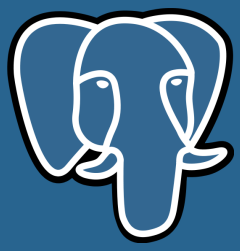
Na zlecenie Komendy Wojewódzkiej PSP w Kielcach

Tomasz Nycz

PostgreSQL/PostGIS

Dobre praktyki

Maj 2017



1 Konfiguracja i zabezpieczenie serwera

1.1 Podstawowe ustawienia

Podstawowe ustawienia serwera bazy danych PostgreSQL znajdują się w plikach konfiguracyjnych *postgresql.conf* i *pg_hba.conf* Ich lokalizacje zależne są od systemu operacyjnego, zobacz przykłady poniżej

Debian/Ubuntu	Windows
/etc/postgresql/9.5/main/postgresql.conf	X:/Program Files/PostgreSQL/9.5/data

Istnieje możliwość wskazania innej niż standardowa lokalizacji plików *pg_hba.conf*, *pg_ident.conf* oraz lokalizacji klastra bazodanowego (systemu plików bazy na dysku). Zmian tych dokonujemy w pliku *postgresql.conf*. Każdorazowa zmiana tych ustawień wymaga restartu serwera.

1.2 Zabezpieczenie TCP/IP

Podstawową metodą zabezpieczenia serwera bazy danych jest ograniczenie dostępu dla osób niepowołanych. Pomijając fizyczne metody zabezpieczenia (np. sieć wydzielona) najprostszym sposobem jest zmiana standardowych ustawień daemona psql w zakresie protokołu TCP/IP. W tym celu używamy dyrektyw *listen_adresses* oraz *port*. Dla listen_adresses domyślnym ustawieniem jest *localhost*, możliwe wartości to * jako wskazanie wszystkich adresów IP, lub lista adresów przedzielonych przecinkami. W tej sytuacji warto wskazać jeden konkretny adres IP dostępny dla wszystkich klientów serwera, oraz nietypowy numer portu.

```
listen_addresses = 'localhost'
port = 49666
max_connections = 100
```

1.3 Autentykacja

Kolejnym krokiem konfiguracji dostępu do baz danych jest wykorzystanie Host Based Authentication. PostgreSQL pozwala na zdefiniowanie dozwolonych metod dostępu na poziomie konkretnych hostów lub sieci, baz danych, użytkowników serwera. W tym celu w pliku *pg_hba.conf* tworzymy kolejne wpisy wg następującego schematu:

CONNECTION DATABASE USER ADDRESS METHOD [OPTIONS] .

Oto dopuszczalne wartości *Connection type*:

- local
- host

- hostssl
- hostnossl

Typ *local* zakłada dostęp wyłącznie poprzez Unix-domain socket. Jest to metoda przewidziana głównie dla połączeń administracyjnych. Metoda *host* zakłada wykorzystanie standardowego TCP/IP, a gdy jest dostępny również SSL. Pozostałe typy wymuszają konkretne zachowanie SSL.

Przykładowy plik pg_hba.conf

```
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
host    all             all             samenet                 cert
hostssl all             all             192.168.0.0/24      ldap
# IPv6 local connections:
host    all             all             ::1/128                 trust
```

2 Zabezpieczenie bazy danych

2.1 System GRANT

Podstawową formą zabezpieczenia in-dbase w psql jest system GRANT/REVOKE. Sprowadza się do bardzo prostych zapytań:

```
GRANT uprawnienie ON objekt TO rola;
REVOKE uprawnienie ON objekt TO rola;

```

Dostępna lista uprawnień to:

- SELECT

  Pozwala na wykonanie operacji SELECT z obiektów tabeli. Jest to podstawowe uprawnienie konieczne dla dostępu do tabeli.

- INSERT

  Pozwala na tworzenie nowych wierszy. Można wskazać również konkretne kolumny dla których uprawnienie będzie dostępne. Uprawnia również do COPY FROM.

- UPDATE

  Pozwala na aktualizację istniejącego wiersza

- DELETE

  Pozwala usunąć pojedyńczy wiersz

- TRUNCATE

  Czyści zawartość tabeli, pozostawia strukturę.

- REFERENCES Pozwala na tworzenie kluczy i indeksów.

- TRIGGER

  Uprawnia do tworzenia wyzwalaczy

- CREATE

  Dla bazy pozwala na tworzenie schematów, w schematach tabel, w przypadku tabel również indeksów, oraz obiektów tymczasowych o to create, alter, or drop his own user's user mappings associated with that server.

