

Relacyjne systemy zarządzania bazami danych

Jan Łukomski 291089

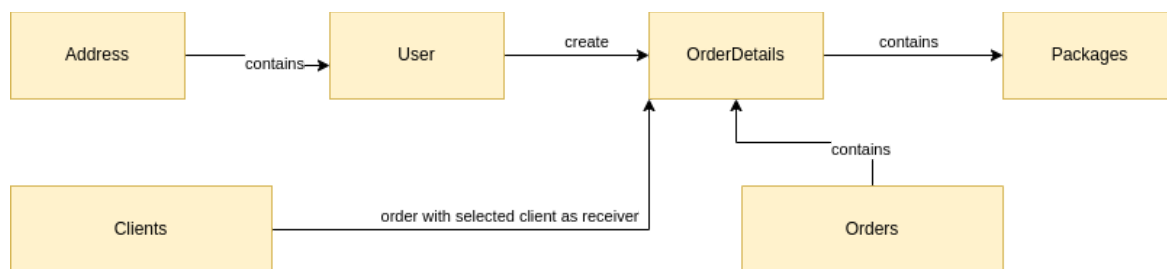
Informatyka Stosowana,
Studia Magisterskie,
Wydział Elektryczny

Sprawozdanie z projektu 1

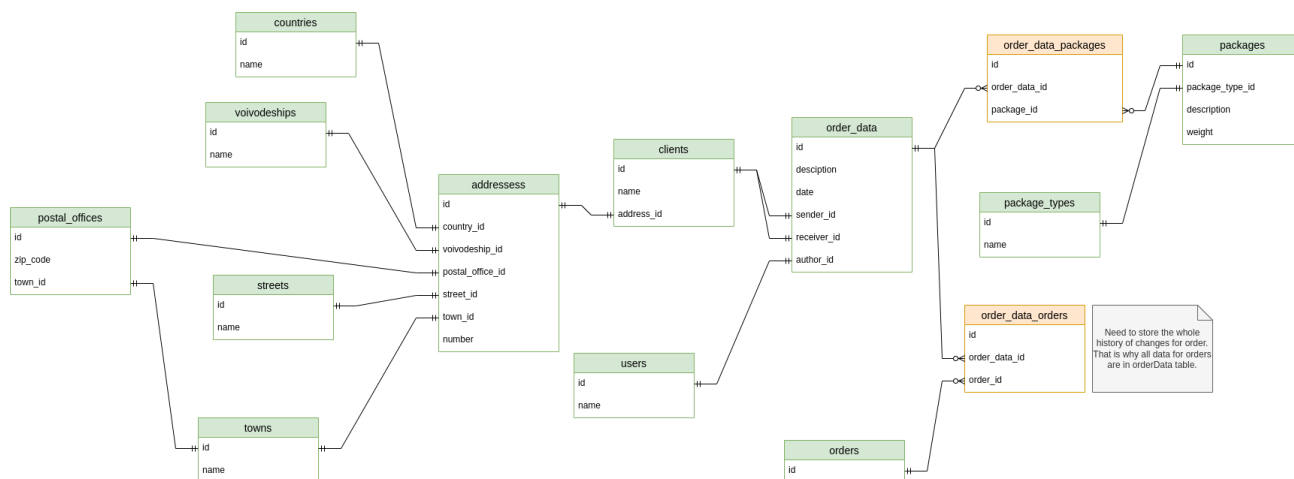
Przygotowanie bazy danych

Wybrana została baza danych Postgres SQL.

W pierwszym kroku został zrobiony schemat bazy ERD. Jak się okazało wymagał on późniejszej modyfikacji, ale i tak był to przydatny krok, który pomógł przygotować schemat logiczny bazy.

