GIS support

Materiały szkoleniowe
SQL w PostgreSQL w praktyce
(poziom średniozaawansowany)

MINISTERSTWO KLIMATU

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

Spis treści

Blok 1 - Wprowadzenie

PostgreSQL

Zgodnie z informacjami zawartymi na stronie głównej projektu PostgreSQL to potężny, obiektowo-relacyjny system bazy danych o otwartym kodzie źródłowym, z ponad 30-letnim aktywnym rozwojem, dzięki któremu zyskał dobrą reputację dzięki niezawodności, użyteczności funkcji i wydajności. Sam projekt wyewoluował z innego projektu prowadzonego na uniwersytecie Berkeley o nazwie Ingres. Nieoficjalne źródła podają, że nazwa powstała jako żart osób tworzących projekt, polegający na zastosowaniu nazewnictwa fragmentów ciągu (prefix, infix, postfix) do nazwy ingres, której naturalnym następstwem będzie postgres. Zgodnie z dokumentacją projektu w roku 1996 twórcy postanowili zaznaczyć zgodność bazy danych ze standardem SQL dodając odpowiedni postfix do nazwy. W wyniku spłaszczenia podwójnego 's' nazwę wymawiamy jako jeden wyraz - Postgresql, nie literując dodatkowo SQL (wymowa postgre-eS-Qu-eL jest błędna), oraz skracamy do Postgres przez wzgląd na poprzednie nazwy, a nie do Postgre jak by wynikało z odcięcia postfixu SQL - takie skracanie jest błędne.

PostGIS

PostGIS, jak podaje główna strona projektu to rozszerzenie przestrzennej bazy danych dla obiektowo-relacyjnej bazy danych PostgreSQL. Dodaje obsługę obiektów geograficznych, umożliwiając wykonywanie zapytań o lokalizację w języku SQL.

PostGIS dodaje dodatkowe typy (geometria, geografia, raster i inne) do bazy danych PostgreSQL. Dodaje również funkcje, operatory i rozszerzenia indeksów, które mają zastosowanie do tych typów przestrzennych. Te dodatkowe funkcje, operatory, powiązania indeksów i typy zwiększają moc bazy PostgreSQL, czyniąc z niej szybki, bogaty w funkcje i niezawodny system zarządzania przestrzenną bazą danych.

Blok 2 - Instalacje i konfiguracje

Instalacja serwera bazy danych oraz rozszerzeń

Używane rozszerzenia

W toku szkolenia będziemy używali trzech rozszerzeń bazy danych:

- **postgis** - rozszerzenie bazy o obsługę danych przestrzennych
- **hstore** - typ danych oparty o strukturę klucz=wartość
- **pg_stat_statements** - rozszerzenie o statystyki pozwalające na analizę planów wykonywanych zapytań

Instalacja w systemach Linux

Większość dystrubucji systemów Linux ma gotowe prekompilowane paczki zarówno dla bazy danych PostgreSQL jak i dla poszczególnych jej rozszerzeń. W systemach opartych na `apt` bazę instalujemy poleceniem

```
apt install postgresql postgis
```

Instalacja w systemach Windows

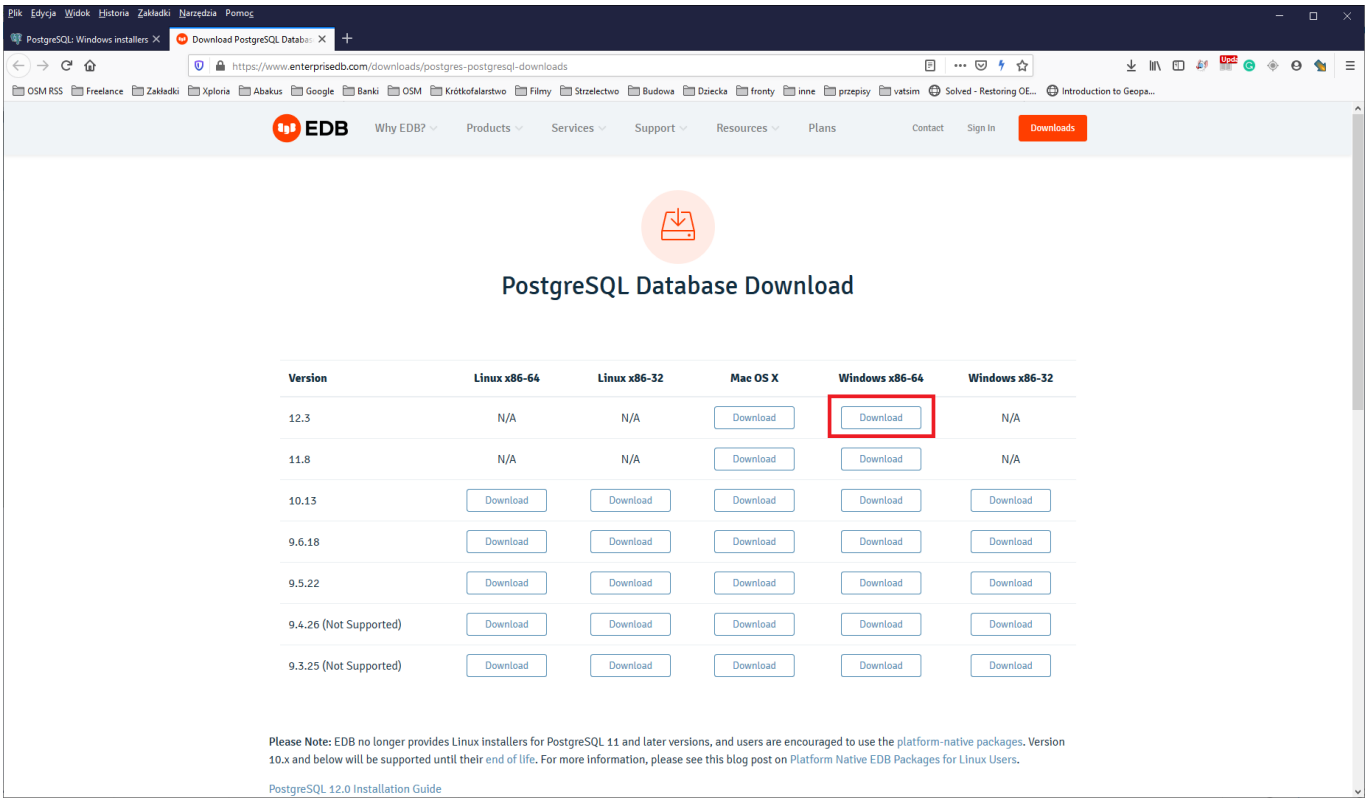
W systemach operacyjnych Windows, zgodnie z oficjalną dokumentacją bazę danych wraz z rozszerzeniami należy instalować używając instalatorów dostarczanych przez EnterpriseDB. Instalatory te zawierają:

- serwer bazy danych
- pgAdmin - narzędzie do zarządzania oraz pracy na bazie danych
- StackBuilder - manager pakietów pozwalający na pobieranie oraz instalowanie dodatkowych rozszerzeń i sterowników

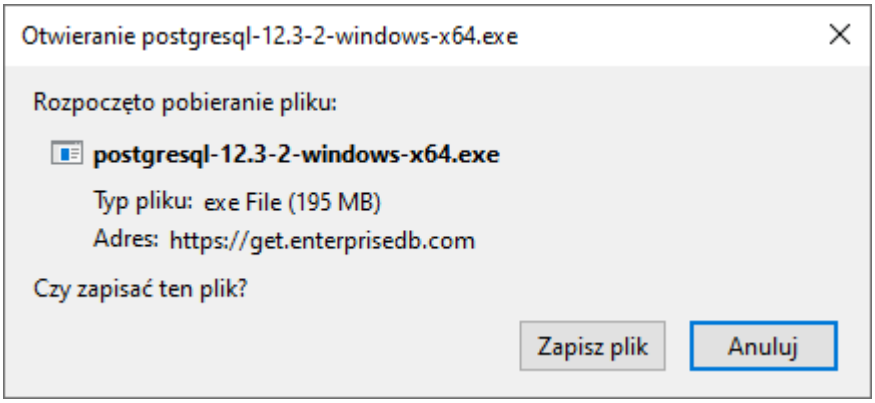
Instalator może być uruchomiony w trybie interaktywnym oraz cichym.

Instalator możemy pobrać ze strony https://www.enterprisedb.com/downloads/postgres-postgresql-downloads, dostępny jest on również w folderze z materiałami szkoleniowymi (postgresql-12.3-2-windows-

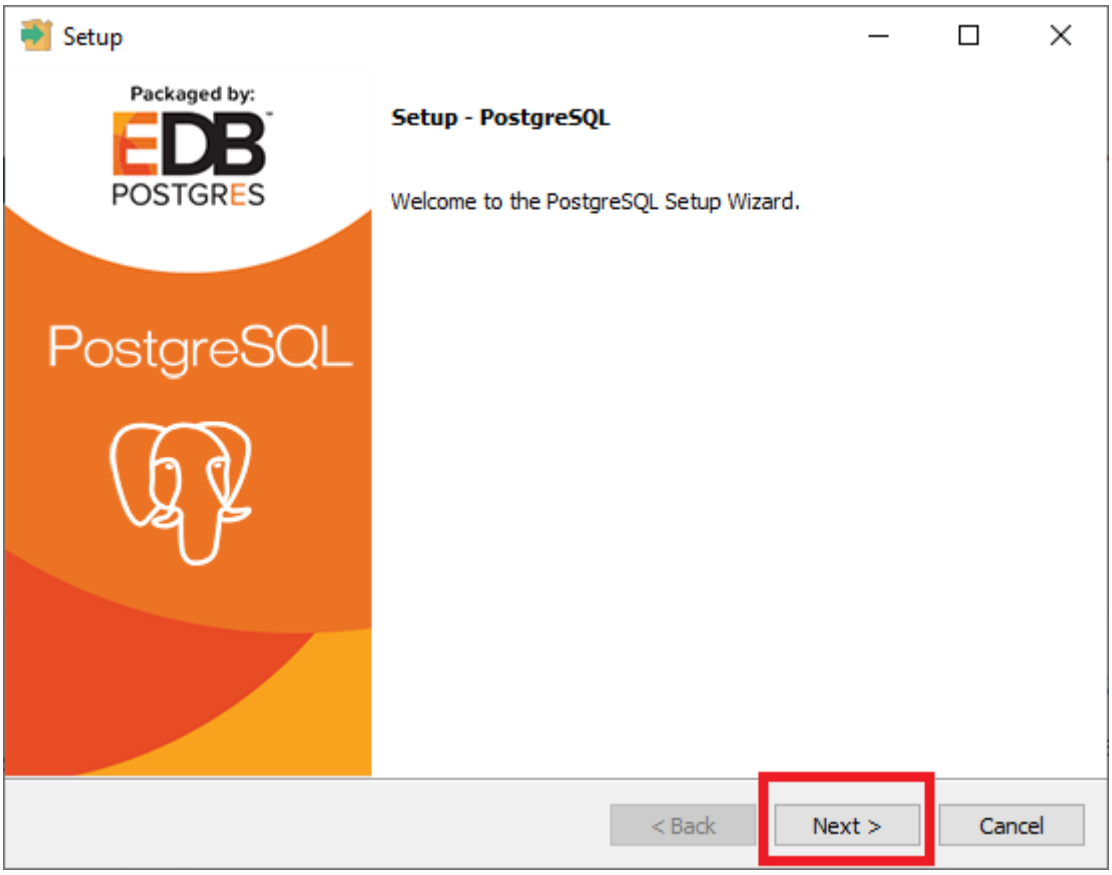
x64.exe). Szkolenie przeprowadzimy na wersji Postgresql 12.