2.2 Row-level security

Szczególną formą systemu GRANT jest tzw. Row-level Security (RLS), istniejące w Postgresql od wersji 9.5.

```
CREATE TABLE hydranty (
user_role_name NAME,
typ TEXT,
srednica TEXT
);

ALTER TABLE hydranty ENABLE ROW LEVEL SECURITY;

CREATE POLICY uzytkownik_hydrant ON hydranty
FOR SELECT, DELETE
USING(user_role_name = current_user);

CREATE POLICY uzytkownik_hydrant ON hydranty
FOR SELECT, DELETE
WITH CHECK(user_role_name = current_user);

```

2.3 Kopia bezpieczeństwa

2.3.1 pg_dump

Podstawową formą tworzenia kopii bezpieczeństwa jest polecenie pg_dump. Wywołujemy je ze wskazaniem bazy danych, tabel, oraz formatu kopii.

Przykładowe wywołanie poniżej:

```
pg_dump baza \> plik.sql
```

Jest to najprostsza forma, wykonująca kopię bezpieczeństwa całości wskazanej bazy danych (wszystkie schematy, oraz funkcje). Istotnym jest zaznaczenie że operator wykonujący polecenie pg_dump, musi posiadać uprawnienia dostępu do wszystkich obiektów w wskazanej bazie. A więc najczęściej operację tą wykonywać będzie superużytkownik bazy danych.

Kolejny przykład to replikacja jednorazowa pomiędzy dwoma serwerami bazy danych.

```
pg_dump -h localhost baza | psql -h 192.168.0.100 baza1
```

Inne ważne ustawienia pg_dump to:

- -E encoding
- -F format
- -n schema / -N schema
- -t schema / -T schema
- -d dbname -h host -p port -U username

W przypadku przełącznika format możemy przyjąć następujące wartości:

- p plain
- c custom

- d directory
- t tar

Standardowo powinniśmy używać formatu plain jeśli mamy zamiar korzystać z przywracania przy pomocy **psql** lub **pgAdmin**, lub custom/directory dla przywracania przy pomocy **pg_restore**

Dla zarchiwizowania bazy danych znajdującej się na odległym serwerze do kopii bezpieczeństwa użyjemy np. takiego polecenia:

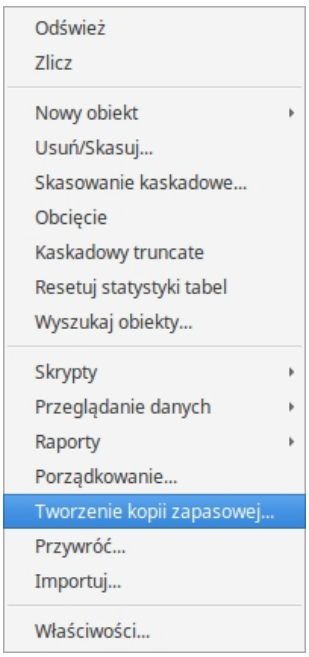
`pg_dump -h 192.168.0.102 -U osm -d osm -Fc -f osm.sql`

Przywrócenia takiej bazy zostanie analogicznie wykonane poleceniem:

`pg_restore -h 192.168.0.102 -U osm -d osm -f osm.sql`

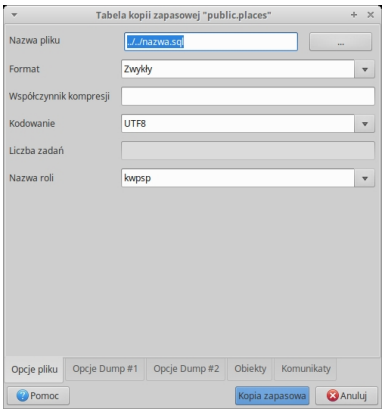
2.3.2 pgAdmin

W przypadku pgAdmin'a wykonywanie kopii bezpieczeństwa dokonuje się dla każdego poziomu (baza, schema, tabela) w podobny sposób. Opcja ta dostępna jest między innymi z menu kontekstowego (prawy klawisz myszy). Następnie w oknie wskazujemy plik docelowy na dysku,

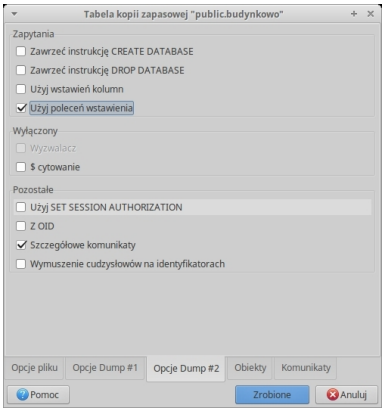


Rysunek 2.1: Menu kontekstowe pgAdmin

tryb wykonania kopii (zwykły, użytkownika, katalog lub TAR), kodowanie znaków, oraz rolę użytkownika (preferowana ta sama rola, która jest właścicielem tabeli). Należy pamiętać że w przypadku przywracania tabeli z kopii, z poziomu programu pgAdmin, konieczne jest użycie wstawień wierszy, dostępne z drugiej zakładki.