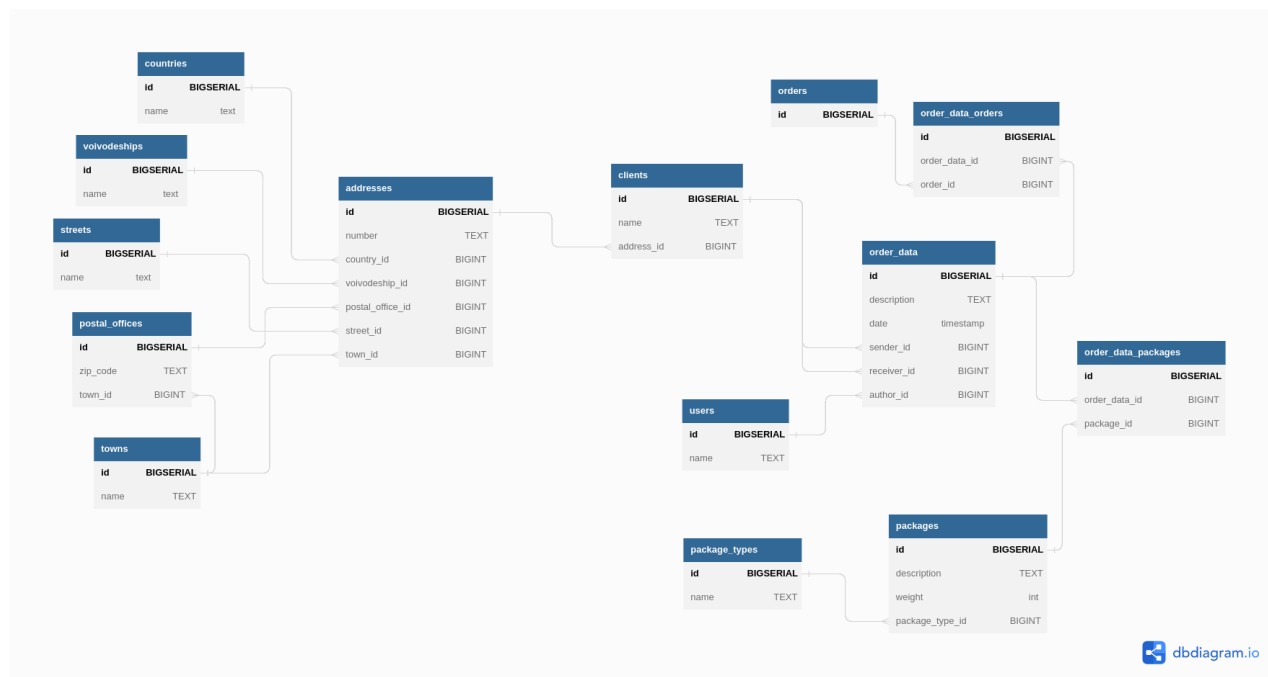


W kolejnym kroku został przygotowany logiczny schemat bazy w postaci 3 normalnej.



Pominięte w nim zostały typy danych oraz oznaczenia kluczy głównych (lub obcych), żeby nie zaciemniały obrazu nadmiarem informacji.

Pełen schemat przedstawiam poniżej.



