpgsql;

call updateEmployee(11, 'TEST1', 'TEST2', '123456789', 'INTERN', 1111);
```

Kolejna procedura zajmuje się zmianą danych pracownika. Działa podobnie do poprzedniej. Podajemy argumenty do zmiany i za pomocą UPDATE ustawiamy nowe dane

Wycinek 16. Stworzenie zamówienia

```
CREATE OR REPLACE PROCEDURE createOrder(customerId int)
AS $$
    BEGIN
        INSERT INTO "Order"(customerid, status, date)
        VALUES (customerId, 'CREATED', CURRENT_DATE);
    END
$$ LANGUAGE plpgsql;

call createOrder(1);
```

W analogiczny sposób do poprzednich, działa powyższa procedura dodająca zamówienie

Wycinek 17. Opłacenie zamówienia

```
CREATE OR REPLACE PROCEDURE makePaymentForOrder(
       orderId int, value bigint, type varchar, provider varchar
)
AS $$
    BEGIN
        UPDATE "Order" AS orders
        SET status = 'PAID'
        WHERE orders.id = orderId;

        INSERT INTO "Payment"(orderid, value, type, provider)
        VALUES (orderId, value, type, provider);
    END
$$ LANGUAGE plpgsql;

call makePaymentForOrder(17, 999, 'Bank', 'Pekao');
```

Ta natomiast zawiera w sobie dwa kroki. Podajemy numer zamówienia, które jeszcze nie zostało opłacone. Podajemy również dane takie jak ilość pieniędzy czy typ płatności. Procedura najpierw zmieni status zamówienia a następnie stworzy wpis o płatności w tabeli „Payments”

Wycinek 18. Funkcja wyciągające dane klienta po ID

```
CREATE FUNCTION selectCustomerById(customerId int)
RETURNS TABLE (
	id int, firstName varchar, lastName varchar, email varchar,
	phoneNumber varchar, password varchar
)
AS
$$
	BEGIN
		RETURN QUERY
		SELECT * FROM "Customer" AS customer
		WHERE customer.id = customerId;
	END
$$ LANGUAGE plpgsql;
```

Dodałem również proste funkcje. Powyższa pobiera id użytkownika i na podstawie tego pobiera za pomocą SELECT dane użytkownika z danym id. Następnie zwraca je w postaci rekordu tabelki z określonymi polami

3. Wnioski

Baza danych jest jednym z najważniejszych elementów współczesnych aplikacji. Jej odpowiednie zaprojektowanie może znacząco wpłynąć na jej wydajność i skalowalność. Wybór odpowiedniego systemu bazodanowego również może znacznie wpłynąć na odpowiednie działanie programu.

4. Spis rysunków i tabel