Rysunek 2.2: pgAdmin - ustawienia kopii zapasowej



Rysunek 2.3: Kopia zapasowa - Użyj wstawień wierszy

3 Schema bazy i partycjonowanie

3.1 Tabela partycjonowana

3.2 Widok i tabela tymczasowa

Istnieją dwie formy dostępu do bazy danych w oparciu o tymczasową lub ograniczoną wersję danych. Są to widoki oraz tabele tymczasowe. Widok nie zmienia w żaden sposób danych źródłowych, lecz każdorazowo wykonuje zapytanie typu SELECT do tabel źródłowych. Zaletą jest oszczędność miejsca na dysku, dane nie są duplikowane, podstawową wadą takiego rozwiązania jest czas wykonania zapytania do widoku - szczególnie w przypadku skomplikowanych konwersji. Widok tworzymy poleceniem:

```
CREATE VIEW nazwa AS SELECT * FROM tabela;
```

Szczególną formą jest widok zmaterialiowany - dane wynikowe są przechowywane w gotowej postaci, lecz w każdej chwili można je zaktualizować przy pomocy predefiniowanego zapytania. Służą do tego następujące polecenia:

```
CREATE MATERIALIZED VIEW widok_m AS SELECT x FROM y;
REFRESH MATERIALIZED VIEW widok_m;
```

Innym rozwiązaniem, szybszym i wydajniejszym od widoku jest tabela tymczasowa. Tworzona jest ona tylko na czas trwania danej sesji w psql. Taka forma jest szczególnie wygodna wtedy gdy dane które będą przechowywane w tej tabeli, mają służyć do zagregowania, czy innej analityki, a następnie będą usuwane.

```
CREATE TEMPORARY TABLE nazwa AS ... ON COMMIT operacja;
```

gdzie wartość parametru operacja to - PRESERVE ROWS, DELETE ROWS lub DROP.

3.3 Schema bazy danych

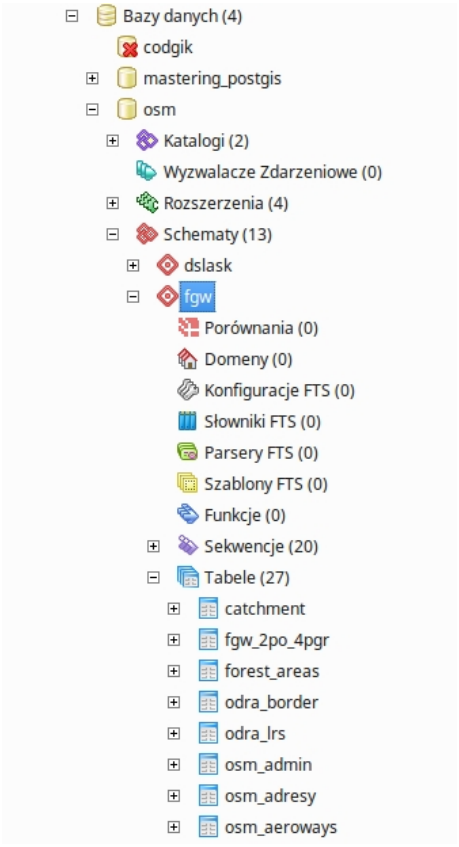
Schemat bazy danych to nazwana kolekcja elementów bazy danych. Może zawierać tabele, widoki, indeksy, sekwencje, typy danych, operatory i funkcje. Najbliższym odpowiednikiem w systemie operacyjnym jest katalog. Jednak w przypadku schematu bazy danych zagnieżdżanie jest niemożliwe. Jako główną przyczynę wykorzystania schematu bazy danych możemy wymienić konieczność rozdzielenia użytkowników i/lub aplikacji, tak, aby nazwy tworzonych obiektów (np. tabele tymczasowe), nie kolidowały ze sobą.

