



WYDZIAŁ  
**ELEKTROTECHNIKI  
I INFORMATYKI**  
POLITECHNIKI RZESZOWSKIEJ

# **Bazy danych projekt**

**System zgłaszania usterek miejskich oparty na aplikacji mobilnej**

**Piotr Zaradkiewicz 2EF-ZI  
nr indeksu 117381**

## **Spis Treści**

- 1. Temat projektu**
- 2. Opis tematu**
- 3. Zagadnienia związane z tematem**
- 4. Funkcje bazy danych**
- 5. Wybór technologii i bazy danych do zrealizowania projektu**
- 6. Wybór narzędzi do zrealizowania projektu**
- 7. Prezentacja przygotowanego repozytorium wraz z opisem**
- 8. Prezentacja przygotowanego środowiska**
- 9. Prezentacja diagramu bazy danych**
- 10. Opis tabel bazy danych i ich funkcji**
- 11. Prezentacja wykonania bazy danych**
- 12. Proste zapytania SQL**
- 13. Użytkownicy bazy danych oraz ich role**
- 14. Zaawansowane zapytania SQL (dodawanie, aktualizacja, selekcja danych)**
- 15. Funkcje i procedury obsługujące Bazę danych**
- 16. Zarządzanie bazą danych (backup, restore)**
- 17. Wariant testowy i na produkcji**
- 18. Komunikacja z językiem programowania**
- 19. Przykład zastosowania ORM**

**1. Temat projektu:** System zgłaszania usterek miejskich oparty na aplikacji mobilnej

**2. Opis tematu:**

Opracowanie kompleksowego systemu mobilnego do zgłaszania usterek miejskich przez mieszkańców Rzeszowa, umożliwiającego łatwe przekazywanie informacji o problemach z lokalizacją GPS i zdjęciami.

**3. Zagadnienia związane z tematem**

Projektowana baza danych będzie przechowywała informacje o użytkownikach aplikacji, informacje o zgłoszonych usterekach takie jak: typ usterki, opis usterki, lokalizacja wraz ze zdjęciem poglądowym, priorytet oraz status. Zalogowany użytkownik poprzez aplikację mobilną będzie miał możliwość zgłoszenia nowej usterki oraz będzie mógł wyszukiwać i przeglądać usterki znajdujące się w bazie.

Niezałogowani użytkownicy posiadający aplikację będą mogli wyszukiwać i wyświetlać istniejące usterki ale nie będą mogli dodawać nowych.

Technicy oraz osoby administrujące bazę będą mogły dodawać komentarze i przypisywać priorytety zgłoszonym usterek oraz zmieniać ich status.

Na podstawie zgłoszonych usterek będą mogły być tworzone rankingi najaktywniejszych użytkowników poprzez liczbę zgłoszonych przez nich usterek oraz ilość zgłoszeń, które doprowadziły do rozwiązania problemu.

#### 4. Funkcje bazy danych

Baza danych wstępnie będzie posiadać następujące funkcje:

- **Raportowanie usterek** (zalogowani użytkownicy będą mogli przez aplikację mobilną zgłosić usterki wypełniając formularz)
- **Wyszukiwanie i wyświetlanie** (wszyscy użytkownicy aplikacji mogą przeszukiwać i przeglądać bazę zgłoszonych usterek)
- **Śledzenie statusu** (status nadany usterek taki jak np. „nowa usterka” , „zgłoszona” „naprawiona” pozwoli na śledzenie zmian)
- **Priorytetyzacja** (Osoby zajmujące się administrowaniem systemu powinny mieć możliwość nadania priorytetu danej usterek tak aby te najpoważniejsze mogły być naprawiane w pierwszej kolejności)
- **Informacje zwrotne** (osoby zajmujące się zlecaniem napraw bądź saymi naprawami powinny mieć możliwość dodawania informacji na temat danej usterek – komentarza na temat naprawy lub informacji o braku możliwości naprawy)

- **Kategoryzacja** (Wprowadzenie kategorii usterek pozwoli na łatwiejsze ich wyszukiwanie i przeglądanie)
- **Przechowywanie informacji o użytkownikach** (pozwoli na stworzenie rankingów najaktywniejszych użytkowników oraz zgłoszonych przez nich usterek o różnych statusach)

## 5. Wybór technologii i bazy danych do zrealizowania projektu

Postanowiłem w projekcie wybrać jedną z relacyjnych baz danych. Spośród wielu możliwych systemów zarządzania relacyjnymi badaniami danych wybrałem PostgreSQL.

Mój wybór był motywowany chęcią zapoznania się z tą technologią.

## 6. Wybór narzędzi do zrealizowania projektu

### Serwer

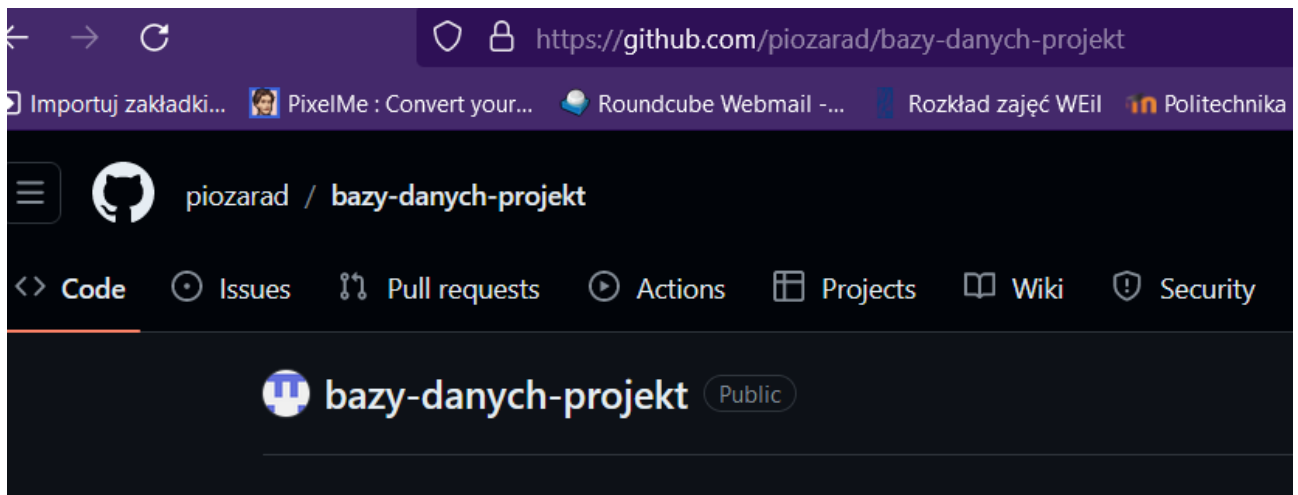
Baza danych będzie zainstalowana na systemie linux Ubuntu 22.04

### Klient – pgAdmin 4

PgAdmin to popularne i wszechstronne narzędzie do zarządzania bazami danych PostgreSQL. Pozwala na wykonywanie zapytań, zarządzanie użytkownikami, tabelami, indeksami, schematami itp