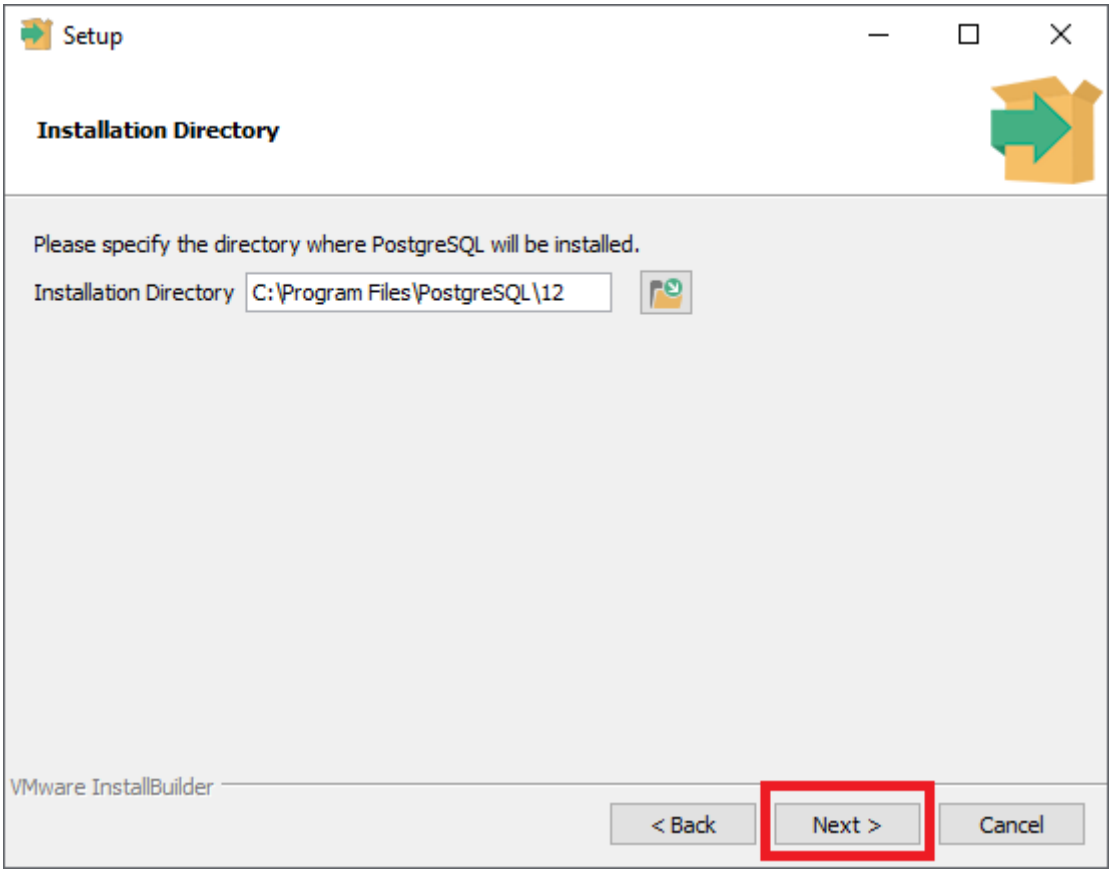
Plik instalatora pobieramy i zapisujemy lokalnie



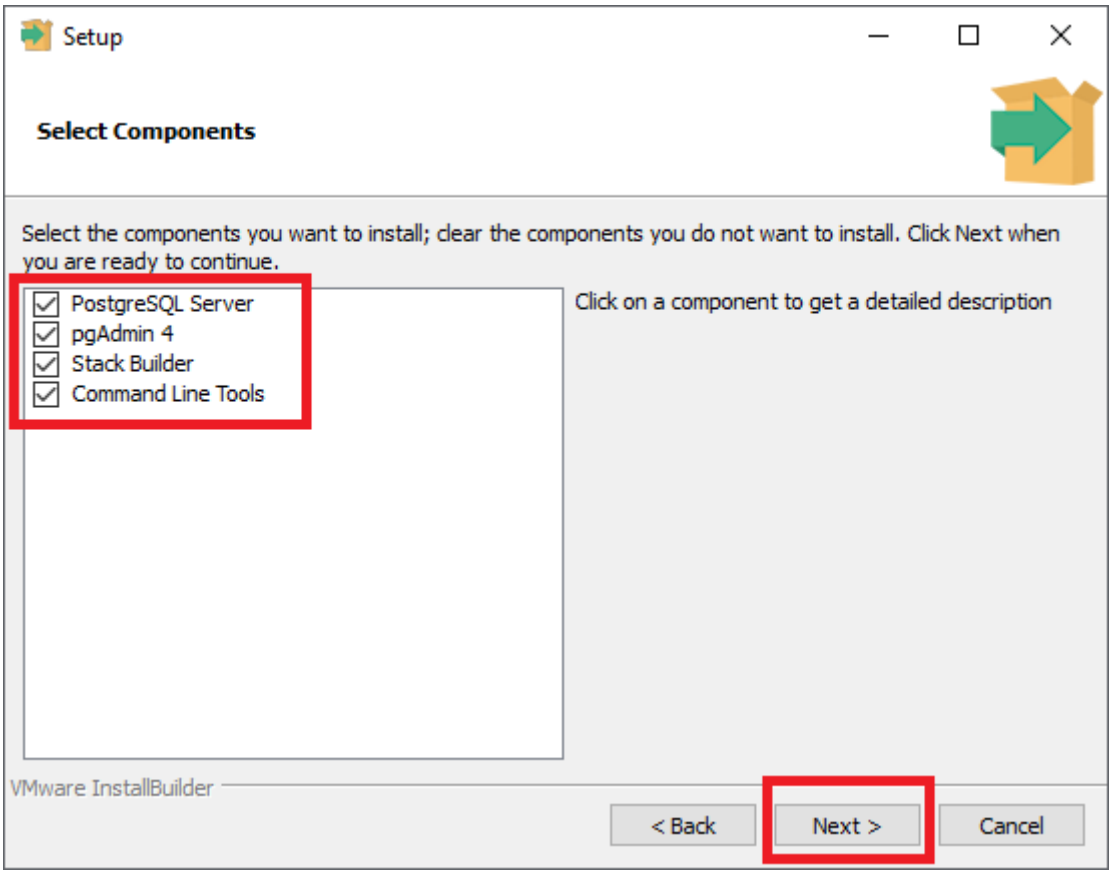
Po czym uruchamiamy instalator lokalnie. Możliwe że będzie trzeba potwierdzić uprawnienia administratora w kontroli konta użytkownika.



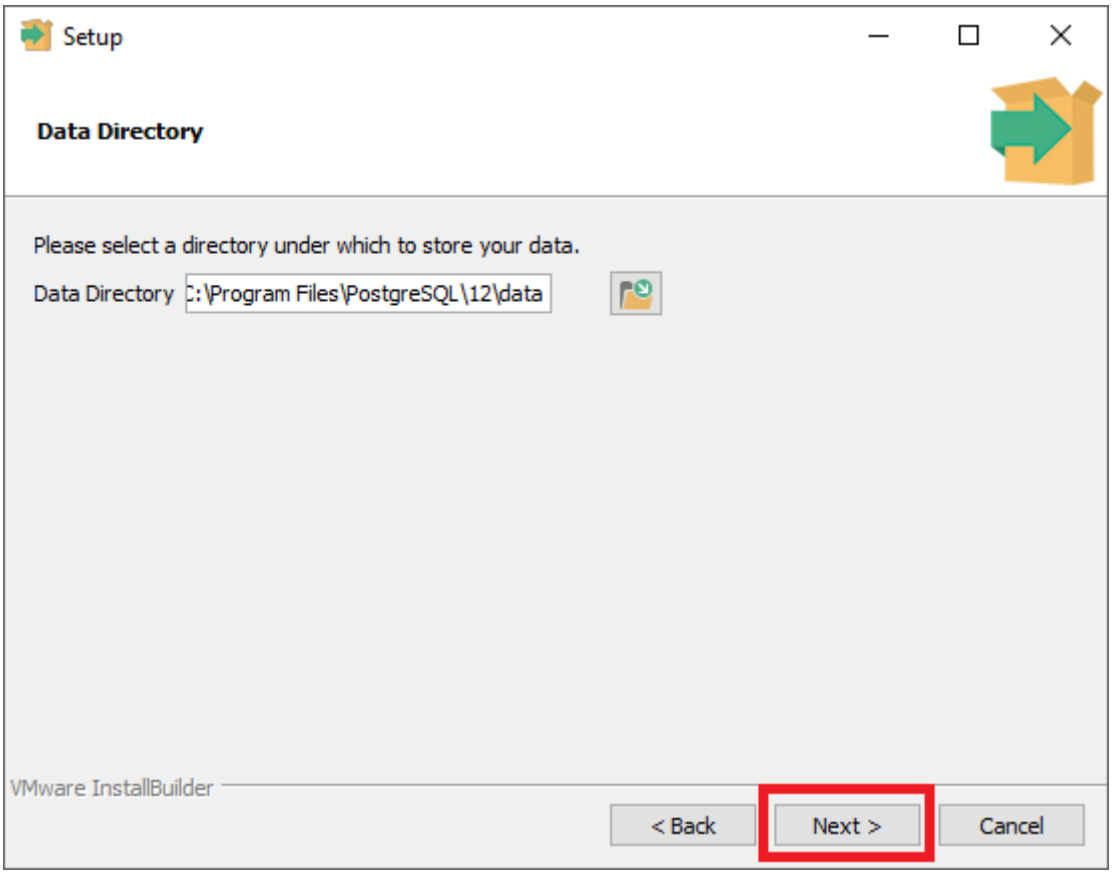
W kolejnym oknie podajemy ścieżkę w której instalator zainstaluje bazę danych - na potrzeby szkolenia sugerujemy pozostawienie wartości domyślnej. W przypadku zmiany tej wartości należy zapamiętać nową ścieżkę ponieważ będzie ona konieczna do podania ponownie później.