W analogii do tworzenia tabeli schemat bazy danych tworzymy wywołaniem `CREATE SCHEMA nazwa;` Aby przenieść istniejącą tabelę do wybranej schemy wykorzystujemy słowo kluczowe ALTER.

```
ALTER TABLE nazwa\_tabeli SET SCHEMA nazwa\_schemy;
```

Należy pamiętać że domyślnie PostgreSQL wyszukuje tabele wyłącznie w schemacie *public*. Jeśli chcemy ułatwić sobie życie, skorzystajmy z następującej zmiennej środowiskowej.

```
SET search_path = nazwa_schemy, public;
```



Rysunek 3.1: pgAdmin - zawartość przykładowego schematu

Dla narzędzi typu pgAdmin nie ma konieczności ustawiania takich zmiennych, dba o to oprogramowanie.

4 Wersjonowanie bazy danych

4.1 Audit triggers

Najprostsze rozwiązanie do nadzoru nad zmianami w bazie danych to tzw. audit triggers. Opiera się ono na tzw. wyzwalaczach (triggers), oraz funkcji realizującej operacje logowania zmian do tabeli. Dla operacji INSERT, UPDATE i DELETE tworzy się osobną procedurę logowania, a następnie wskazuje wyzwalaczowi kiedy ma zostać wywołana.

```
CREATE TRIGGER tablename_audit
AFTER INSERT OR UPDATE OR DELETE ON tablename
FOR EACH ROW EXECUTE PROCEDURE audit.if_modified_func();
```

Bardziej rozbudowana wersja tego rozwiązania znajduje się pod adresem

```
https://github.com/2ndQuadrant/audit-trigger
```

Jego instalacja jest bardzo prosta, wywołujemy skrypt SQL przy pomocy psql, a następnie wywołujemy następującym poleceniem dodanie tabeli do audytu.

```
SELECT audit.audit_table('schema.tabela');
```

Istnieje również wtyczka do QGIS pozwalająca na zarządzanie operacjami rollback względem zmian dokonywanych w tabeli. Nazywa się Postgres 91 plus Auditor.

4.2 Inne rozwiązania na rynku

Innym rozwiązaniem dostępnym na rynku jest między innymi Postgis Versioning opracowane przez firmę Oslandia. Jest to wtyczka dla QGISa, przy pomocy której wszystkie operacje związane z wersjonowaniem realizujemy w interfejsie graficznym. Wymaga ona pracy na grupach warstw. Kolejnym rozwiązaniem jest wersjonowanie dostępne w QGIS 2.16+, do którego możemy się dostać z poziomu DB Managera

5 Zapytania SQL - case studies

5.1 Zapytania Intersects

Pierwsze zapytanie ma za zadanie porównanie dla obiektów punktowych numeru TERYT TERC zapisanego w zbiorze, z numerem TERYT gminy na której terenie znajduje się geometria obiektu.

```
SELECT row_number() over (),
e.geometry,
e.teryt_gmin,
e.nazwa,
p.jpt_kod_je AS computed_teryt,
p.jpt_nazwa_ AS computed_nazwa
FROM public.jednostki e, gminy p
WHERE ST_Intersects(e.geometry,p.geom)
AND e.teryt_gmin!=left(p.jpt_kod_je,6);
```

Operację tą wykonujemy przy pomocy dwóch warunków, jednego przestrzennego, przecinania się obiektów (znajdowania się w takiej konfiguracji przestrzennej, w której mają jakąkolwiek część wspólną), oraz porównania ciągów znakowych przy pomocy operatora negacji, przy czym jeden z ciągów znakowych jest wstępnie przygotowany funkcją left(śtring",liczba)

Kolejne zadanie jest typową operacją **Zlicz punkty w poligonie**

```
SELECT COUNT(*) AS liczba, e.geom
FROM public.jednostki j, public.gminy e
WHERE st_intersects(j.geometry, e.geom) GROUP BY e.geom
```

Oprócz funkcji ST_Intersects(geomA,geomB) używana jest również klauzula GROUP BY. Służy ona do agregowania tych wierszy danych które mają wspólne dane. W naszym przypadku częścią wspólną jest e.geom, zaś COUNT(*) ma za zadanie zliczenie każdorazowo ilości wierszy zaagregowanych. Warto zaznaczyć że w klauzuli GROUP BY muszą znaleźć się wszystkie obiekty (kolumny) wybrane w zapytaniu SELECT za wyjątkiem tych które są funkcjami agregującymi (COUNT, SUM, etc.)