Implementacja

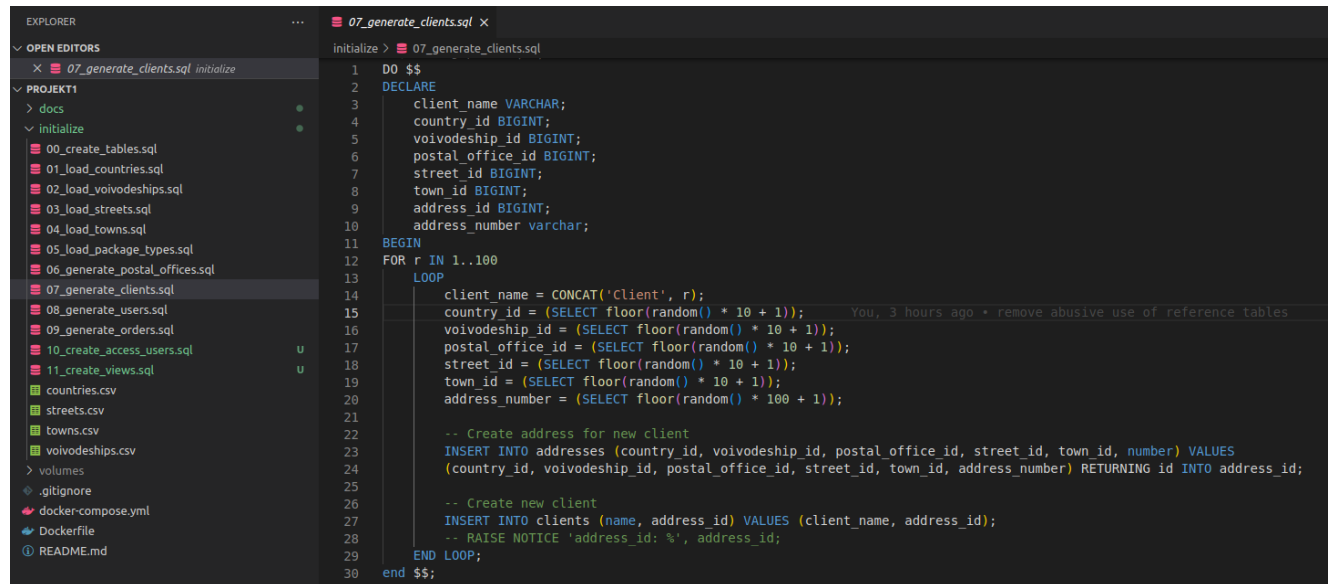
Po narysowaniu logicznego schematu bazy rozpoczęto implementację rozwiązania. Postawiony został silnik Postgres SQL na dockerze, następnie zostały napisane pliki inicjujące bazę danych wraz z wypełnieniem przykładowymi danymi. Takie rozwiązanie pozwoliło na wygodne poprawianie bazy. Gdy podczas testów cokolwiek zostało uszkodzonego - jedyne co trzeba było zrobić to usunąć pliki bazy i uruchomić skrypt od początku.

```
initialize
00_create_tables.sql
01_load_countries.sql
02_load_voivodeships.sql
03_load_streets.sql
04_load_towns.sql
05_load_package_types.sql
06_generate_postal_offices.sql
07_generate_clients.sql
08_generate_users.sql
09_generate_orders.sql
10_create_access_users.sql
11_create_views.sql
countries.csv
streets.csv
towns.csv
voivodeships.csv

2 -- Model tables
3 --
4 DROP TABLE IF EXISTS countries;
5 CREATE TABLE countries (
6   id BIGSERIAL PRIMARY KEY,
7   name text
8 );
9
10 DROP TABLE IF EXISTS voivodeships;
11 CREATE TABLE voivodeships (
12   id BIGSERIAL PRIMARY KEY,
13   name text
14 );
15
16 DROP TABLE IF EXISTS streets;
17 CREATE TABLE streets (
18   id BIGSERIAL PRIMARY KEY,
19   name text
20 );
21
```

Skrypty inicjalizujące umieszczone zostały w jednym folderze i są uruchamiane w kolejności alfabetycznej, dlatego zostało zadbane o zachowanie jasnej kolejności wykonywania.

Na początku została stworzona struktura tabel. Dane takie jak nazwy krajów, ulic, miast czy województw zaciągnięte zostały z dostępnych w internecie baz danych. Widoczne są pliki CSV z danymi, które są zaciągane do bazy. Następnie kolejne dane zostały wygenerowane losowymi danymi.