## 7. Prezentacja przygotowanego repozytorium wraz z opisem

Do celów projektowych stworzyłem publiczne repozytorium na githubie pod adresem <https://github.com/piozarad/bazy-danych-projekt>



rys. 1 repozyturyum na githubie

## 8. Prezentacja przygotowanego środowiska

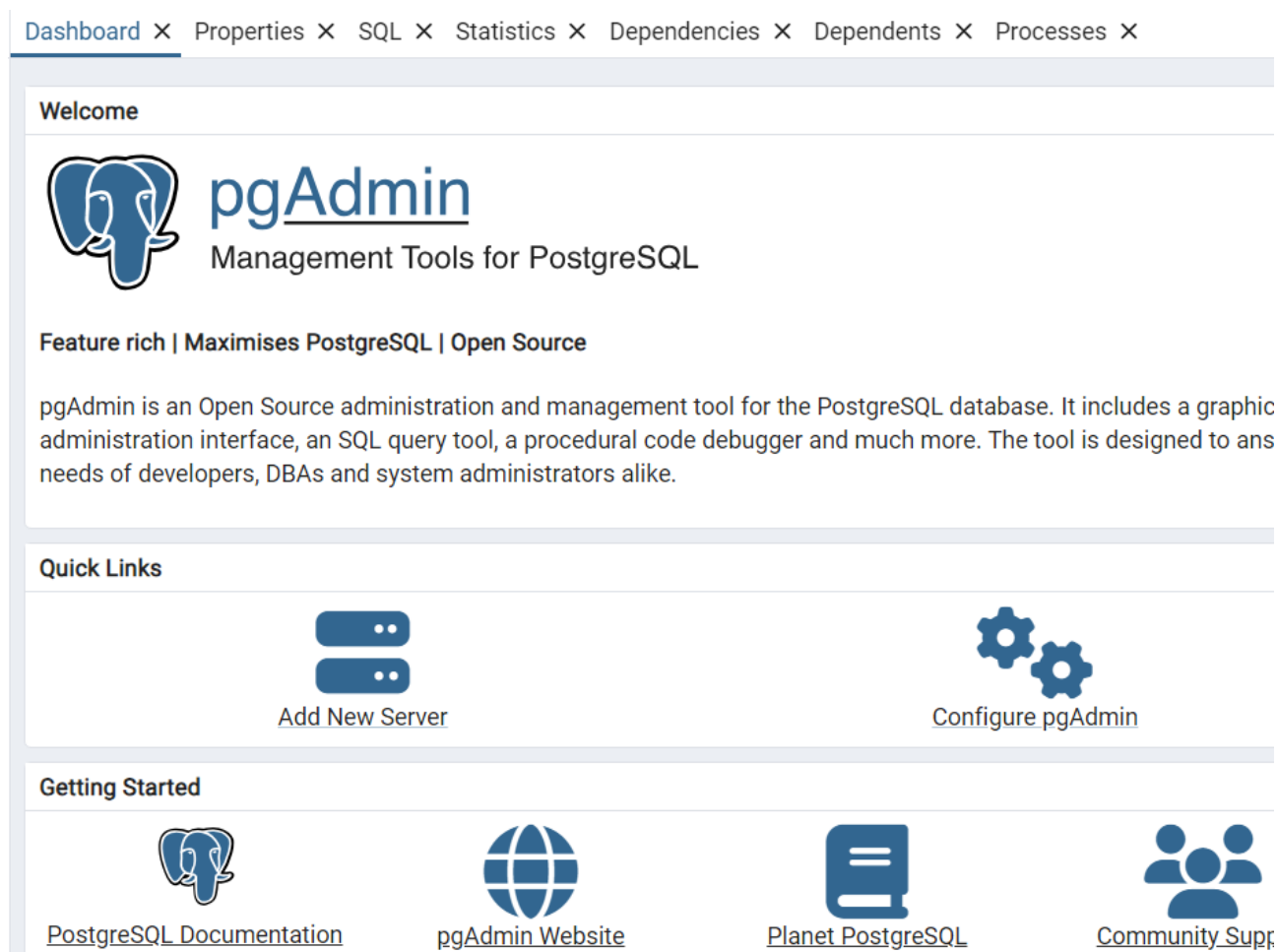
A screenshot of a terminal window with a dark background. The prompt is 'piotrz@piotrz-VirtualBox: ~'. The command 'lsb\_release -a' has been entered and executed. The output is as follows:

```
piotrz@piotrz-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.1 LTS
Release:        22.04
Codename:       jammy
piotrz@piotrz-VirtualBox:~$
```

rys. 2 prezentacja środowiska – Ubuntu 22

```
piotr@piotr-VirtualBox: ~  
piotr@piotr-VirtualBox:~$ service postgresql status  
● postgresql.service - PostgreSQL RDBMS  
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor pre  
   Active: active (exited) since Sun 2024-05-12 08:09:14 CEST; 12min ago  
   Process: 517 ExecStart=/bin/true (code=exited, status=0/SUCCESS)  
   Main PID: 517 (code=exited, status=0/SUCCESS)  
   CPU: 1ms  
  
maj 12 08:09:14 piotr-VirtualBox systemd[1]: Starting PostgreSQL RDBMS...  
maj 12 08:09:14 piotr-VirtualBox systemd[1]: Finished PostgreSQL RDBMS.  
lines 1-9/9 (END)
```

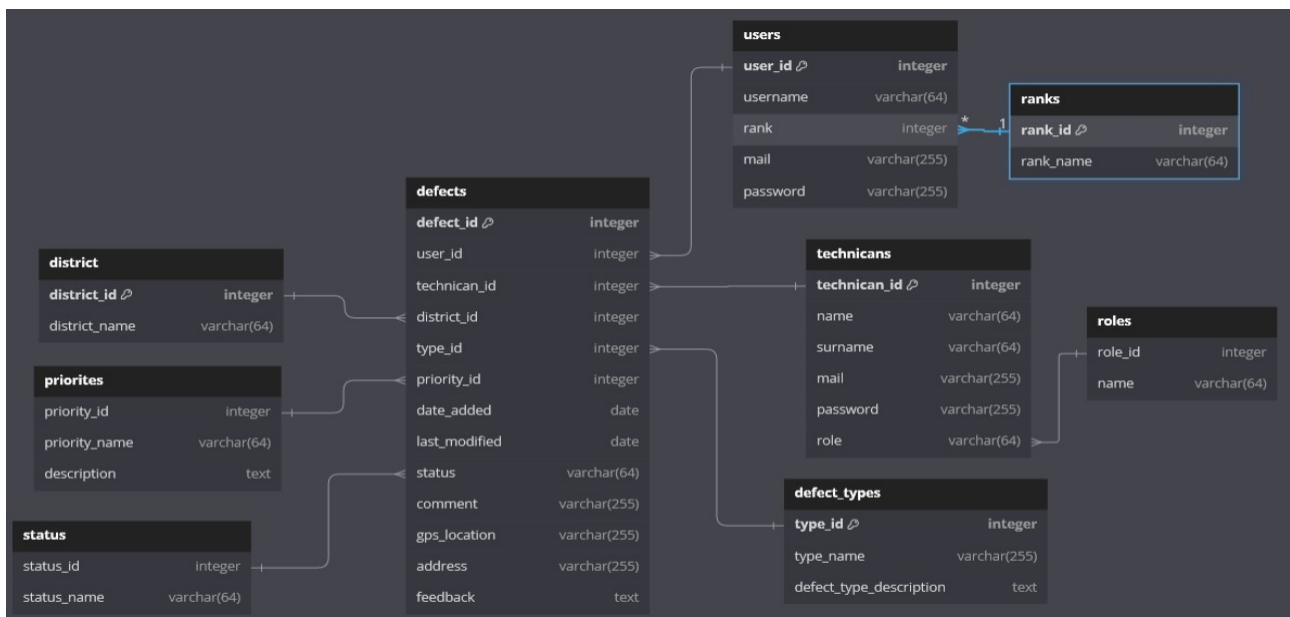
rys. 3 prezentacja środowiska – PostgreSQL RDBMS



rys. 4 prezentacja środowiska – pgAdmin

## 9. Prezentacja diagramu ERD bazy danych

Diagram ERD (Entity-Relationship Diagram) to graficzna reprezentacja struktury bazy danych. Służy do modelowania danych i relacji między nimi, co pomaga w projektowaniu baz danych.