5.2 Tworzenie geometrii

PostGIS pozwala na tworzenie nowych geometrii w oparciu o istniejące atrybuty. Szczególnymi funkcjami są ST_Union(geom) i ST_Collect(geom). Pierwsza wykonuje operację nazywaną *Dissolve*, zaś druga tworzy tzw. GEOMETRYCOLLECTION. Zobaczmy na przykładzie połączenie wszystkich pól siatki jednokilometrowej z zbioru GUS żozmieszczenie ludności", w których wartość jest niezerowa.

```
CREATE TABLE zamieszkane AS SELECT ST_Union(geom) AS geom FROM public.ludnosc_km
   WHERE tot > '0'
```

5.3 Operatory zasięgu przestrzennego

```
&&
```

Operator ten zwraca stan logiczny dla porównania zasięgów przestrzennych tzw. bounding box. Najpopularniejszym zastosowaniem tego operatora jest wstępne ograniczenie liczby rekordów i geometrii które będą później analizowane np przy pomocy funkcji ST_Intersects()

```
SELECT ST_Intersects(a.geom, (SELECT geom FROM b)) WHERE a.geom && b.geom;
```

```
<->
```

Operator ten zwraca odległość obiektów, może służyć np. do sortowania wyników po odległości.

```
SELECT row_number () OVER () as qgisid,*
FROM warsaw.osm_emergency e
WHERE type='fire_hydrant'
AND st_intersects(e.geometry, (SELECT geometry FROM warsaw.osm_admin WHERE id
    ='14'))
ORDER BY e.geometry <-> (SELECT geometry FROM warsaw.osm_emergency WHERE name='
    Nazwa OSP' LIMIT 1)
```

Jeśli potrzebujemy uzyskać samą wartość odległości, powinniśmy wykorzystać funkcję ST_Distance(g1,g

5.4 Zarządzanie danymi

Pierwsze zadanie jakie omówimy to łączenie dwóch tabel do jednego widoku. Wykorzystujemy w tym celu klauzulę UNION. Aby połączyć więcej niż dwie tabele, każdy kolejny obiekt powinien być zagnieżdżony w kolejnym nawiasie.

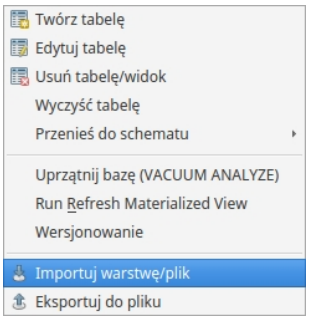
```
CREATE VIEW szkolenie.rule_them_all AS (
(SELECT * FROM szkolenie.kielecki
UNION
SELECT * FROM szkolenie2.skarzyski)
UNION
SELECT * FROM szkolenie.buski)
UNION
SELECT * FROM szkolenie2.konecki
```

6 Hurtowe importowanie danych

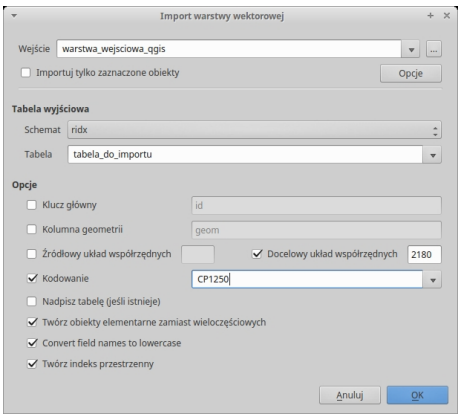
6.1 QGIS DB Manager

Podstawowym narzędziem dla zasilania bazy danych informacją przestrzenną jest QGIS wraz z swoim domyślnie uaktywnionym rozszerzeniem QGIS DB Manager. Przy pomoc tego jednego narzędzia możemy w QGIS zarządzać bazami danch PostgreSQL, SQLite, oraz GeoPackage, a także tzw. warstwami wirtualnymi - zapytaniami SQLite wykorzystującymi warstwy wektorowe OGR.

W menadżerze bazy danych, aby rozpocząć importowanie, musimy połączyć się z istniejącą, zdefiniowaną bazą danych. Po nawiązaniu połączenia w menu `Tabela->Importuj warstwę/plik`



Rysunek 6.1: Importowanie danych - menu programu



Rysunek 6.2: Importowanie danych - okno dialogowe

Następnie wskazujemy interesujące nas parametry wejściowe. Warto zwrócić uwagę na pola określające nazwę kolumny geometrii (do późniejszego wykorzystania w zapytaniach), kodowanie wejściowe (istotne przy imporcie warstw stworzonych w ArcGIS), wejściowy i docelowy układ współrzędnych (tu jeśli nie zamierzamy dokonywać konwersji układu wypełniamy tylko wartość dla docelowego układu). W kolejnych wierszach znajdziemy przydatne narzędzia dla zwiększenia funkcjonalności naszej tabeli. Tworzenie indeksu przestrzennego powinno być dla nas nawykiem

- zawsze pomaga on przyśpieszyć zapytania oparte o geometrię obiektu. Zamiana nazw pól na małe litery uprości nasze przyszłe zapytania, ponieważ PostgreSQL jest wrażliwy na wielkość liter w nazwach schematu, tabeli i kolumny. Tworzenie obiektów elementarnych gdy chcemy zadbać o rozbicie multipoligonów na obiekty elementarne.