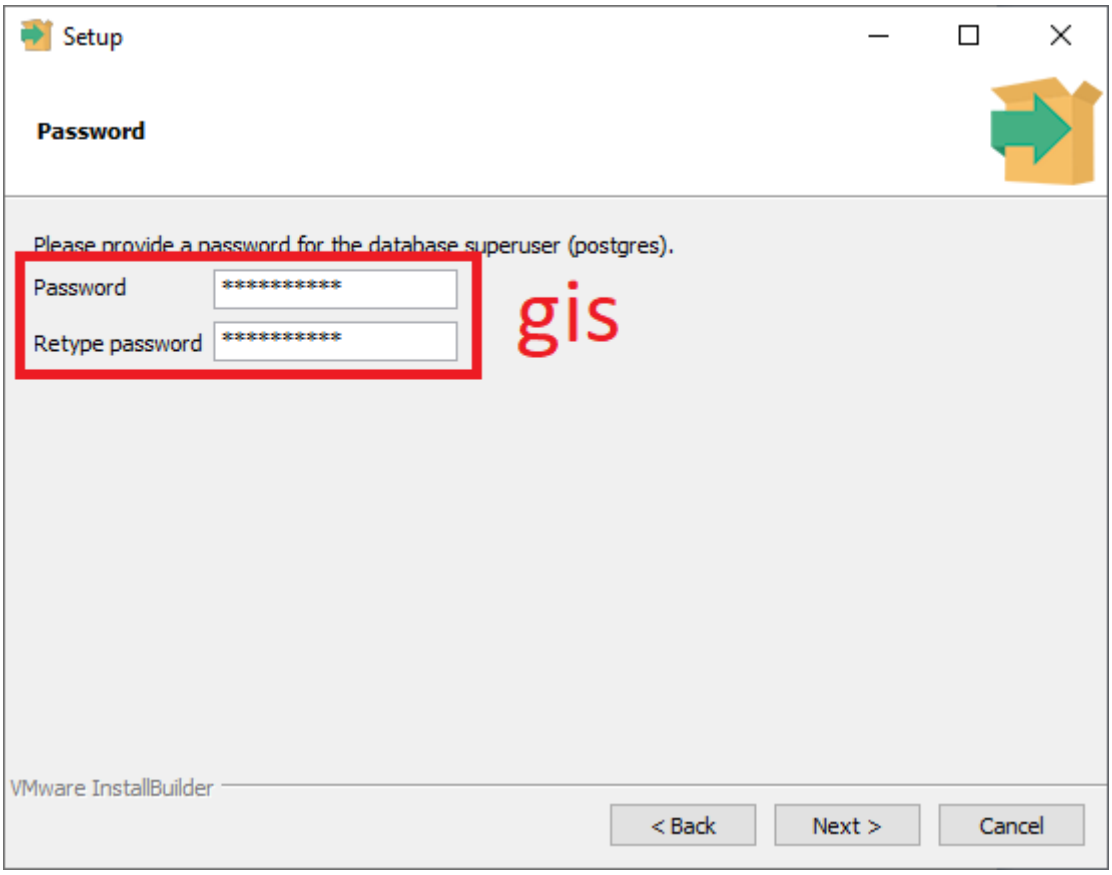
W kolejnym kroku wybieramy wszystkie komponenty do instalacji (zaznaczamy wszystkie pola).



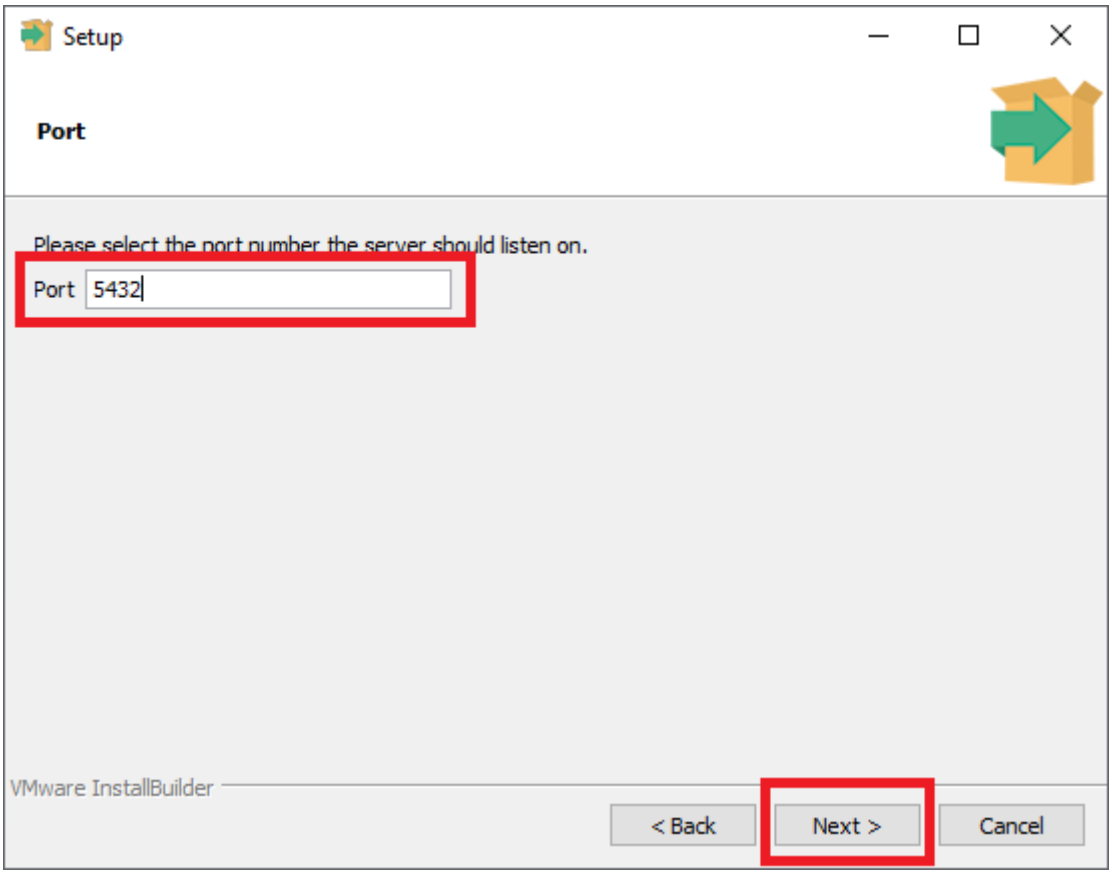
Kolejny krok pozwala na zdefiniowanie folderu w którym przechowywane będą dane. W tym przypadku również można zostawić wartość domyślną.



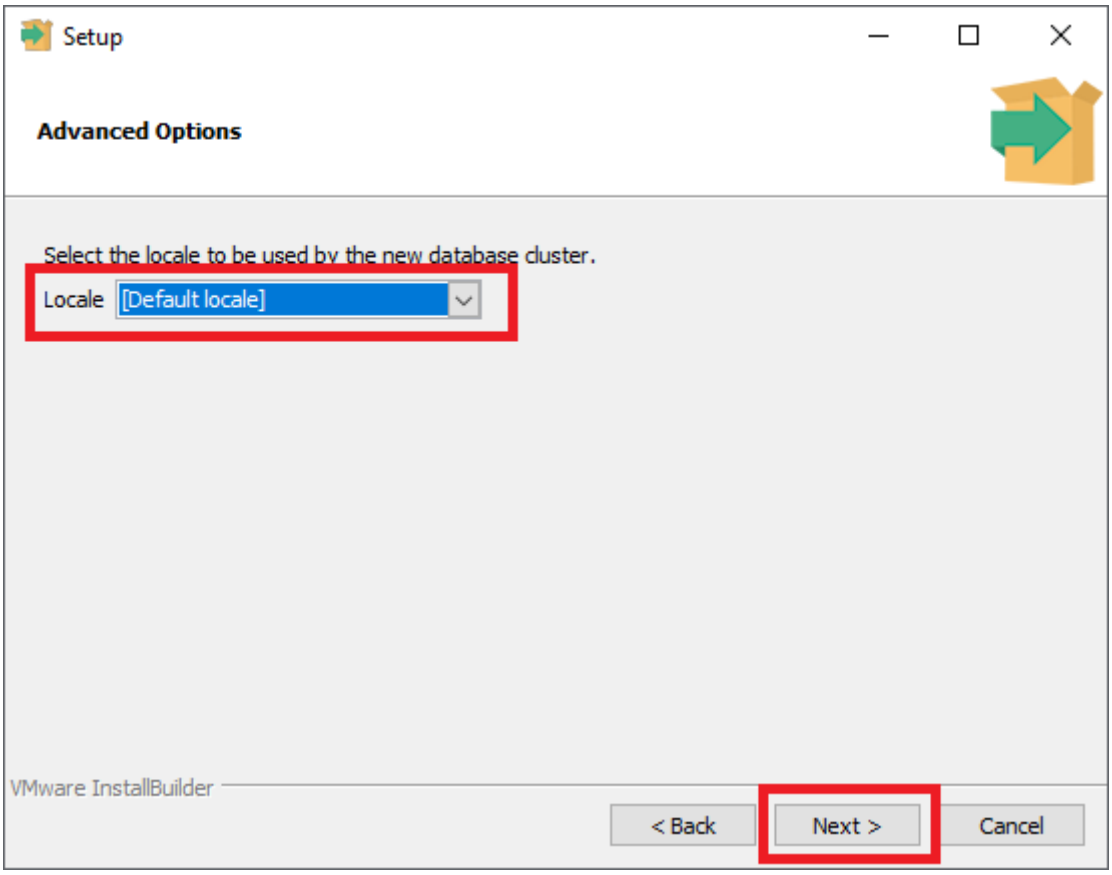
Kolejny krok to ustawienie hasła do bazy danych - na potrzeby szkolenia sugerujemy ustawienie `gis`. W przypadku ustawienia innego hasła należy go bezwzględnie zapamiętać - w przeciwnym razie konieczna będzie ponowna instalacja.



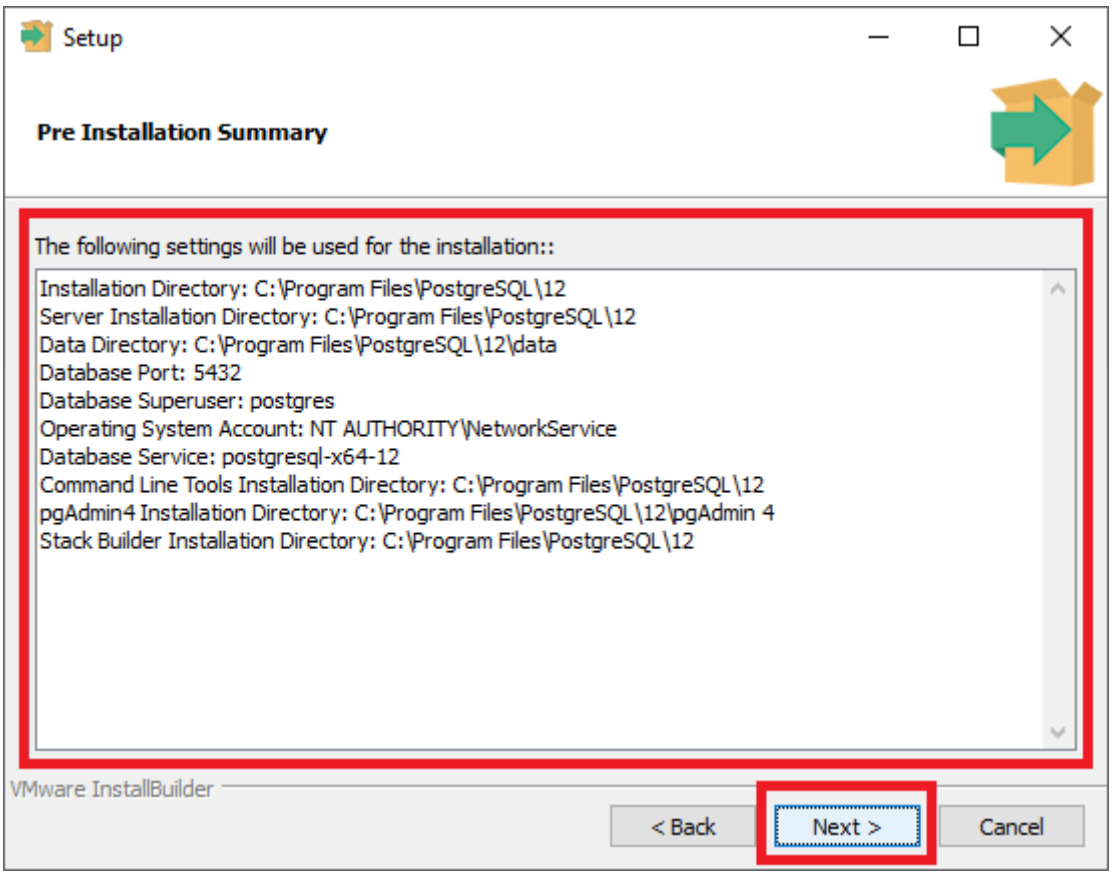
Kolejny krok umożliwia zmianę domyślnego portu, na którym nasłuchuje baza - znów sugerujemy pozostawienie wartości domyślnej 5432.