Rys 5. Diagram ERD dla projektu bazy danych



## 10. Opis tabel bazy danych i ich funkcji

**Tabela defect** – jest główną tabelą w bazie danych. Służy do przechowywania wszystkich istotnych informacji na temat raportowanych usterek takich jak:

- osoba zgłaszająca usterkę
- przypisany do usterki pracownik
- id dzielnicy ( do grupowania usterek względem dzielnic miasta )
- typ usterki (do grupowania względem typów nr usterki związane z jezdnią, chodnikiem, uszkodzenia budynków zagrażające przechodniom itp. )
- data dodania
- data modyfikacji (do rejestracji zmian danych usterki zwłaszcza jej statusu i feedbacku od osób z administracji)
- priorytet usterki
- status usterki modyfikowany przez technika (początkowo będzie to ‘zgłoszona’, później np. ‘w trakcie naprawy’, ‘odrzucona’, ‘naprawiona’ itp.)
- komentarz osoby zgłaszającej dotyczący istotnych z jego punktu widzenia informacji na temat tej usterki
- adres
- lokalizacja gps
- feedback będzie sposobem komunikacji osób zgłaszających z osobami zajmującymi się naprawą usterek np. w przypadku jeżeli naprawa usterki będzie z jakichś powodów utrudniona, niemożliwa lub zajmie dużo czasu to będzie można tu zawrzeć te informacje dla wszystkich osób przeglądających system

**Tabela user** – służy do przechowywania informacji na temat zalogowanych użytkowników w systemie. Tylko zalogowani użytkownicy będą mieli możliwość zgłaszania usterek (osoby niezalogowane będą mogły jedynie przeglądać dodane w

aplikacji usterki). Zawiera pola podstawowe takie jak id nazwę użytkownika, mail, hasło oraz rangę.

**Tabela technicians** – jest to tabela, która będzie przechowywać informację na temat pracowników zajmujących się pracą nad usterekami. Będzie zawierać takie informacje jak: id, imię nazwisko, hasło do logowania w systemie, mail i rola

**Tabela district** – tabela będzie przechowywała dzielnice miasta. Pomaga on grupować usterki pod względem miejsc występowania. W trakcie działania systemu może się okazać, że do miasta zostaną dołączone nowe tereny. Posiada ona tylko dwa pola: id oraz nazwę dzielnicy.

**Tabela defect types** – jest to tabela przechowująca typy zgłaszanych usterek. Użytkownik będzie mógł przy dodawaniu nowej usterki w formularzu wybrać typ zgłaszanej usterki. Są tutaj następujące pola: id, nazwa oraz opis pomagający użytkownikom na prawidłowe przyporządkowanie typu usterki do zgłaszanego problemu.

**Tabela rank** – tabela rang użytkowników ma służyć do nadawania użytkownikom rang bazujących na ilości zgłoszonych usterek i ewentualnie dodatkowych kryteriów takich jak ilość naprawionych usterek czy ilość zgłoszonych usterek o danym typie. Rangi mają za zadanie zachęcenie użytkowników do większej aktywności w aplikacji. Zawiera dwa pola: id oraz nazwa rangi.

**Tabela roles** – przechowuje nazwy ról jakie pełnią poszczególni technicy. Została dodana aby nie przechowywać zbyt dużej ilości danych typu varchar w tabeli **defects**.

**Tabela status** – przechowuje dane na temat statusu danej usterki. Mają one za zadanie przekazywanie informacji o statusie zgłoszenia i późniejszej ewentualnej

naprawy danej usterki. Statusy będą mogły być zmieniane przez techników. Dodana usterka będzie miała nadany status początkowy 'niepotwierdzony'. Podobnie jak tabela wyżej ta została dodana aby zmniejszyć ilość danych typu varchar w bazie danych.

**Tabela priorities** - tabela zawierająca priorytety raportowanych usterek. Priorytet jest wybierany przez użytkownika podczas raportowania usterki. Służy ona do priorytetyzacji raportów.

### Relacje:

**priorities** jest w relacji jeden do wielu z tabelą **defects**.

**defect\_types** jest w relacji jeden do wielu z tabelą **defects**.

**users** jest w relacji jeden do wielu z encją **defects**.

**priorities** jest w relacji jeden do wielu z encją **defects**.

**technican** jest w relacji jeden do wielu z encją **defects**.

**district** jest w relacji jeden do wielu z encją **defects**.

**status** jest w relacji jeden do wielu z encją **defects**.

**rank** jest w relacji jeden do wielu z encją **user**.

**technican** jest w relacji jeden do wielu z encją **roles**

**status** jest w relacji jeden do wielu z encją **defects**

## 11. Prezentacja wykonania bazy danych

Kod tworzący tabele:

```
CREATE TABLE priorities (  
  priority_id SERIAL PRIMARY KEY,
```

```
priority_name VARCHAR(64),  
description TEXT  
);
```

```
CREATE TABLE status (  
    status_id SERIAL PRIMARY KEY,  
    status_name VARCHAR(64)  
  
);
```

```
CREATE TABLE defect_types (  
    type_id SERIAL PRIMARY KEY,  
    type_name VARCHAR(255),  
    defect_type_description TEXT  
);
```

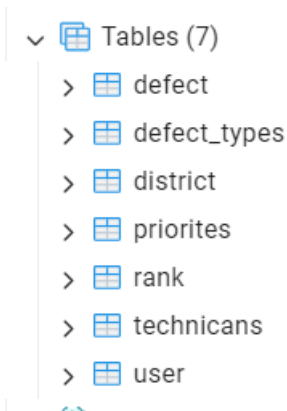
```
CREATE TABLE technicians (  
    technician_id SERIAL PRIMARY KEY,  
    name VARCHAR(64),  
    surname VARCHAR(64),  
    mail VARCHAR(255),  
    password VARCHAR(255),  
    role INTEGER  
);
```

```
CREATE TABLE ranks (  
    rank_id SERIAL PRIMARY KEY,  
    rank_name VARCHAR(64)  
);
```

```
CREATE TABLE users (  
  user_id SERIAL PRIMARY KEY,  
  username VARCHAR(64),  
  rank INTEGER,  
  mail VARCHAR(255),  
  password VARCHAR(255),  
  FOREIGN KEY (rank) REFERENCES rank (rank_id)  
);
```

```
CREATE TABLE district (  
  district_id SERIAL PRIMARY KEY,  
  district_name VARCHAR(64)  
);
```

```
CREATE TABLE defects (  
  defect_id SERIAL PRIMARY KEY,  
  user_id INTEGER NOT NULL REFERENCES user (user_id),  
  technican_id INTEGER REFERENCES technicians (technican_id),  
  district_id INTEGER REFERENCES district (district_id),  
  type_id INTEGER REFERENCES defect_types (type_id),  
  priority_id INTEGER REFERENCES priorities (priority_id),  
  date_added DATE,  
  last_modified DATE,  
  status VARCHAR(64),  
  comment VARCHAR(255),  
  gps_location VARCHAR(255) NOT NULL,  
  address VARCHAR(255),  
  feedback TEXT  
);
```



rys. 6 Stworzone tablice widoczne w programie PgAdmin

## Wprowadzenie danych do tabel (część)

### Tablica district

```
INSERT INTO "district" (district_name ) VALUES  
('Centrum'),  
('Baranowka'),  
('Podwislocze'),  
('Staroniwa'),  
('Pobitno'),  
('1000-lecia'),  
('Mieszka 1');
```