6.2 Imposm3

Imposm3 jest opartym o język Go, oraz biblioteki libleveldb, libgeos i protobuf. Obecnie obsługiwane są wyłącznie systemy operacyjne typu Linux. Dane wejściowe muszą być w formacie *Protobuffer* (PBF). W linii poleceń wskazujemy docelową bazę danych, oraz schemat. Następnie przy pomocy LevelDB program odfiltrowuje z zbioru PBF obiekty wskazane w tzw. mappingu (schemacie bazy). Wynikowa tabela tworzona jest w układzie EPSG:2180 (PUWG 1992).

Przykładowe wywołanie Imposm3:

```
imposm import -connection postgis://osm:osm@localhost:5432/osm -mapping mapping.
    yaml -read plik.pbf -write -overwritecache -dbschema-import nasz\_schemat
```

6.3 Pakiety typu ETL

ETL czyli Extract, Transform, Load to pakiety przetwarzania danych, również przestrzennych, które pozwalają przygotować kompletne rozwiązania dla importu danych do bazy lub ich zapis do plików.

- Safe Software FME - jest jednym z najbardziej rozpowszechnionych pakietów komercyjnych na rynku, obsługującym ponad 200 różnych formatów przestrzennych i bazodanowych
- ArcGIS Data Interoperability Extension - rozszerzenie ArcGIS o trochę mniejszych możliwościach
- Talend Studio
- GeoKettle - opartę o Javę
- Hale Studio -

Jednym z przykładów wykorzystania ETL może być złączenie bazy danych BDOO oraz zbiorów tematycznych poświęconych np. zaopatrzeniu w wodę, w jedną spójną bazę danych o ujednoliconych nazwach pól.

7 Rozszerzenia PostGIS

7.1 PostGIS Raster

PostGIS Raster to rozszerzenie pozwalające wykorzystywać pliki rastrowe, obsługiwane przez GDAL bezpośrednio z poziomu bazy danych i wykonywać na nich operacje analityczne.

7.1.1 Import danych

Podstawowym sposobem rozpoczęcia pracy z rastrami w PostGIS, jest ich zaimportowanie do bazy przy pomocy *raster2pgsql*. Przykładowe wywołanie programu:

```
raster2pgsql -s 4326 -C -Y -l 2,4 -I -F -t 1000x1000 *.tif eudem.dem | psql -d baza -U uzytkownik
```

W pierwszej części polecenia wskazujemy opcje importowania, takie jak układ współrzędnych (-s XXXX), wykorzystanie składni COPY zamiast INSERT (-Y), wskazanie które poziomy powiększenia (piramidy) powinny zostać utworzone (-l x,y,z). Kolejno wskazujemy plik źródłowy w formacie obsługiwanym przez GDAL, w naszym przypadku korzystamy z wildcarda *, aby wczytać zawartość całego katalogu, oraz schemę i tabelę w bazie danych, do której ma nastąpić import. W drugiej części wywołania wskazujemy czy wygenerowany kod SQL ma zostać zapisany do pliku, czy przekazany bezpośrednio (pipe) do procesu serwera bazy danych psql. Sprawdźmy jak wygląda wynik tej operacji. W konsoli psql wpisujemy `\dt eudem.*` i uzyskujemy listę tabel w schemie *eudem*:

```
postgis_semi-# \dt eudem.*
  List of relations
 Schema |  Name   | Type  | Owner
--------+---------+-------+-------
 eudem  | dem     | table | osm
 eudem  | o_2_dem | table | osm
 eudem  | o_4_dem | table | osm
 eudem  | o_8_dem | table | osm
 (4 rows)

 postgis_semi-#
```

7.1.2 Operacje DEM

Dla szczególnego rodzaju operacji na rastrach PostGIS oferuje gotowe funkcje realizujące obliczenia na numerycznym modelu terenu. Są to operacje przygotowujące mapę spadków (slope), ekspozycji, cieniowanie rzeźby terenu, oraz współczynniki TPI i TRI.

```
SELECT ST_Slope(rast, DEGREES, 1.0) FROM eudem.dem;
```

wygeneruje mapę nachylenia stoku (spadku). Wszystkie parametry poza kolumną z danymi rastrowymi są opcjonalne, ale pozwalają np. na przeliczenie jednostek Z z stopni geograficznych do metrów (scale=111200).