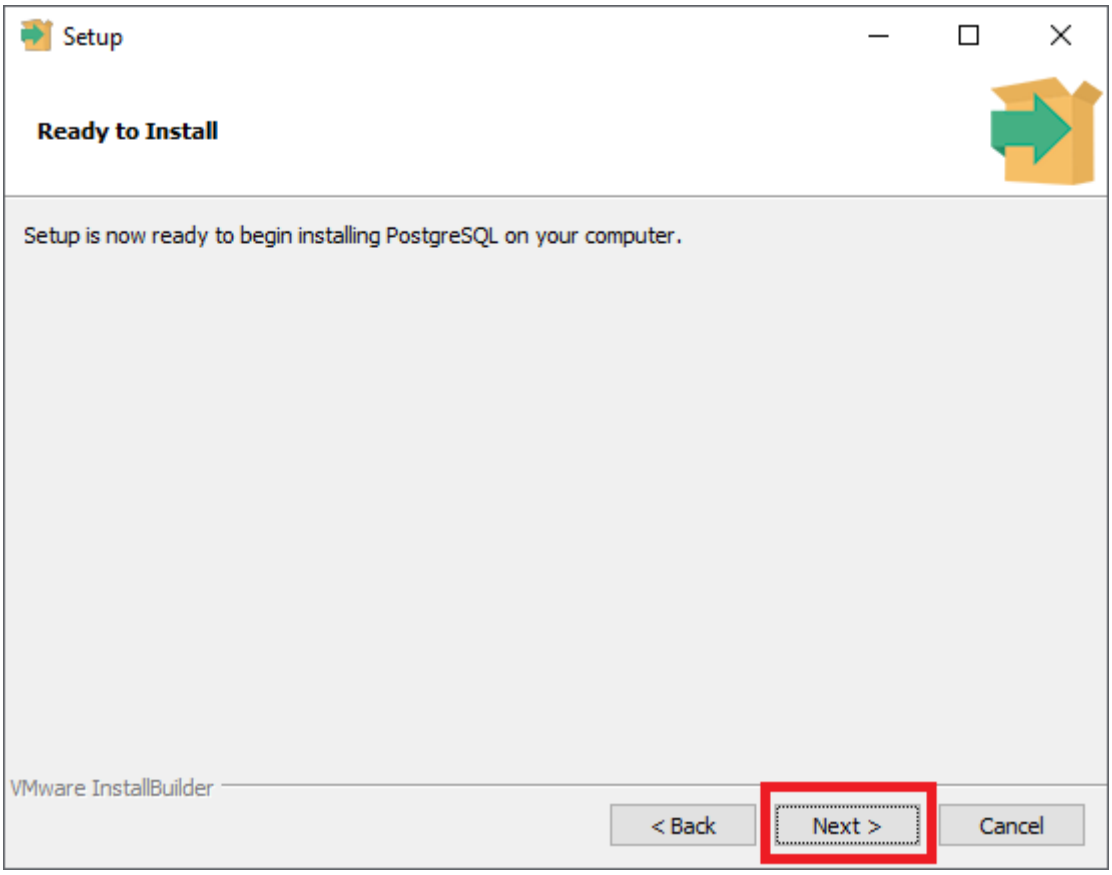
Ustawienia lokalne systemu również zostawiamy na wartości domyślnej.



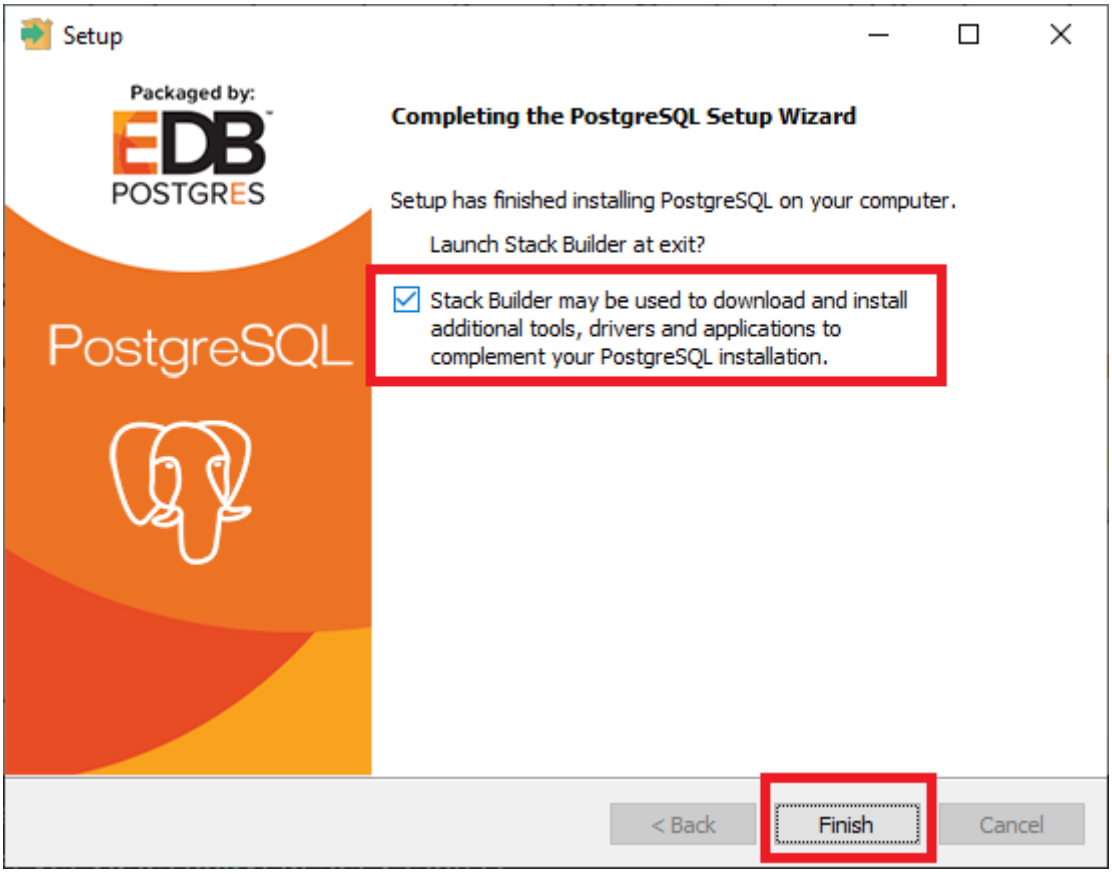
Przeglądamy podsumowanie i jeśli wszystko się zgadza wybieramy opcję `dalej`.



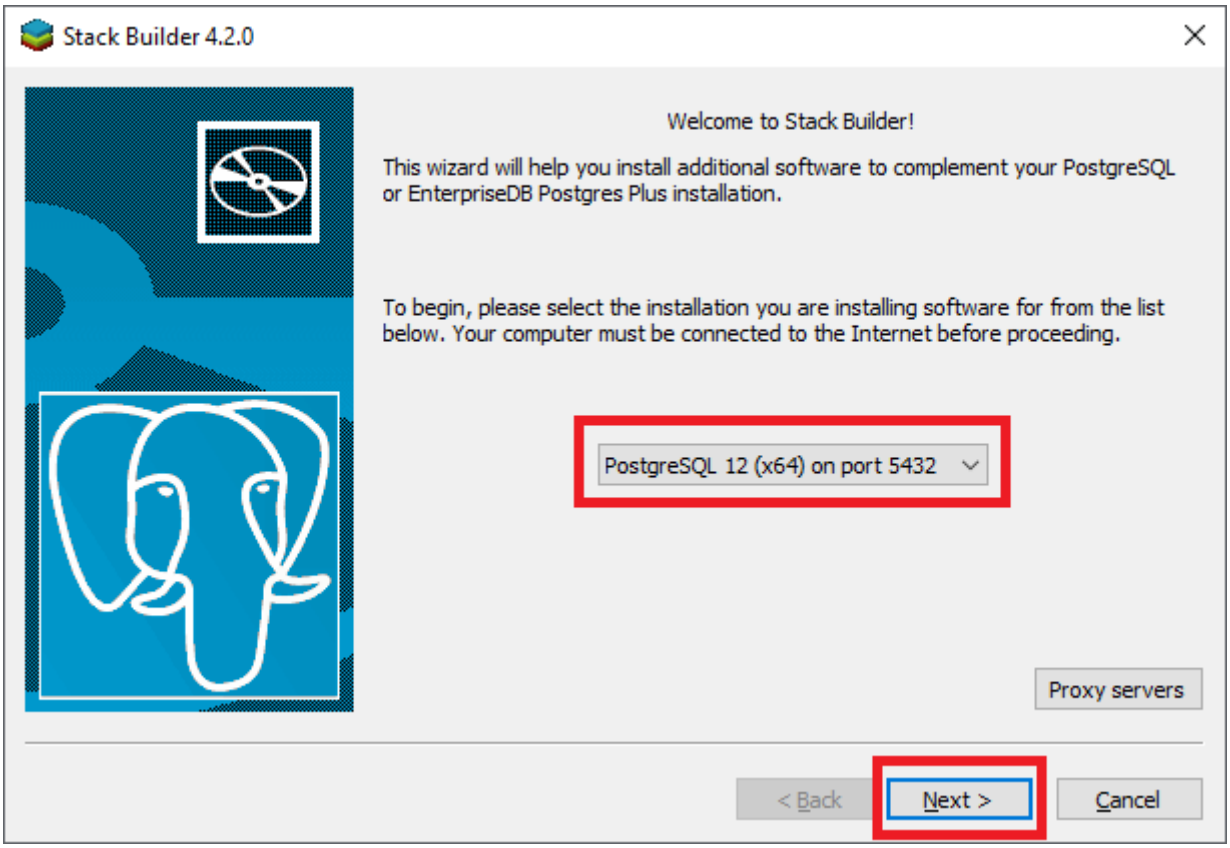
Instalator sprawdzi ustawienia i jeśli wszystko przebiegło prawidłowo wyświetli informację, że może przystąpić do instalacji, wybieramy `dalej`.