### Tablica status

```
INSERT INTO "status" (status_name ) VALUES  
('Usunieta'),  
('Oczekujaca na potwierdzenie'),
```

```
('Oczekujaca na przypisanie'),  
( 'Oczekujaca na naprawe'),  
( 'W trakcie naprawy'),  
( 'Naprawiona');
```

### Tablica defect types

```
INSERT INTO "defect_types" (type_name, defect_type_description) VALUES  
( 'droga', 'wady drogi takie jak koleiny czy dziury utrudniajace poruszanie sie po  
niej'),  
( 'chodnik', 'wady chodnika utrudniajace poruszanie sie '),  
( 'komunikacja miejska', 'usterki zwiazane ze stanem przystankow, biletomatami itp'),  
( 'park', 'usterki zwiazane z infrastruktura znajdujaca sie w parkach miejskich'),  
( 'pustostany', 'budynki stwarzajace zagrozenie np odpadajacy tynk'),  
( 'miejsca parkingowe', 'usterki zwiazane ze stanem miejsc parkingowych i  
parkomatow'),  
( 'inne', 'usterki nie zwiazane z zadna inna kategoria');
```

### Tablica priorities

```
INSERT INTO "priorities" (priority_name, description) VALUES  
( 'estetyczne', 'usterka nie stwarza niebezpieczenstwa dla mieszkancow, nie wpływa  
istotnie na uzytkowanie infrastrury miejskiej ale wpływa negatywnie na walory  
estetyczne'),  
( 'uczalliwe', 'usterka nie stwarza zagrozenia ale wpływa negatywnie na uzytkowanie  
rzeczy której dotyczy'),  
( 'potencjalnie niebezpieczne', 'usterki mogace stwarzac zagrozenie w pewnych  
warunkach np. dziury na drodze, nierownosci chodnika mogace powodowac  
potkniecie sie itp'),  
( 'niebezpieczne', 'usterki wymagajace natychmiastowej uwagi');
```

### Tabela technicians

```
INSERT INTO "technicans" (name, surname, mail,password,role) VALUES  
(('Milosz','Krawiec','mkrc@wp.pl','e46he6','administrator zgloszen'),  
(('Jan','Kowalski','jk@gmail.com','sebkvtjsw383','mechanik');
```

### Tabela rank

```
INSERT INTO "rank" (rank_name) VALUES  
(('nowy uzytkownik'),  
(('zaawansowany'),  
(('weteran');
```

### Tabela users

```
INSERT INTO "user" (username,rank,mail,password) VALUES  
(('jk89',1,'jk89@wp.pl','rvasrew'),  
(('malkontent66',3,'mkwjs@gmail.com','93058gjg53'),  
(('zrezygnowany94',2,'prod@gmail.com','g3450g8j');
```

### Tabela defects



```

INSERT INTO "defect" (user_id,technican_id,district_id, type_id, priority_id,
date_added,last_modified,status_id,comment,gps_location,address,feedback)
VALUES
(1,1,1,2,1,'2024-01-22','2024-01-26',3,'chodnik zapada sie od nieprawidlowo
parkujacych samochodow','50.03,20.99','ul Cieplińskiego',null),
(2,1,3,1,1,'2024-02-05','2024-03-01',5,'Koleiny na drodze','52.03,21.99','ul
wyzwolenia','przyblizona data naprawy to czerwiec 2024'),
(1,null,2,3,1,'2024-05-22',null,2,'popekana szyba przystanku
autobusowego','49.98,20.79','ul Lisa Kuli',null),
(3,null,5,6,1,'2024-04-29','2024-05-01',3, 'oczekiwanie na przydzial
technika','uszkodzony znak informujacy o poczatku miejsc
parkingowych','50.78,21.01','ul. Szkolna',null);

```

rys 3. Przykład wprowadzonych danych – tabela defects

## 12. proste zapytania SQL

```

SELECT date_added, status,comment FROM "defect" WHERE user_id=1;

```

	date_added date	status character varying (64)	comment character varying (255)
1	2024-01-22	potwierdzona	chodnik zapada sie od nieprawidlowo parkujacych samochodow
2	2024-05-22	oczekujaca na potwierdzenie	popekana szyba przystanku autobusowego

rys 7. usterki dodane przez użytkownika o id=1

```
SELECT defect_id,technican_id, priority_id fro FROM m "defect" WHERE
technican_id IS NULL;
```

	defect_id [PK] integer	technican_id integer	priority_id integer
1	3	[null]	1
2	4	[null]	1

rys. 8 usterki, które nie mają jeszcze przyporządkowanego odpowiedzialnego za nie technika

```
SELECT defect_id, district.district_name, status FROM "defect" INNER JOIN
"district" ON defect.district_id=district.district_id WHERE defect.district_id=1;
```

	defect_id integer	district_name character varying (64)	status character varying (64)
1	1	Centrum	potwierdzona

rys. 9 Usterki zgłoszone w centrum

### 13. Użytkownicy bazy danych oraz ich role

### **Użytkownik (niezalogowany)**

- Użytkownik niezalogowany posiada ograniczoną możliwość przeglądania zawartości bazy danych za pośrednictwem aplikacji, która ogranicza się do szukania i przeglądania usterek (tabela defect)

### **Użytkownik (zalogowany)**

- Użytkownik zalogowany może przeglądać istniejące defekty, może dodawać nowe do bazy danych za pośrednictwem aplikacji, kasować defekty zgłoszone przez siebie oraz je modyfikować. Może również modyfikować swoje dane z tabeli user takie jak mail oraz hasło (zmiana hasła do konta)

### **Technik**

- Technicy to pracownicy zajmujący się naprawianiem zgłoszonych usterek w terenie
- Dany technik zostaje przydzielony do konkretnej usterki i jest za nią odpowiedzialny
- Technik może zmieniać status przydzielonej mu usterki, dodawać komentarze na temat postępujących prac lub przyczyn zwłoki
- Ze względu na role technicy mogą pełnić też inne funkcje jak np. stacjonarny mógłby zajmować się wstępną weryfikacją zgłoszonych usterek, weryfikacją w terenie i przypisywaniem usterek do techników zajmujących się naprawami lub odrzucaniem niepoprawnych zgłoszeń

### **Administratorzy**

- Posiadają najwyższy poziom uprawnień i mogą wykonywać wszystkie operacje na bazie danych, w tym tworzenie i usuwanie użytkowników, tabel i baz danych, przyznawanie i odbieranie uprawnień, konfigurowanie parametrów serwera i wykonywanie kopii zapasowych.

- Odpowiedzialni są za utrzymanie bezpieczeństwa, wydajności i dostępności bazy danych.