7.1.3 Eksport danych

Eksport przy pomocy gdal

```
gdal_translate PG:"host=localhost port=5434 user=postgres
dbname=mastering_postgis schema=data_import table=gray_50m_partial
where='filename=\'gray_50m_partial_bl.tif\'' mode=2" -of GTiff -
outsize 50% 50% gray_50m_partial_small.tiff

```

Ekport BBOX ale bez docinania, wybiera całe kafle.

```
gdal_translate PG:"host=localhost port=5434 user=postgres
dbname=mastering_postgis schema=data_import table=gray_50m_partialwhere='
  ST_Intersects(rast, ST_MakeEnvelope(14,49,24,55,4326))'
mode=2" -of GTiff -outsize 50% 50% gray_50m_partial_small.tiff

```

Przykład trzeci - docięcie do granic BBOXa

```
gdal_translate PG:"host=localhost port=5434 user=postgres
dbname=mastering_postgis schema=data_import table=gray_50m_partial
where='ST_Intersects(rast, ST_MakeEnvelope(14,49,24,55,4326))'
mode=2" -of GTiff -projwin 14 55 24 49 poland.tiff

```

7.2 PgRouting

pgRouting rozszerza PostGIS o możliwości routingu (wyznaczania drogi) na podstawie przestrzennej bazy danych. Jest szczególnie przydatny w zastosowaniach związanych z OSM, głównie z powodu gotowych narzędzi do importu danych w formacie gotowym do wykorzystania przez pgRouting (np. przy pomocy osm2po).

Następujące algorytmy obliczeniowe dostępne są w aktualnej wersji rozszerzenia:

- All Pairs Shortest Path, Johnson's Algorithm
- All Pairs Shortest Path, Floyd-Warshall Algorithm
- Shortest Path A*
- Bi-directional Dijkstra Shortest Path
- Bi-directional A* Shortest Path
- Shortest Path Dijkstra
- Driving Distance
- K-Shortest Path, Multiple Alternative Paths
- K-Dijkstra, One to Many Shortest Path
- Traveling Sales Person
- Turn Restriction Shortest Path (TRSP)

7.2.1 Import i preprocessing danych

W pierwszej kolejności spróbujemy poznać strukturę tabel wymaganych przez rozszerzenie pgRouting. W tym celu przygotujemy sobie taką tabelę, na podstawie danych BDOO. Z istniejącej tabeli (warstwy) OT_SKDR_L wybieramy tylko atrybuty opisowe drogi, pomijając identyfikatory IIP.

```
CREATE EXTENSION pgrouting;
CREATE TABLE siec AS SELECT id, geom, x_kod, "katZarzadzania" AS typ, "klasaDrogi"
    AS techniczna, "materialNawierzchni" AS nawierzchnia, "liczbaJezdniDrogi" AS
   jezdnie, numer, "nazwaDrogi" AS nazwa FROM public.ot_skdr_l;
```

Kolejny krok to utworzenie pola dla prędkości maksymalnej, pól dla węzła początkowego i końcowego, a także pola dla kosztu - czy to czasowego, czy kilometrowego.

```
ALTER TABLE public.siec ADD COLUMN vmax integer;
```

Kolejny krok to przeliczenie wartości vmax, przy założeniu prędkości podobnych z wymaganymi przez rozporządzenia i wytyczne KSRG. Dla autostrad i dróg szybkiego ruchu przyjmiemy prędkość 70km/h, dla dróg krajowych 60, wojewódzkich 50, dla pozostałych 40 km/h.

```
UPDATE public.siec SET Vmax = CASE
WHEN techniczna IN('A','S') THEN '70'::int
WHEN typ = 'K' AND techniczna NOT IN ('A','S') THEN '60'::int
WHEN typ = 'W' THEN '50'::int
ELSE '40'::int
END
```

Zwróćmy uwagę na konstrukcję zapytania UPDATE. Wskazujemy że aktualizowaną kolumną będzie vmax, następnie przy pomocy polecenia CASE WHEN oraz THEN tworzymy listę warunków do spełnienia, a także wartość jaka ma zostać wprowadzona, przy tym rzutujemy tą wartość na typ integer. Zakończenie jest konieczne ;)

Kolejny krok to zadbanie o to aby sieć była podzielona na odcinki pomiędzy skrzyżowaniami.

```
CREATE TABLE public.siec_noded AS
SELECT
(ST_Dump(geom)).geom AS geom
FROM (
SELECT ST_Node(geom) AS geom FROM (SELECT ST_Union(geom) AS geom FROM public.siec)
    a
) b;
```

Zajrzyjmy do tej tabeli - jak widać, utraciliśmy dane o parametrach drogi.