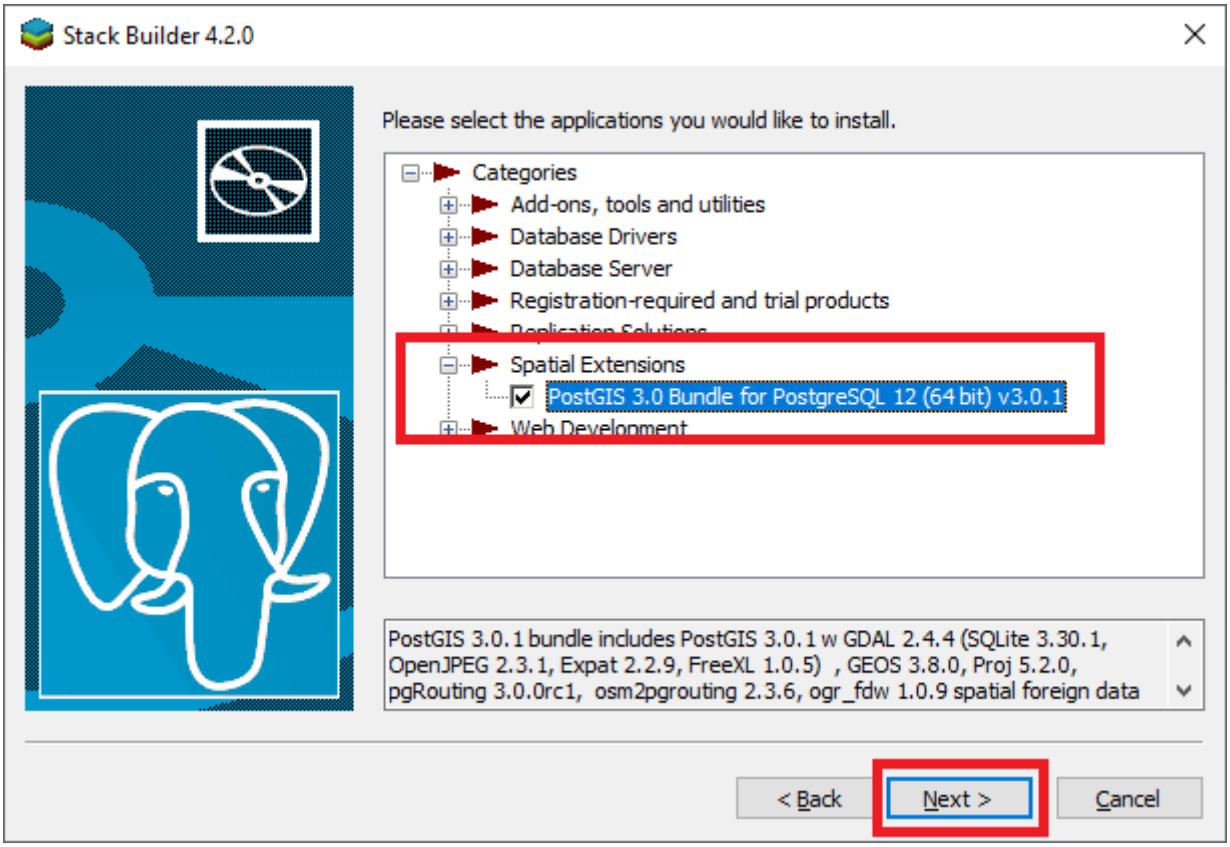
Po zainstalowaniu bazy danych instalator zapyta czy ma uruchomić StackBuilder - narzędzie pozwalające na instalowanie dodatkowych sterowników czy rozszerzeń. Sprawdzamy czy okienko wyboru jest zaznaczone i wybieramy `Finish`.



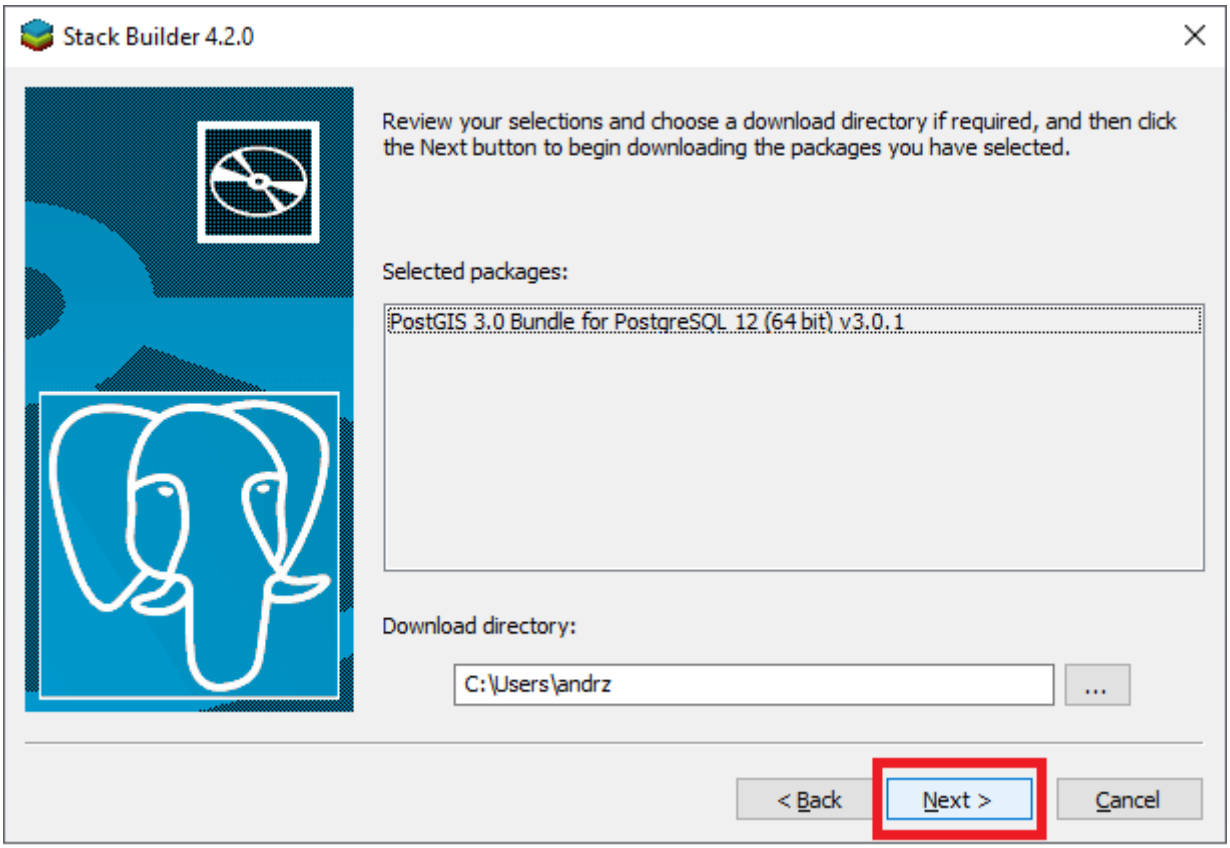
W kolejnym kroku wyświetlony zostanie StackBuilder - wybieramy bazę danych do której będziemy instalowali dodatki. W oknie wyboru wybieramy `PostgreSQL 12 on port 5432` i klikamy przycisk `Next`.



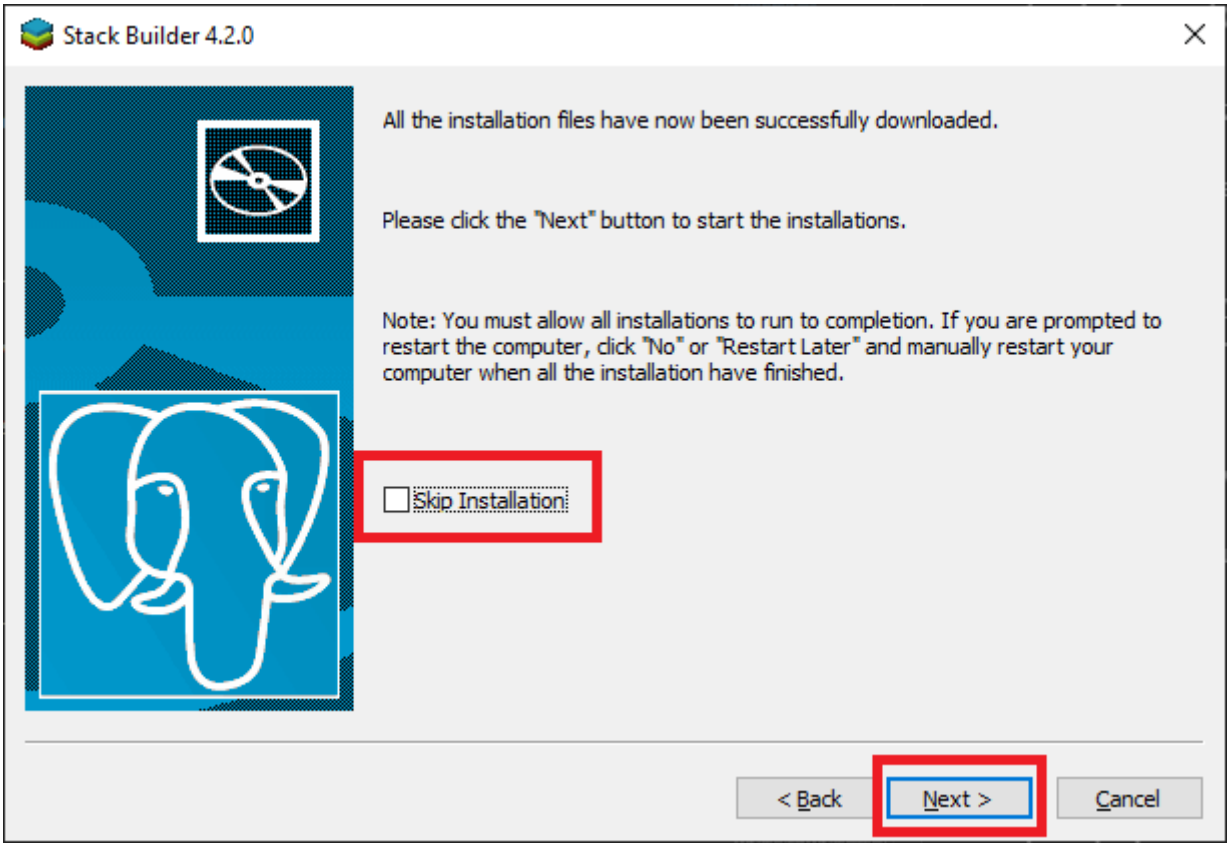
W kolejnym kroku rozwijamy gałąź `Spatial extensions` i zaznaczamy okno wyboru przy pozycji `PostGIS 3.0`.



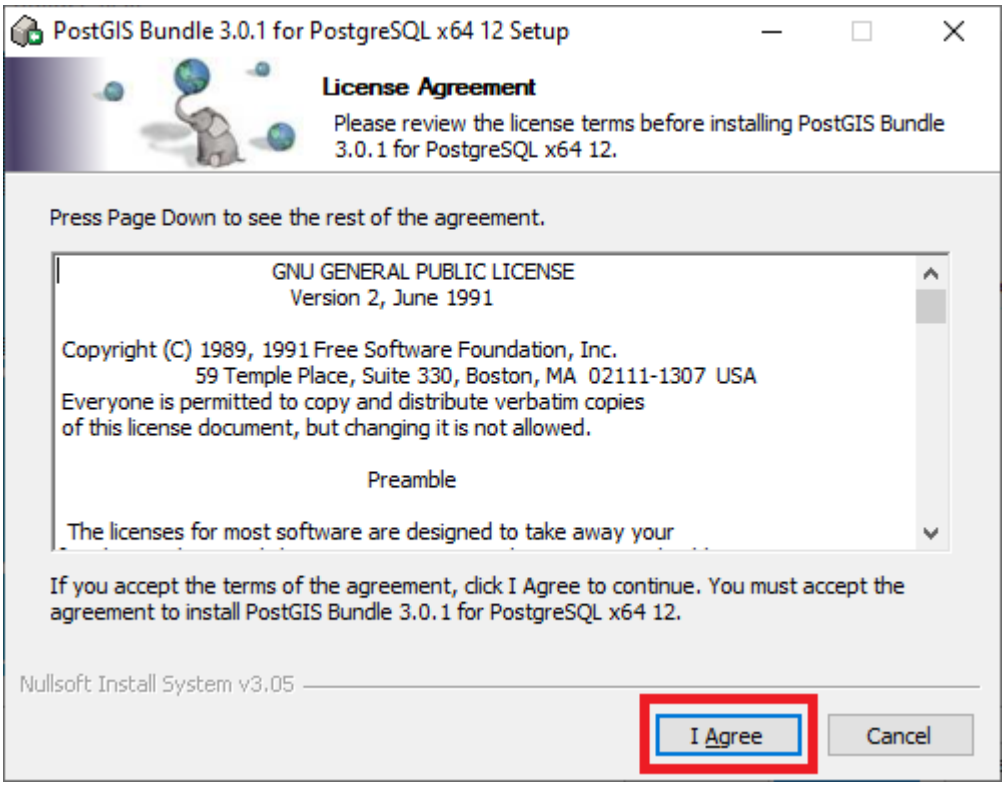
Kolejny krok to podsumowanie i wybór folderu w którym zapisane zostaną pobrane rozszerzenia.