## 14. Zaawansowane zapytania SQL (dodawanie, aktualizacja, selekcja danych)

Poniższy kod wybiera trzech użytkowników, którzy posiadają największą liczbę zgłoszeń z tablicy **defects**, wyświetla nazwę użytkownika, nazwę ich rangi oraz sumę wszystkich zgłoszeń i sortuje tabelę wynikową po liczbie zgłoszeń malejąco. Wyświetlić powinni się użytkownicy, którzy mogą nie mieć przyporządkowanej rangi.

```
SELECT
    users.username, ranks.rank_name, count(*) as Liczba_zgloszen
from
    defects
inner join users on users.user_id = defects.user_id
left join ranks on users.rank = ranks.rank_id
group by
    users.username, ranks.rank_name
order by count(*) DESC
limit 3;
```

	username character varying (64) 🔒	rank_name character varying (64) 🔒	liczba_zgloszen bigint 🔒
1	zrezygnowany94	zaawansowany	2261
2	malkontent66	weteran	2193
3	jk89	nowy uzytkownik	2154

rys. 10 Ranking użytkowników

Kolejna funkcja tworzy tabelę usterek, które nie zostały naprawione bądź odrzucone od czterech lat i mają najwyższy priorytet.

```

SELECT
    * FROM defects
INNER JOIN priorities ON priorities.priority_id = defects.priority_id
INNER JOIN status ON status.status_id = defects.status_id
where
    priorities.priority_name like 'niebezpieczne'
AND defects.date_added + 4*365 < Now()
AND status_name NOT LIKE 'Usunieta'
AND status_name NOT LIKE 'Naprawiona';

```

	defect_id integer	user_id integer	technican_id integer	district_id integer	type_id integer	priority_id integer	date_added date
1	39	1	1	7	6	4	2020-02-22
2	2014	1	1	6	5	4	2020-03-15
3	117	1	2	7	6	4	2020-05-09
4	327	2	1	3	2	4	2020-04-11
5	413	1	2	3	3	4	2020-01-04
6	499	1	2	2	5	4	2020-02-26
7	512	3	2	4	3	4	2020-03-21
8	589	2	2	6	4	4	2020-04-13
9	752	1	2	3	5	4	2020-01-20

rys 11 Wynik działania powyższego zapytania

Przyporządkowanie wszystkim usterkom od początku roku na ul. Brydaka technika o id=2. Pomijane są usterki odrzucone i naprawione.

## UPDATE

defects

## SET

technican\_id=2

## WHERE

defects.adress **like** 'ul. Brydaka'

**AND** date\_added > '2024-01-01'

**AND** status\_id != 1

**AND** status\_id != 6;

	defect_id [PK] integer	date_added date	technican_id integer	status_id integer
1	2386	2024-05-01	1	6
2	2397	2024-03-05	2	6
3	2362	2024-01-20	2	4
4	2371	2024-04-26	2	3
5	2391	2024-04-18	2	3
6	2406	2024-05-27	2	3

rys. 12 Fragment tabeli po aktualizacji

## 15. Funkcje i procedury obsługujące Bazę danych

Przykładową procedurą, która może się przydać w środowisku testowym jest procedura generowania rekordów do bazy danych. Napisałem poniższą procedurę, która pozwoliła mi zapełnić tabelę **defect** danymi:

### DECLARE

```

v_user_id INTEGER:= FLOOR(RANDOM() *3 +1);
v_technican_id INTEGER := FLOOR(RANDOM() * 2 + 1);
v_district_id INTEGER;
v_type_id INTEGER;
v_priority_id INTEGER;
```

```
v_date_added DATE;
v_last_modified DATE;
v_gps_locations VARCHAR[] := ARRAY['50.0413, 21.9990', '50.0356, 21.9998',
'50.0372, 21.9858', '50.0342, 21.9846', '50.0292, 21.9736'];
v_status_id INTEGER;
v_gps_location VARCHAR;
v_random_index INTEGER;
```

## BEGIN

```
-- Losowe wartości dla district_id, type_id, priority_id, status_id
v_district_id := FLOOR(RANDOM() * 7 + 1);
v_type_id := FLOOR(RANDOM() * 7 + 1);
v_priority_id := FLOOR(RANDOM() * 4 + 1);
v_status_id := FLOOR(RANDOM() * 6 + 1);

-- Losowe daty dla date_added i last_modified począwszy od roku 2020
v_date_added := '2020-01-01'::date + INTERVAL '1 day' * FLOOR(RANDOM() *
(DATE '2024-06-01' - DATE '2020-01-01'));
v_last_modified := v_date_added + INTERVAL '1 day' * FLOOR(RANDOM() *
100 + 1);

-- Losowy GPS
v_random_index := FLOOR(RANDOM() * ARRAY_LENGTH(v_gps_locations,
1) + 1);

SELECT v_gps_locations[v_random_index] INTO v_gps_location;

-- Wstawienie rekordu do tabeli defects
INSERT INTO defects (user_id, technician_id, district_id, type_id, priority_id,
```



```
date_added, last_modified, comment_, gps_location, adress, feedback, status_id)
  VALUES (v_user_id,v_technican_id, v_district_id, v_type_id, v_priority_id,
v_date_added, v_last_modified, NULL, v_gps_location, 'ul. Kwiatkowskiego',
NULL, v_status_id);

END;
```

w powyższym kodzie zmieniałem tylko ręcznie pole adresu (ulicy).

Utworzyłem funkcję get\_defects(), która pozwala na proste zapytanie, którego rezultatem jest czytelna tablica defektów (nazwy pól zamiast id)

```
BEGIN
  RETURN QUERY
  SELECT
    defects.defect_id,
    users.username,
    district.district_name,
    defect_types.type_name,
    priorites.priority_name,
    defects.date_added,
    defects.last_modified,
    defects.comment_,
    defects.gps_location,
    defects.adress,
    defects.feedback,
    status.status_name
```

**FROM**

defects

**INNER JOIN** users **ON** users.user\_id = defects.user\_id

**INNER JOIN** district **ON** district.district\_id = defects.district\_id

**INNER JOIN** defect\_types **ON** defect\_types.type\_id = defects.type\_id

**INNER JOIN** priorities **ON** priorities.priority\_id = defects.priority\_id

**INNER JOIN** status **ON** status.status\_id = defects.status\_id;

**END;**

wywołanie wygląda następująco:

**select \* from** get\_defects());

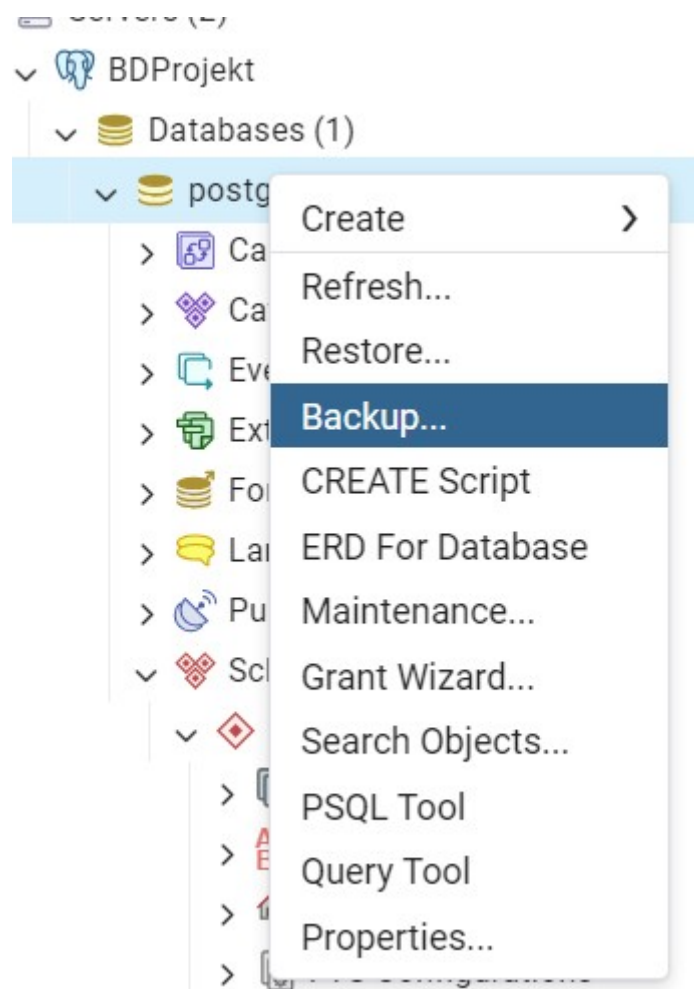
	defect_id integer	username character varying	district_name character varying	type_name character varying	priority_name character varying	date_added date	last_modified date	comment_ character varying
1	1	jk89	Centrum	chodnik	estetyczne	2024-01-22	2024-01-26	chodnik zapada się
2	2	malkontent66	Podwisłocze	droga	estetyczne	2024-02-05	2024-03-01	Koleiny na drodze
3	3	jk89	Baranowka	komunikacja miejska	estetyczne	2024-05-22	[null]	popiekana szyba pr
4	4	zrezygnowany94	Pobitno	miejsca parkingowe	estetyczne	2024-04-29	2024-05-01	uszkodzony znak i

rys 13 Wynik działania powyższej funkcji (fragment tabeli)

## 16. Zarządzanie bazą danych (backup, restore)

Zarządzanie bazą danych może się odbywać za pomocą aplikacji pgAdmin.

Posiada on funkcje umożliwiające robienia backup i restore.



Rys. 14 PgAdmin opcja backup bazy danych

W następnym okienku wystarczy podać nazwę kopii zapasowej i kliknąć backup:

**Backup (Database: postgres)** ✕

General Data Options Query Options Table Options Options Objects

Filename	<input type="text" value="projektBDBBackup"/> <span>📁</span>
Format	<input type="text" value="Custom"/>   ▾
Compression ratio	<input type="text"/>
Encoding	<input type="text" value="Select an item..."/>   ▾
Number of jobs	<input type="text"/>
Role name	<input type="text" value="Select an item..."/>   ▾

ℹ ? ✕ Close ↺ Reset 💾 Backup

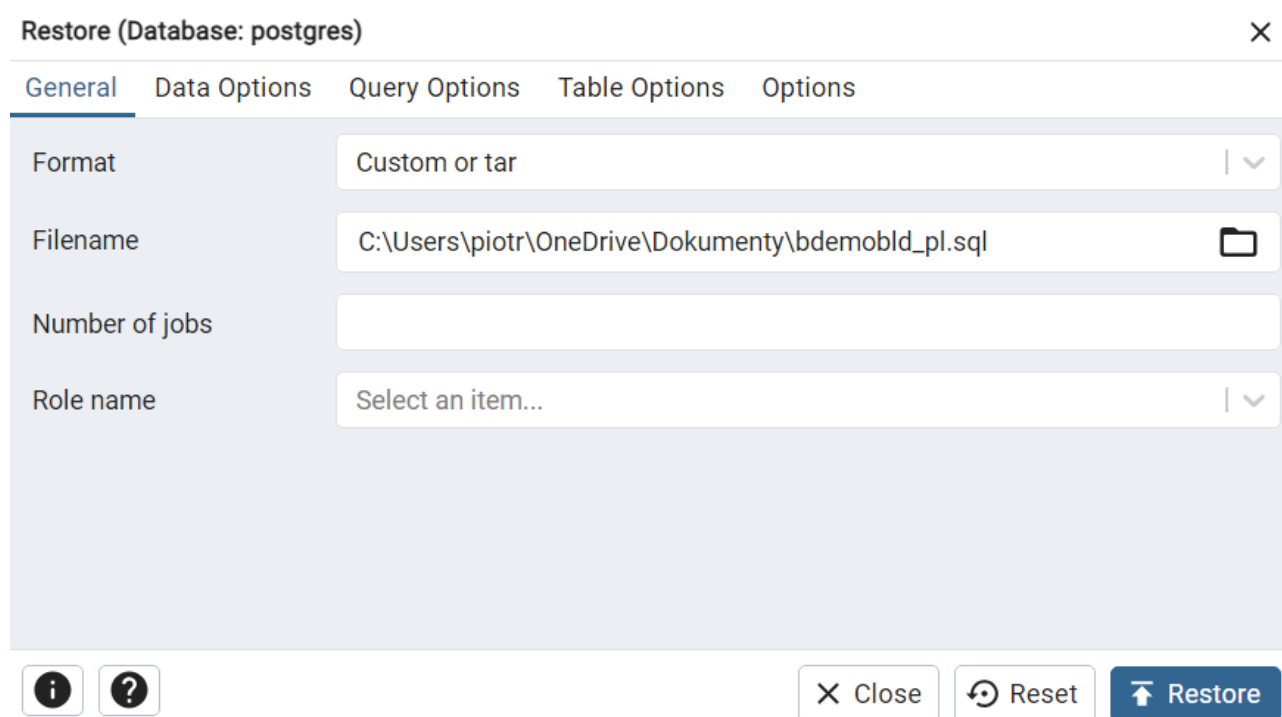
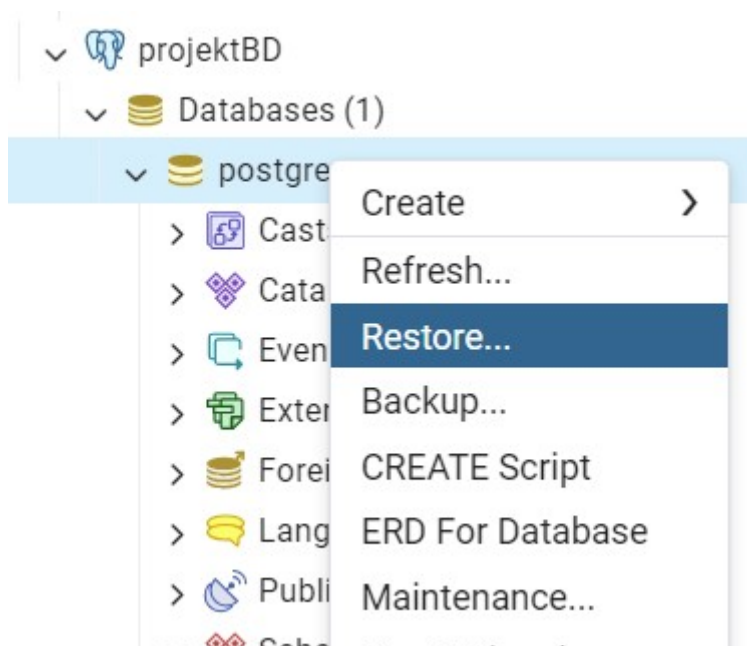
**Process completed** ✕

Backing up an object on the server 'projektBD (localhost:5432)' from database 'postgres'