The screenshot shows a code editor with a dark theme. On the left, there is an 'EXPLORER' sidebar showing a project structure. The 'PROJEKT1' folder is expanded, showing subfolders 'docs' and 'initialize'. The 'initialize' folder contains several SQL files, with '07_generate_clients.sql' selected. The main editor area shows the content of '07_generate_clients.sql'. The script starts with a 'DO \$\$' block, followed by a 'DECLARE' section with variables for client_name, country_id, voivodeship_id, postal_office_id, street_id, town_id, address_id, and address_number. A 'BEGIN' block follows, containing a 'FOR' loop that iterates from 1 to 100. Inside the loop, there is a 'LOOP' section that generates random values for the variables and inserts them into the 'addresses' table. After the loop, there is an 'INSERT INTO clients' statement that inserts the generated client data into the 'clients' table. The script ends with 'END LOOP;' and 'end \$\$;'.

```
1 DO $$
2 DECLARE
3     client_name VARCHAR;
4     country_id BIGINT;
5     voivodeship_id BIGINT;
6     postal_office_id BIGINT;
7     street_id BIGINT;
8     town_id BIGINT;
9     address_id BIGINT;
10    address_number varchar;
11 BEGIN
12 FOR r IN 1..100
13 LOOP
14     client_name = CONCAT('Client', r);
15     country_id = (SELECT floor(random() * 10 + 1));
16     voivodeship_id = (SELECT floor(random() * 10 + 1));
17     postal_office_id = (SELECT floor(random() * 10 + 1));
18     street_id = (SELECT floor(random() * 10 + 1));
19     town_id = (SELECT floor(random() * 10 + 1));
20     address_number = (SELECT floor(random() * 100 + 1));
21
22     -- Create address for new client
23     INSERT INTO addresses (country_id, voivodeship_id, postal_office_id, street_id, town_id, number) VALUES
24     (country_id, voivodeship_id, postal_office_id, street_id, town_id, address_number) RETURNING id INTO address_id;
25
26     -- Create new client
27     INSERT INTO clients (name, address_id) VALUES (client_name, address_id);
28     -- RAISE NOTICE 'address_id: %', address_id;
29 END LOOP;
30 end $$;
```

Użytkownicy oraz uprawnienia

Po wypełnieniu tabel danymi stworzeni zostali użytkownicy bazodanowi. Przydzieleni zostali oni do grup z odpowiednimi uprawnieniami.

```

OPEN EDITORS
10_create_access_users.sql initialize U
PROJECT1
  docs
  initialize
    00_create_tables.sql
    01_load_countries.sql
    02_load_voivodeships.sql
    03_load_streets.sql
    04_load_towns.sql
    05_load_package_types.sql
    06_generate_postal_offices.sql
    07_generate_clients.sql
    08_generate_users.sql
    09_generate_orders.sql
    10_create_access_users.sql U
    11_create_views.sql U
  countries.csv
  streets.csv
  towns.csv
  voivodeships.csv
  volumes
  .gitignore
  docker-compose.yml
  Dockerfile
  README.md
initialize > 10_create_access_users.sql
1  -- Create users
2  CREATE USER joe;
3  CREATE USER olivia;
4  CREATE USER emma;
5  CREATE USER oliver;
6
7  -- Create groups
8  CREATE GROUP observers;
9  CREATE GROUP office_workers;
10 CREATE GROUP managers;
11
12 -- Grant privileges for groups
13 DO
14 $$
15 DECLARE
16   rec record;
17 BEGIN
18   FOR rec IN
19     SELECT *
20     FROM   pg_tables
21     WHERE  schemaname = 'public'
22     ORDER BY tablename
23   LOOP
24     EXECUTE 'GRANT SELECT ON ' || rec.tablename || ' TO observers';
25     EXECUTE 'GRANT SELECT ON ' || rec.tablename || ' TO office_workers';
26     EXECUTE 'GRANT SELECT ON ' || rec.tablename || ' TO managers';
27
28     EXECUTE 'GRANT INSERT ON ' || rec.tablename || ' TO office_workers';
29     EXECUTE 'GRANT INSERT ON ' || rec.tablename || ' TO managers';
30
31     EXECUTE 'GRANT UPDATE ON ' || rec.tablename || ' TO managers';
32   END LOOP;
33 END
34 $$;
35
36 -- Grant users to groups
37 ALTER GROUP observers ADD USER joe, olivia;
38 ALTER GROUP office_workers ADD USER emma;
39 ALTER GROUP managers ADD USER oliver;

```

Baza została przetestowana pod względem dostępu do operacji. Zamieszczam poniżej przykład gdy użytkownik Emma, która jest w grupie office_worker próbuje edytować tabele. Operacja nie powiodła się, ponieważ grupa office_worker nie ma uprawnień do edycji (tylko czytanie i dodawanie danych).

```

vrl=> \du