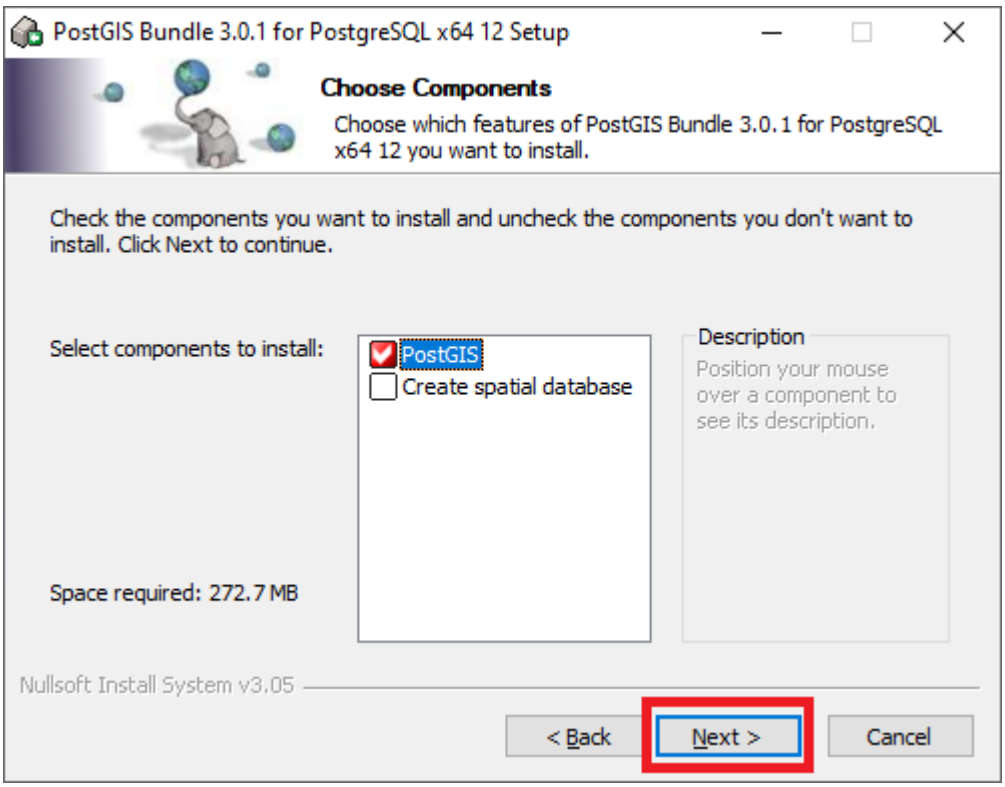
W kolejnym kroku upewniamy się, że okienko wyboru `Skip Installation` **nie** jest zaznaczone i wybieramy `Next`.



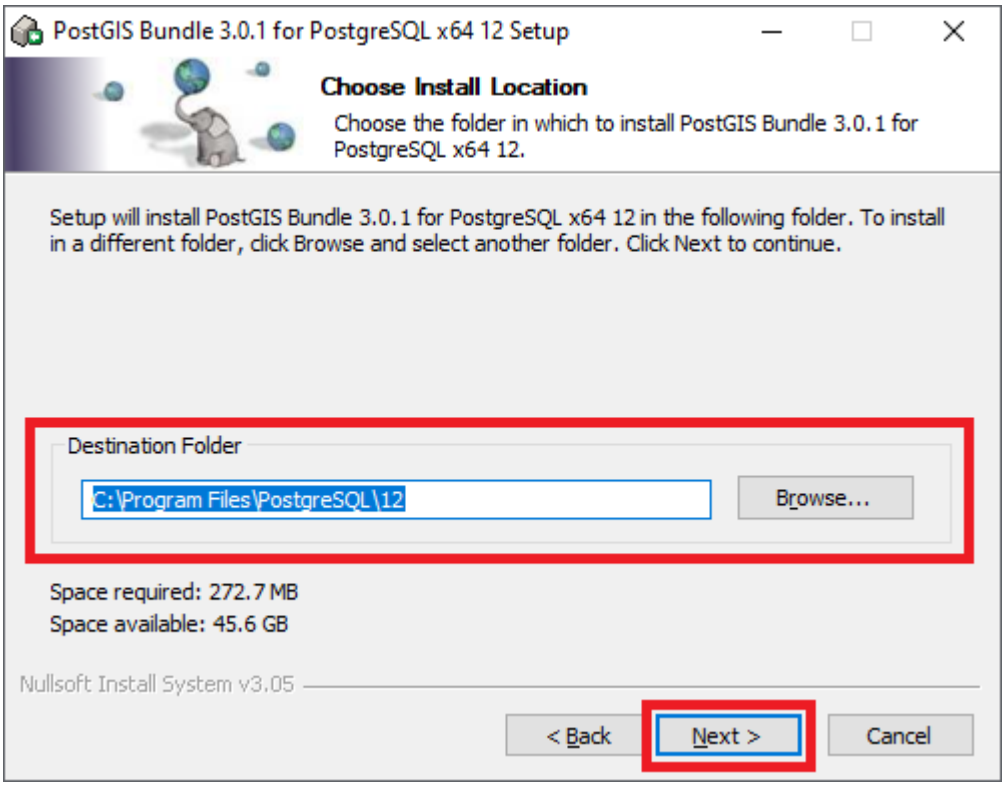
Po pobraniu pakietu instalacyjnego PostGIS zostanie on uruchomiony. W pierwszym kroku potwierdzamy zapoznanie się z licencją.



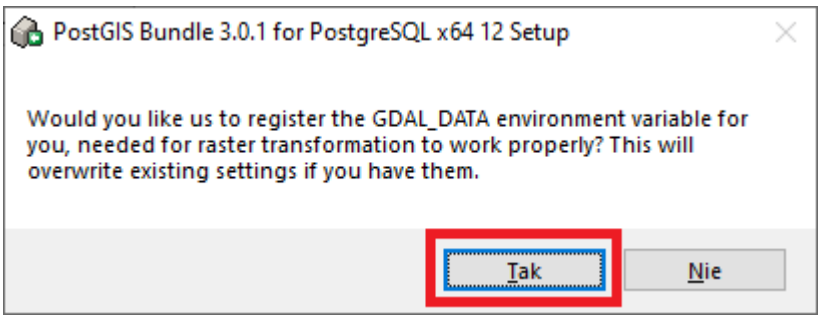
W drugim kroku sprawdzamy czy opcja PostGIS jest zaznaczona i wybieramy `Next`.



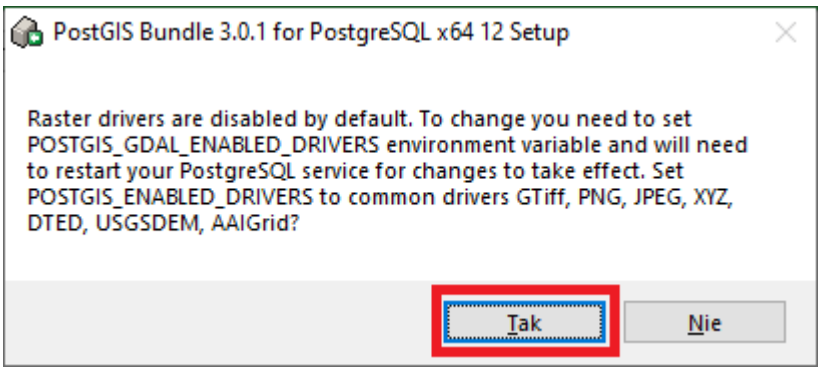
W kolejnym wybieramy lokalizację instalacji. Jeśli domyślna ścieżka instalacji bazy danych została zmieniona tutaj należy również podać zmienioną ścieżkę.



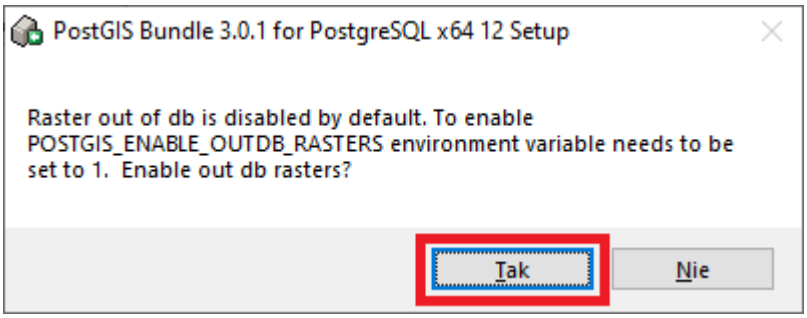
Następnie potwierdzamy zarejestrowanie zmiennej środowiskowej GDAL_DATA



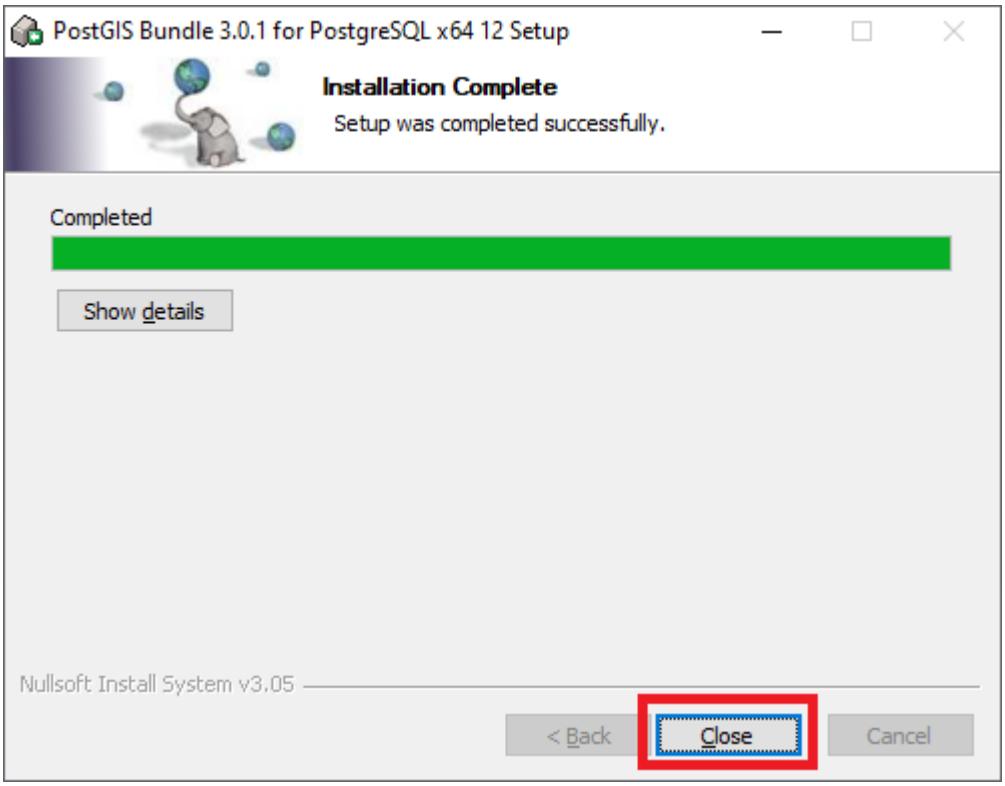
Włączamy sterowniki GDAL do danych rastrowych



i zezwalamy na eksporty rastrów z bazy danych



Jeśli instalacja zakończy się poprawnie powinniśmy zobaczyć okno podsumowania.