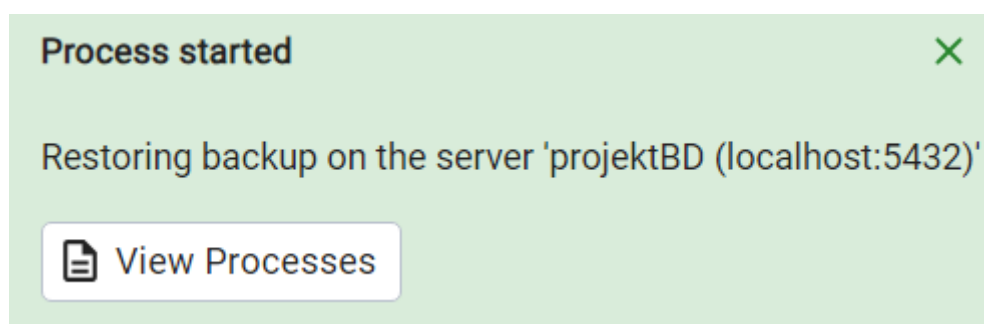
📄 View Processes

Rys. 15 PgAdmin opcja backup bazy danych

Podobnie działa to w wypadku przywracania:



rys. 16 Okno dialogowe przywracania kopii zapasowej



rys. 17 Komunikat przy przywracaniu kopii zapasowej

## 17. Wariant testowy i na produkcji

Przy tworzeniu wariantu testowego można posłużyć się opisanym w poprzednim punkcie mechanizmem backup-restore.

Backup (Database: postgres)

General

Data Options

Query Options

Table Options

Options

Objects

Filename

database\_test\_backup

Format

Custom

Compression ratio

Encoding

Select an item...

Number of jobs

Role name

Select an item...

i

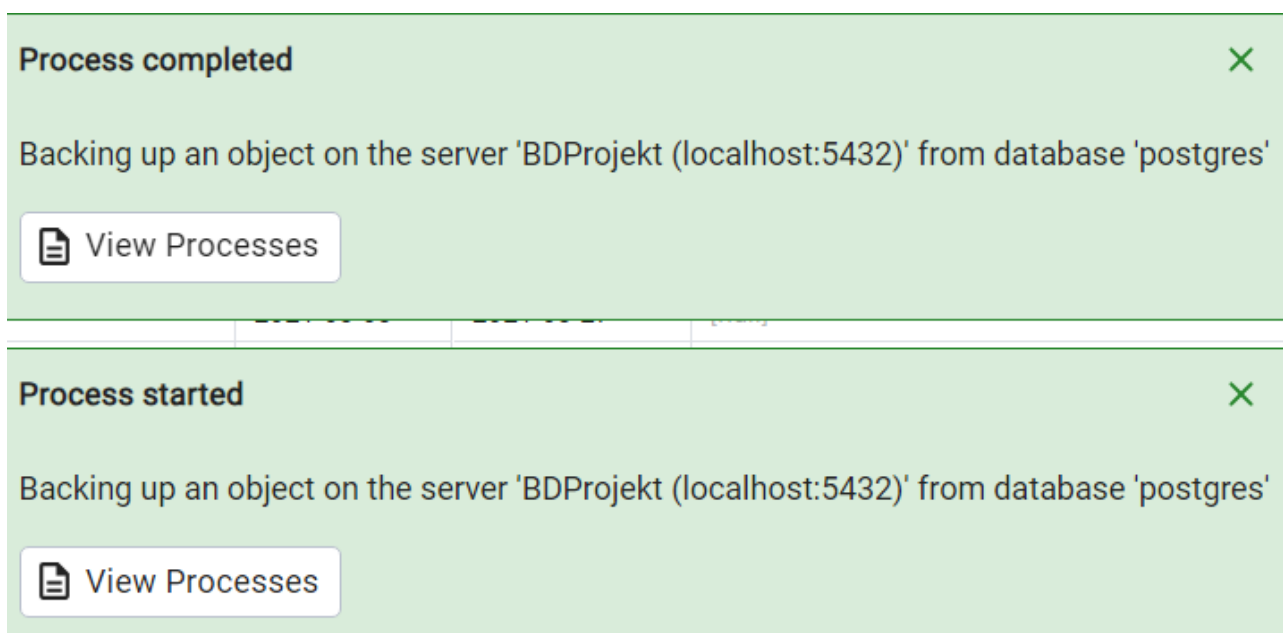
?

Close

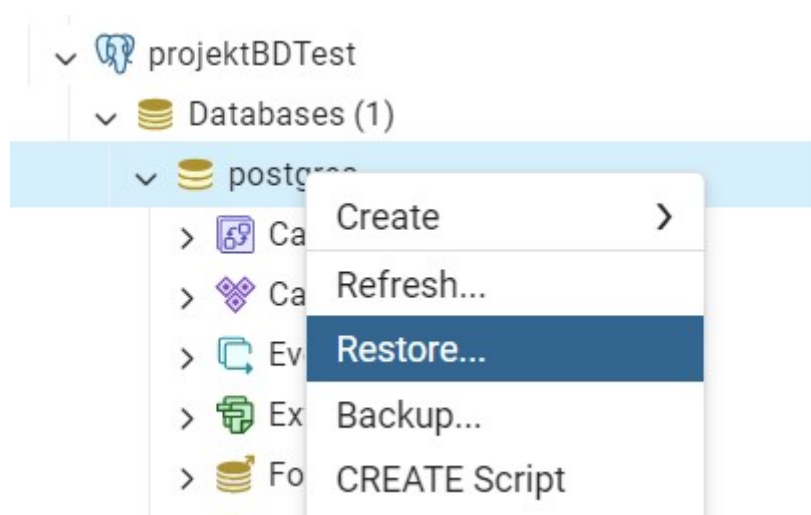
Reset

Backup

Rys. 18 Tworzenie kopii zapasowej dla wariantu testowego bazy danych (1)



Rys. 18 Tworzenie kopii zapasowej dla wariantu testowego bazy danych (2)



Rys. 18 Tworzenie kopii zapasowej dla wariantu testowego bazy danych (3)

Restore (Database: postgres) ✕

General

Data Options

Query Options

Table Options

Options

Format

Custom or tar

Filename

C:\Users\piotr\OneDrive\Dokumenty\database\_test\_backup

Number of jobs

Role name

Select an item...

?

?

✕ Close

↺ Reset

↶ Restore

Rys. 18 Tworzenie kopii zapasowej dla wariantu testowego bazy danych (4)

W rezultacie mamy testową bazę danych będącą kopią głównej BD:

postgres/postgres@projektBDTest

📁

📄

✎

🔍

No limit

📊

▶

⌂

📈

🔄

🔄

☰

?

Query

Query History

1 select \* from defects;

Data Output

Messages

Notifications

⌵

📄

📋

🗑️

📦

⬇️

📈

	defect_id [PK] integer	user_id integer	technican_id integer	district_id integer	type_id integer	priority_id integer	date_added date	last_modified date	cor ch
1	1	1	1	1	2	1	2024-01-22	2024-01-26	ch
2	2	2	1	3	1	1	2024-02-05	2024-03-01	Ko
3	3	1	[null]	2	3	1	2024-05-22	[null]	po
4	4	3	[null]	5	6	1	2024-04-29	2024-05-01	us:
5	7	2	2	4	2	2	2021-03-03	2021-03-27	[nl

rys. 19 Sprawdzenie zawartości testowej bazy danych



## 18. Komunikacja z językiem programowania

Do komunikowania się z bazą danych postanowiłem napisać prostą aplikację okienkową w języku java. Do budowania aplikacji i ściągania niezbędnych bibitek użyłem narzędzia Maven.