Połączmy dane źródłowe. Następnie utworzymy kolumny source i target, oraz zbudujemy topologię.

```
ALTER TABLE public.siec_noded ADD COLUMN source integer;
ALTER TABLE public.siec_noded ADD COLUMN target integer;
SELECT pgr_createTopology('public.siec_noded', 1, 'geom');
```

```
ALTER TABLE public.siec_noded ADD COLUMN koszt float;
UPDATE public.siec SET koszt = (ST_Length(geom)/1000)/"vmax"*60
```

W ten sposób przygotowaliśmy sobie kompletną tabelę dla routingu. Spróbujemy teraz wykonać pierwsze zapytanie pgRouting. Należy pamiętać że mają one formę tzw. inner queries, a więc są to dwa zapytania, z których jedno zagnieżdżone jest jako argument funkcji pgr.

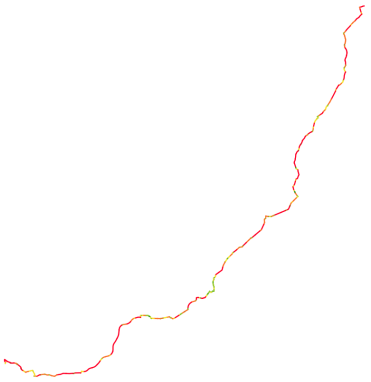
7.2.2 Zapytania

Najbardziej znanym algorytmem wyszukiwania drogi jest algorytm Dijkstry. Argumentami dla funkcji liczącej są punkt początkowy (source), końcowy (target), oraz wartość kosztu wyrażona w dowolnych jednostkach.

```
SELECT * FROM pgr_dijkstra('SELECT id,source,target,koszt AS cost FROM public.
    siec_noded', 84400, 6549, false);
```

Jako wynik zwracane są kolejne segmenty drogi wyznaczonej jako ta o najniższym koszcie. Należy zwrócić uwagę na to że nie jest zwracana geometria dla najkrótszej drogi. Konieczne jest więc złączenie tabel na podstawie identyfikatora krawędzi (edge). Z pomocą przychodzi nam LEFT JOIN:

```
SELECT id,
geom,
route.cost
FROM public.siec_noded e
JOIN (SELECT * FROM pgr_dijkstra('SELECT id,source,target,koszt AS cost FROM
    public.siec_noded', 84400, 6549, false)) AS route
ON e.id=route.edge;
```



Rysunek 7.1: Algorytm Dijkstry - wynik

Kolejnym algorytmem który wart jest naszej uwagi jest "Driving Distance". Tu jako argument podajemy punkt początkowy, tabelę kosztów, oraz koszt końcowy. Na przykład dla centrum Kielc, zasięg sieci drogowej do osiągnięcia w czasie 40 minut:

```
SELECT id,
geom,
route.cost
FROM public.siec_noded e
JOIN (SELECT * FROM pgr_drivingDistance('SELECT id,source,target,koszt AS cost
   FROM public.siec_noded', 40874, 40)) AS route
ON e.id=route.edge;
```

Wynikiem są wszystkie drogi w zasięgu. Warto zwrócić uwagę, ta funkcja potrafi również zwrócić zasięg dla wielu punktów naraz.

```
CREATE TABLE zasieg AS SELECT id,
geom,
route.cost,
route.from_v,
route.agg_cost
FROM public.siec_noded e
```

```
JOIN (SELECT * FROM pgr_drivingDistance('SELECT id,source,target,koszt AS cost
   FROM public.siec_noded', ARRAY[40874,40900], 240)) AS route
ON e.id=route.edge;
```

Jedyną istotną zmianą jest tu użycie ARRAY zamiast pojedyńczego identyfikatora, oraz wskazanie jako wyniku również route.from_v

Po wyliczeniu segmentów dostępnych możemy wyliczyć dodatkową warstwę otoczki wklęsłej.

```
SELECT ST_ConcaveHull(ST_Collect(geom),0.6) FROM szkolenie.catchment_final
```

Spis treści