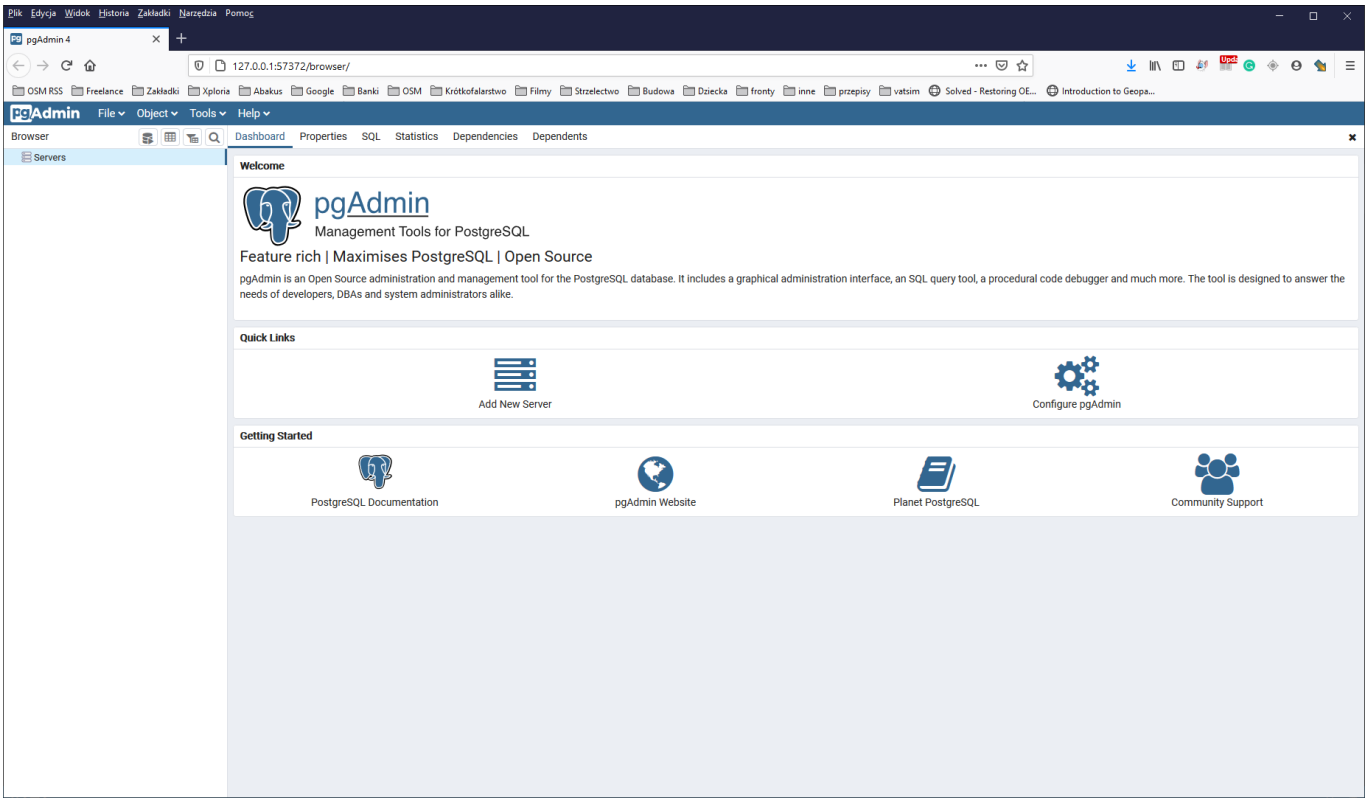


Konfiguracja aplikacji pgAdmin

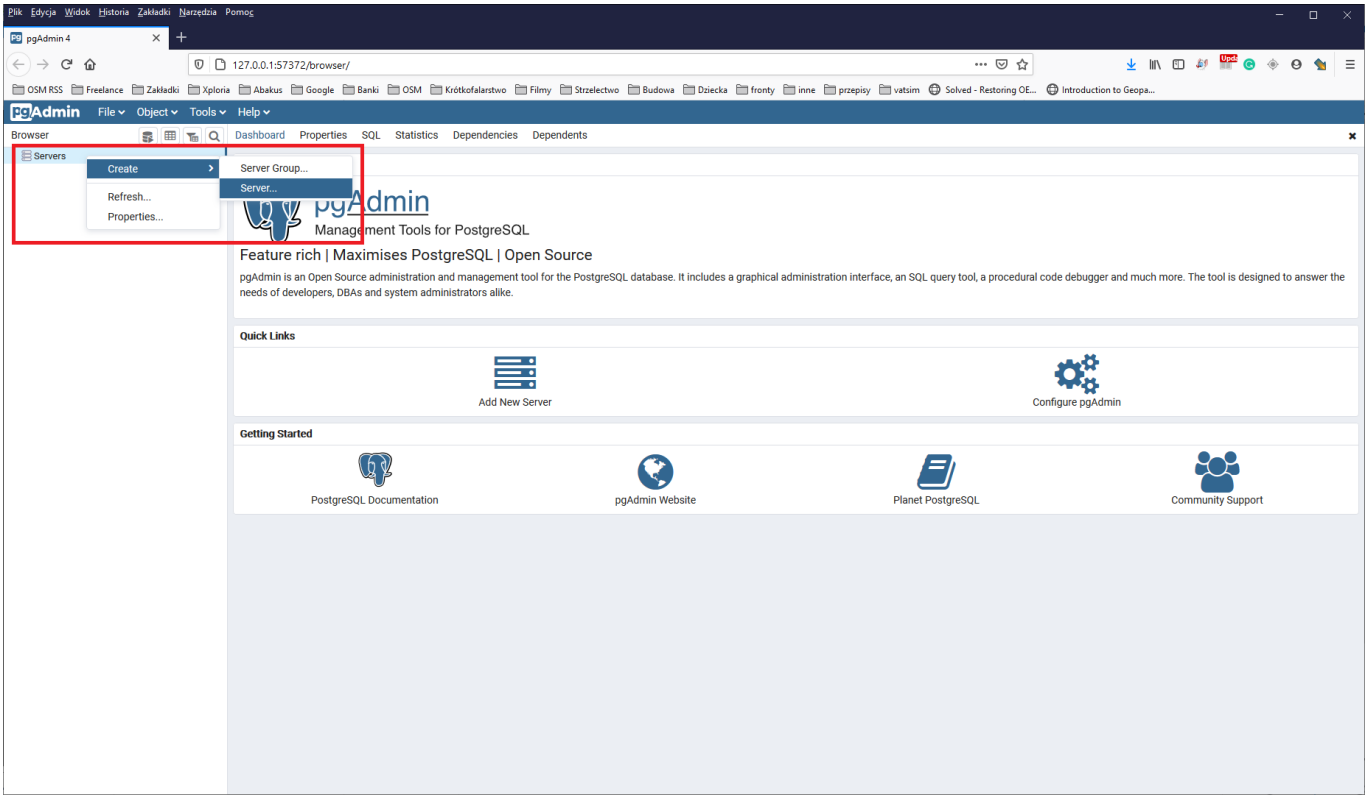
pgAdmin to aplikacja pozwalająca na zarządzanie bazą danych oraz danymi w tej bazie. Do poprawnego działania konieczne jest wstępne jej skonfigurowanie.

Aplikację odnajdujemy w menu Start systemu Windows i uruchamiamy.

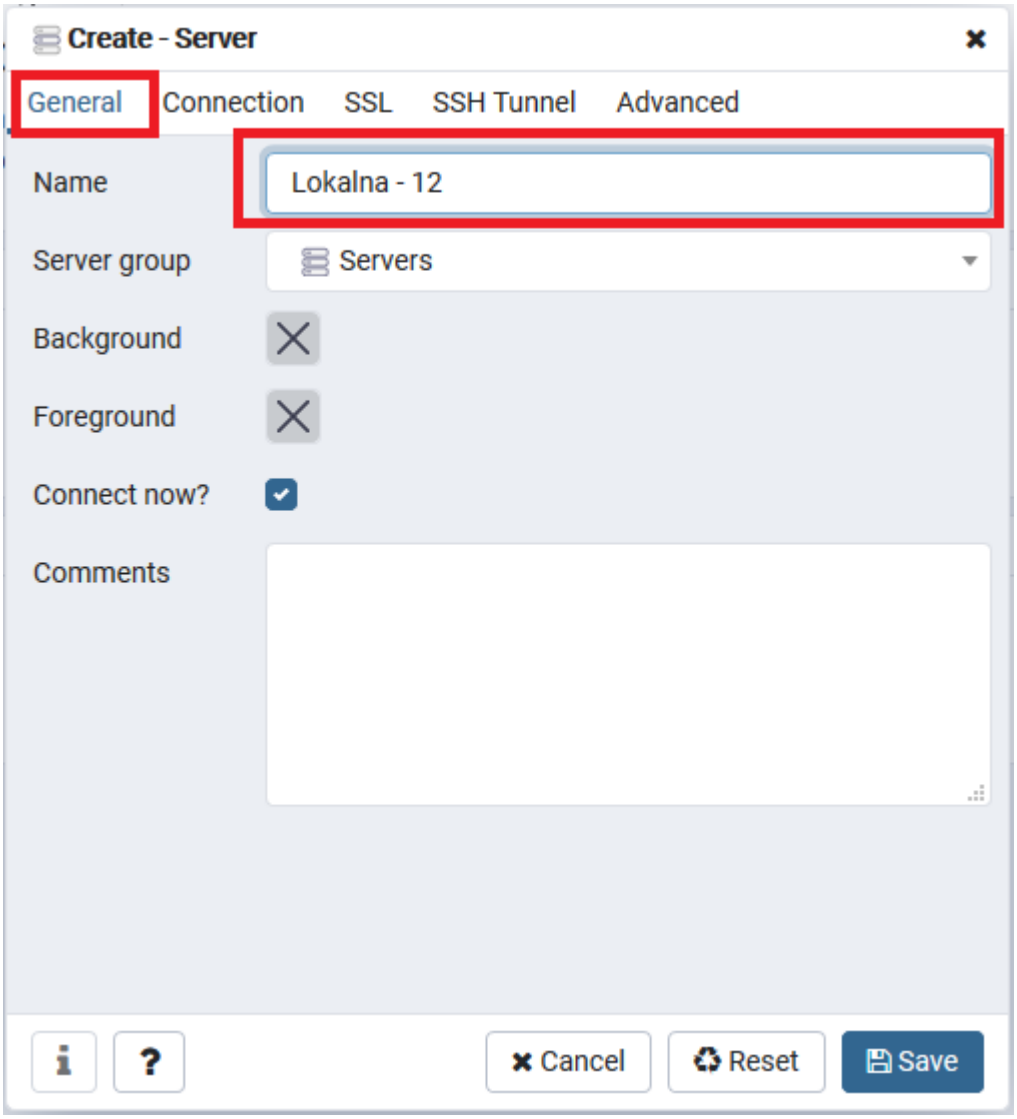
Uruchomiona zostanie przeglądarka internetowa, a przy pierwszym uruchomieniu aplikacja poprosi o wpisanie hasła zabezpieczającego - sugerujemy ponownie użycie hasła `gis`. Wyświetlone zostanie okno aplikacji