Kod do łączenia się z bazą danych w celu wykonania zapytania wygląda następująco:

```
try(Connection connection =
DriverManager.getConnection(jdbcUrl, username, password)) {
    PreparedStatement ps =
connection.prepareStatement(query.getSearchQuery());
    ResultSet result = ps.executeQuery();
    result = ps.executeQuery();
    Defect temp;
    int i=1;
    List <Defect> defects = new ArrayList<>();
    while (result.next()) {

        temp = new DefectBuilder(i++)
            .setUserName(result.getString(2))
            .setDistrict(result.getString(3))
            .setType(result.getString(4))
            .setpriority(result.getString(5))
            .setDateAdded(result.getString(6))
            .setDateLastModified(result.getString(7
))

            .setComment(result.getString(8))
            .setGpsLocation(result.getString(9))
            .setAdress(result.getString(10))
            .setFeedback(result.getString(11))
            .setStatusId(result.getString(12))
            .build();

        defects.add(temp);
    }
    structure.setDefects(defects);

} catch (SQLException e) {
```

```

        JOptionPane.showMessageDialog(null,"Błąd połączenia z
bazą danych","Błąd SQL", JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

```

Klasa SearchQuery wykorzystywana do konstruowania zapytania wygląda następująco:

```

public class SearchQuery {

    private String address;
    private String district;
    private String type;
    private String status;

    public SearchQuery( String address, String
district,String type,String status)
    {
        this.address=address;
        this.district=district;
        this.type=type;
        this.status=status;
    }

    public String getSearchQuery()
    {
        String result= "SELECT * from get_defects() WHERE";

        if(address.length()>0) result += " adres like '" +
address + "' ";
        else result += " adres like 'Rzeszów' ";
        if(district.length()>0) result += " AND district like
'" + district + "' ";
        if(type.length()>0) result += "AND type_name like '"
+ type + "' ";
        if(status.length()>0) result += "AND status_name like
'" + status + "' ";

        result += " LIMIT 20;";
        System.out.println(result);
    }
}

```

```
        return result;
    }
```

## 19. Przykład zastosowania ORM

W projekcie zdecydowałem się na manualne mapowanie obiektów z bazy danych do programu. Napisałem klasę Defect, która opisuje usterki, dodając do niej wewnętrzną klasę DefectBuiler z powodu dużej liczby argumentów konstruktora przy tworzeniu obiektów. Zabieg ten miał poprawić czytelność kodu.

Klasa wygląda następująco:

```
String userName;
    int technicanId;
    String district;
    String type;
    String priority;
    String date_added;
    String last_modified;
    String comment;
    String gpsLocation;
    String adress;
    String feedback;
    String status;

    @Override
    public String toString()
    {
        return defectId + " " + type+ " " +adress;
```

dodatkowo klasa posiada gettery i settery, których nie było potrzeby tu umieszczać.

Klasa wewnętrzna DefectBuilder (fragment)

```
//Builder
    private Defect(DefectBuilder d, int id)
    {
```

```

        this.defectId=id;
        this.adress=d.adress;
        this.comment=d.comment;
        this.date_added=d.date_added;
        this.last_modified=d.last_modified;
        this.district=d.district;
        this.feedback=d.feedback;
        this.technicanId=d.technicanId;
        this.userName=d.userName;
        this.type=d.type;
        this.priority=d.priority;
        this.gpsLocation=d.gpsLocation;
        this.feedback=d.feedback;
        this.status=d.status;
    }

    public static class DefectBuilder
    {
        final int defectId;
        String userName;
        int technicanId;
        String district;
        String type;
        String priority;
        String date_added;
        String last_modified;
        String comment;
        String gpsLocation;
        String adress;
        String feedback;
        String status;

        public DefectBuilder(int defect_id)
        {
            this.defectId=defect_id;
        }
        public DefectBuilder setTechnicanId(int technican_id)
        {
            this.technicanId=technican_id;
            return this;
        }
        public DefectBuilder setDistrict(String districtName)
        {

```

```

        this.district=districtName;
        return this;
    }

```

...

Tworzenie obiektu defect jest pokazane w kodzie poniżej:

```

result = ps.executeQuery();
Defect temp;
int i=1;
List <Defect> defects = new ArrayList<>();
while (result.next()) {

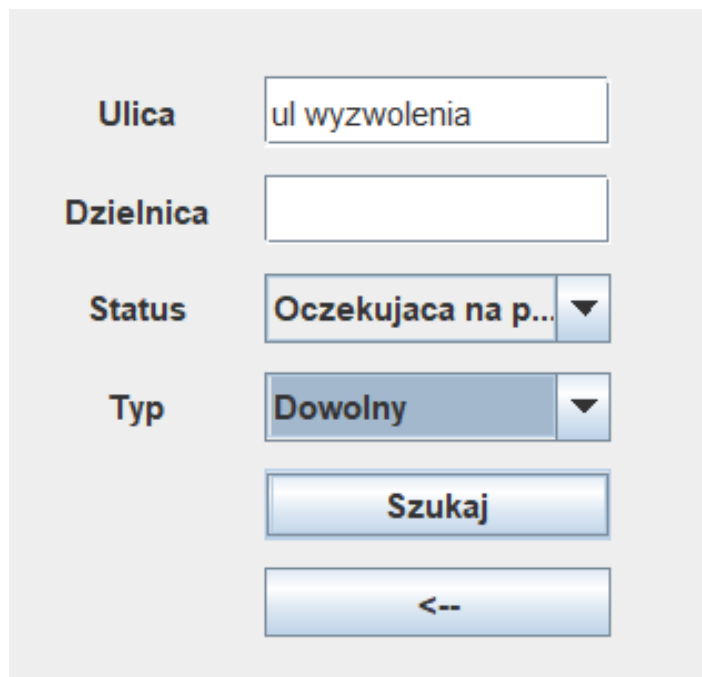
    temp = new DefectBuilder(i++)
        .setUserName(result.getString(2))
        .setDistrict(result.getString(3))
        .setType(result.getString(4))
        .setpriority(result.getString(5))
        .setDateAdded(result.getString(6))
        .setDateLastModified(result.getString(7
))

        .setComment(result.getString(8))
        .setGpsLocation(result.getString(9))
        .setAdress(result.getString(10))
        .setFeedback(result.getString(11))
        .setStatusId(result.getString(12))
        .build();

    defects.add(temp);
}
structure.setDefects(defects);

```

Przykład przeszukiwania bazy danych przy użyciu formularza w napisanej aplikacji:



The image shows a search form with the following elements:

- Ulica**: A text input field containing the text "ul wyzwolenia".
- Dzielnica**: An empty text input field.
- Status**: A dropdown menu with the selected option "Oczekujaca na p..." and a downward arrow.
- Typ**: A dropdown menu with the selected option "Dowolny" and a downward arrow.
- Szukaj**: A blue button with the text "Szukaj".
- <--**: A blue button with the text "<--".

rys. 20 Formularz wyszukiwania

1 droga ul wyzwolenia

Typ zgłoszenia:      droga

Priorytet:      estetyczne

Adres:      ul wyzwolenia

Dzielnica:      Podwisłocze

GPS:      52.03,21.99

Data dodania:      2024-02-05

Data Modyfikacji:      2024-03-01

Zgłoszony przez:      malkontent66

Informacje

Komentarz:  
Koleiny na drodze  
Informacja zwrotna:  
przybliżona data naprawy to czerwiec 2024

rys. 22 Okno z wynikami wyszukiwania