```

Role name	List of roles Attributes	Member of
emma		{office_workers}
joe		{observers}
managers	Cannot login	{}
observers	Cannot login	{}
office_workers	Cannot login	{}
oliver		{managers}
olivia		{observers}
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{}

```

vrl=> select current_user;
current_user
-----
emma
(1 row)

vrl=> update towns set name='Radom' where id=1;
ERROR: permission denied for table towns

```

Widoki (Perspektywy)

Po użytkownikach zostały napisane przykładowe widoki. Widoki mają uzasadnienie biznesowe. Pozwalają wyciągnąć aktualne dane dotyczące zamówień. Schemat bazy został tak przygotowany, żeby pamiętać całą historię zmian w zamówieniu (przy odpowiednim korzystaniu z bazy). Powoduje to

jednak konieczność wybrania dla każdego zamówienia najnowszego elementu z historii powodując, że zapytanie nie jest trywialne.

Poniższe zapytanie wybiera najbardziej aktualne połączenie tabeli “orders” z tabelą “order_data”.

```
vrl=# select * from current_order_data_orders;
```

order_id	order_data_max_id
1	10
2	20
3	30
4	40
5	50
6	60
7	70
8	80

Poniższe zapytanie pokazuje informacje o aktualnych zamówieniach oraz paczkach które są w zamówieniu.

```
vrl=# select * from current_orders;
```

order_id	order_description	package_description	package_type_name	package_weight
1	Order 1 description 10	Package descriptions 1 package_r is 10	Termika C	41
1	Order 1 description 10	Package descriptions 1 package_r is 9	Karton 100x300	86
1	Order 1 description 10	Package descriptions 1 package_r is 8	Karton 40x20	3
1	Order 1 description 10	Package descriptions 1 package_r is 7	Termika C	37
1	Order 1 description 10	Package descriptions 1 package_r is 6	Karton 100x300	76
1	Order 1 description 10	Package descriptions 1 package_r is 5	Termika D	5
1	Order 1 description 10	Package descriptions 1 package_r is 4	Karton 20x20	97
1	Order 1 description 10	Package descriptions 1 package_r is 3	Termika A	46
1	Order 1 description 10	Package descriptions 1 package_r is 2	Termika B	75
1	Order 1 description 10	Package descriptions 1 package_r is 1	Karton 20x20	10
2	Order 2 description 10	Package descriptions 2 package_r is 10	Termika D	19
2	Order 2 description 10	Package descriptions 2 package_r is 9	Termika A	33
2	Order 2 description 10	Package descriptions 2 package_r is 8	Karton 30x20	40
2	Order 2 description 10	Package descriptions 2 package_r is 7	Karton 30x20	70
2	Order 2 description 10	Package descriptions 2 package_r is 6	Termika C	18
2	Order 2 description 10	Package descriptions 2 package_r is 5	Karton 100x300	46
2	Order 2 description 10	Package descriptions 2 package_r is 4	Paleta 40x40	5
2	Order 2 description 10	Package descriptions 2 package_r is 3	Termika B	63

Indeksy

Wygenerowane zostało 10 milionów rekordów tabeli “order_data”. Bez indeksu zapytanie wyciągnięcia pierwszego elementu po dacie zajmowało około 1400 ms. Po dodaniu indeksu na dacie czas przetwarzania zapytania zmniejszył się do 1.5 ms.

```
vrl=# select * from order_data order by date limit 1;
```

id	description	date	sender_id	receiver_id	author_id
5423608	Order 54237 description 8	2014-01-10 20:00:00.032789	1	4	6

(1 row)

```
Time: 1389.224 ms (00:01.389)
vrl=# CREATE INDEX vrl_index ON order_data(date);
CREATE INDEX
Time: 6434.256 ms (00:06.434)
vrl=# select * from order_data order by date limit 1;
```

id	description	date	sender_id	receiver_id	author_id
5423608	Order 54237 description 8	2014-01-10 20:00:00.032789	1	4	6

(1 row)

```
Time: 1.646 ms
```