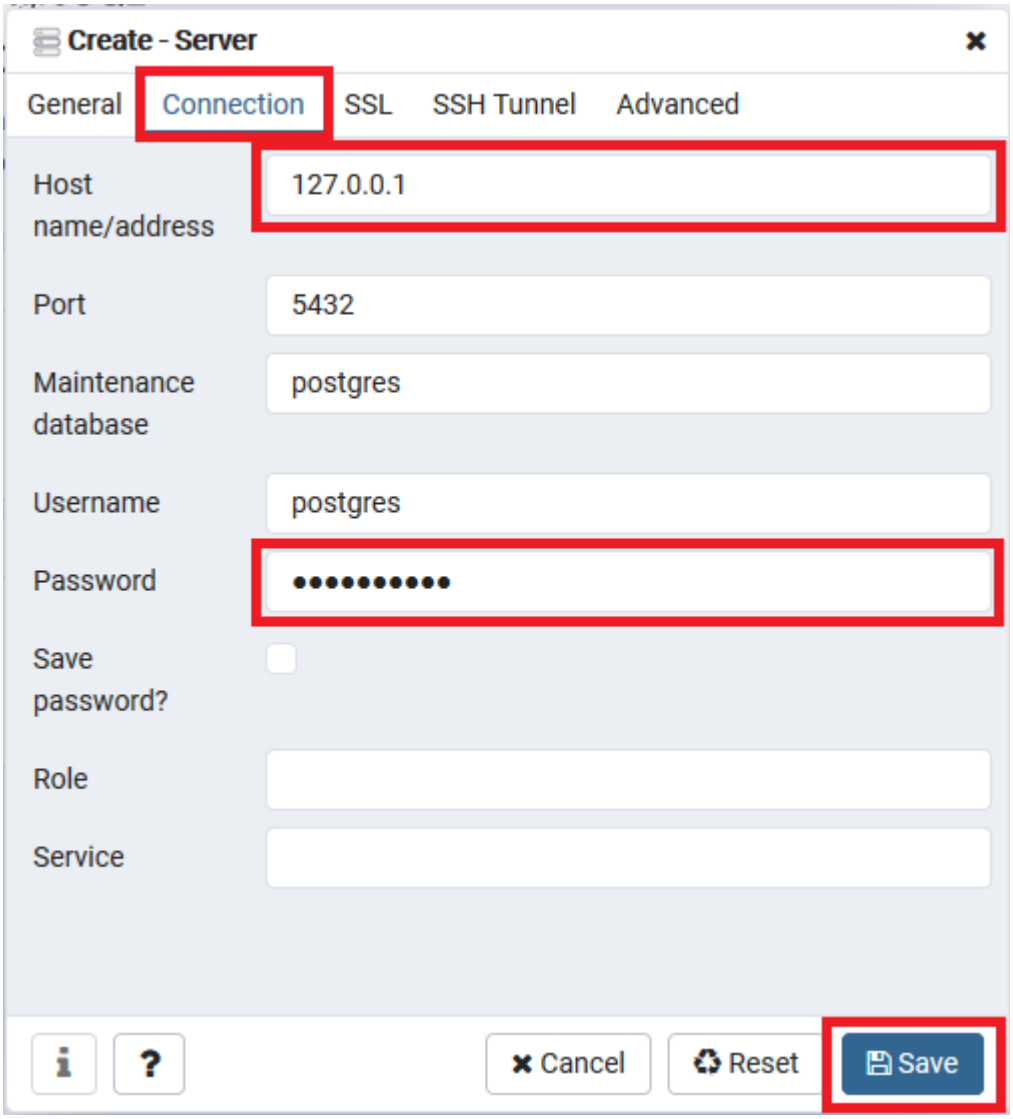
Aplikacja pozwala na zarządzanie równocześnie wieloma bazami danych umieszczonymi na wielu serwerach. W kolejnym kroku połączymy aplikację z bazą danych zainstalowaną w poprzednim kroku szkolenia. W tym celu klikamy prawym przyciskiem myszy na pozycji servers i wybieramy `create > server`



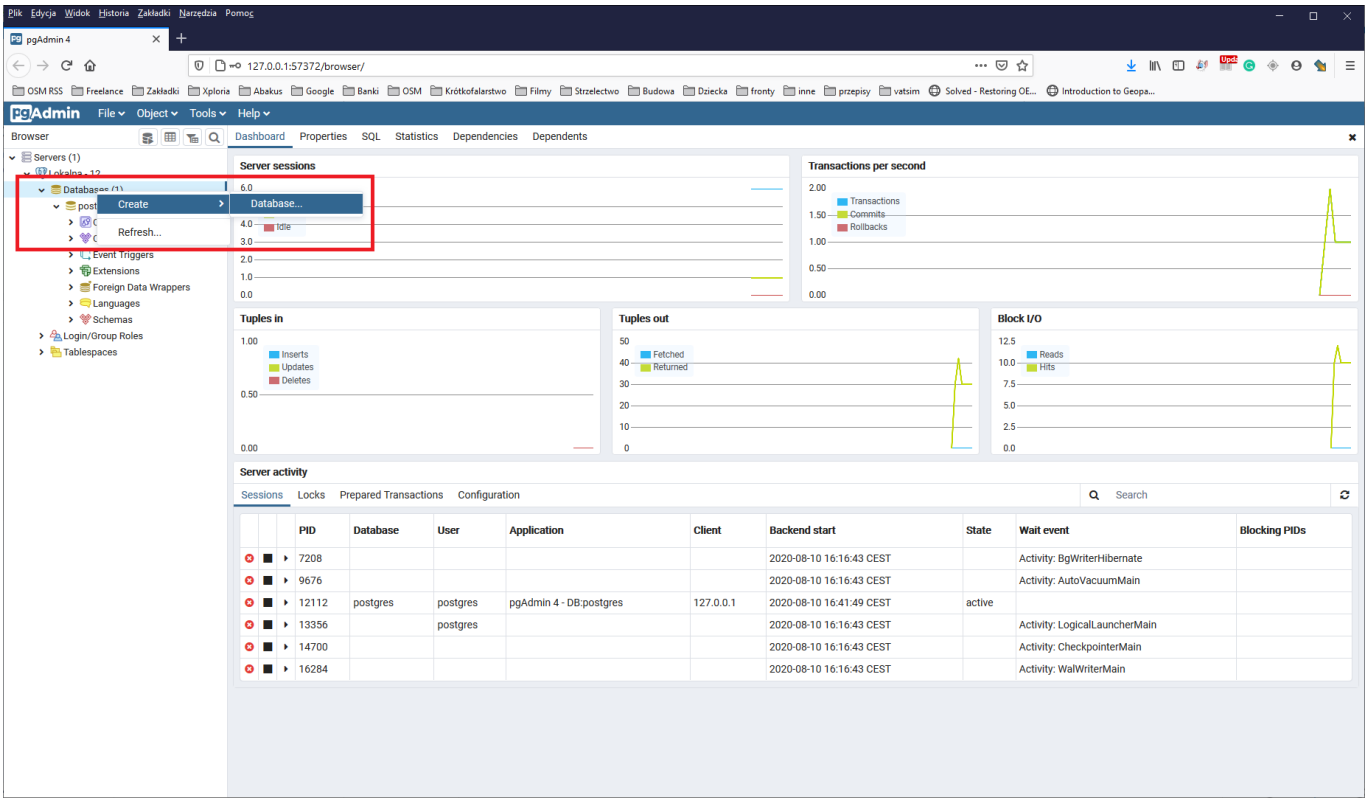
Wyświetlone zostanie okno konfiguracji połączenia z serwerem. W zakładce `General` podajemy nazwę serwera - na potrzeby szkolenia może to być `Lokalna - 12`



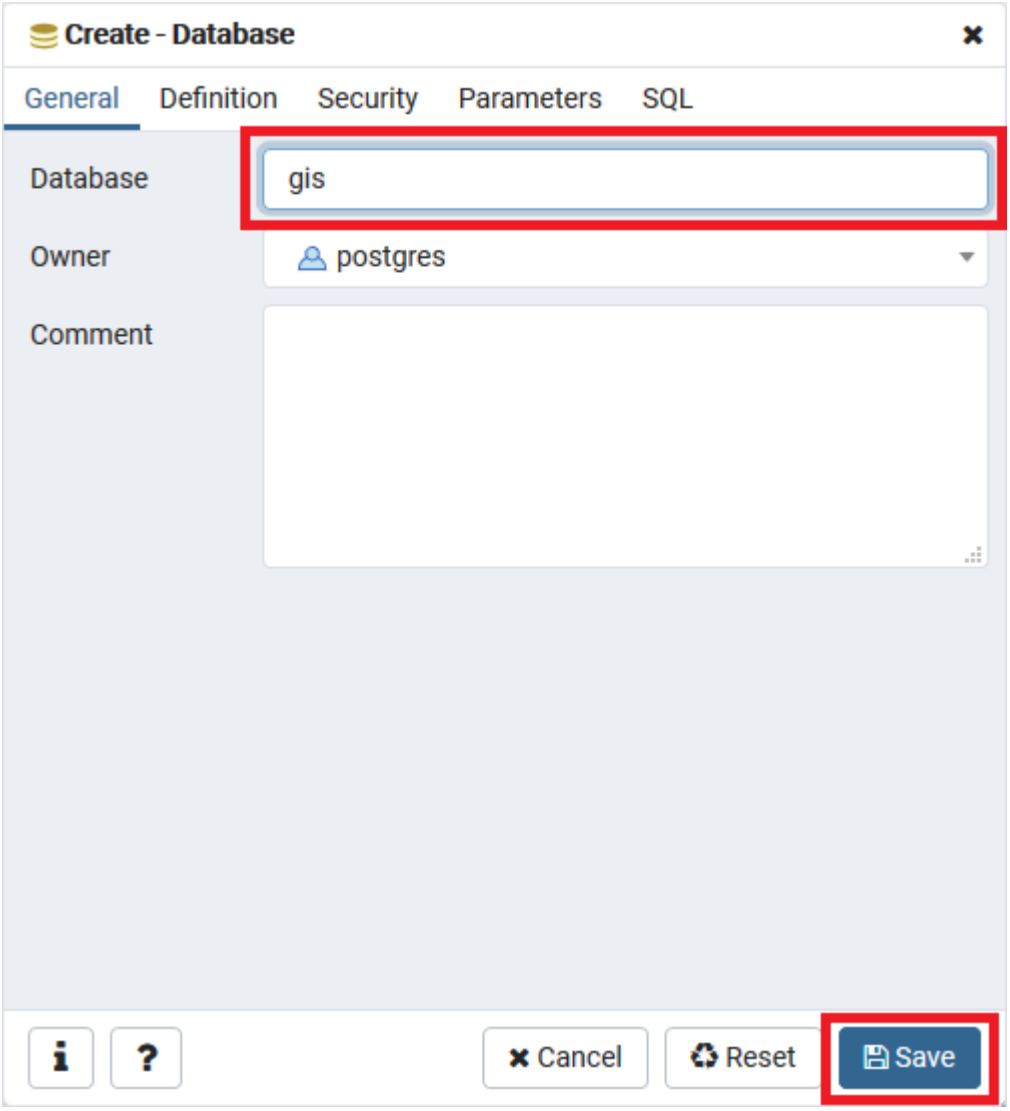
Przechodzimy do zakładki `Connection` gdzie podajemy adres serwera `127.0.0.1` oraz hasło użytkownika `postgres` które zostało utworzone podczas instalacji - zgodnie z zaleceniami powinno to być hasło `gis`.



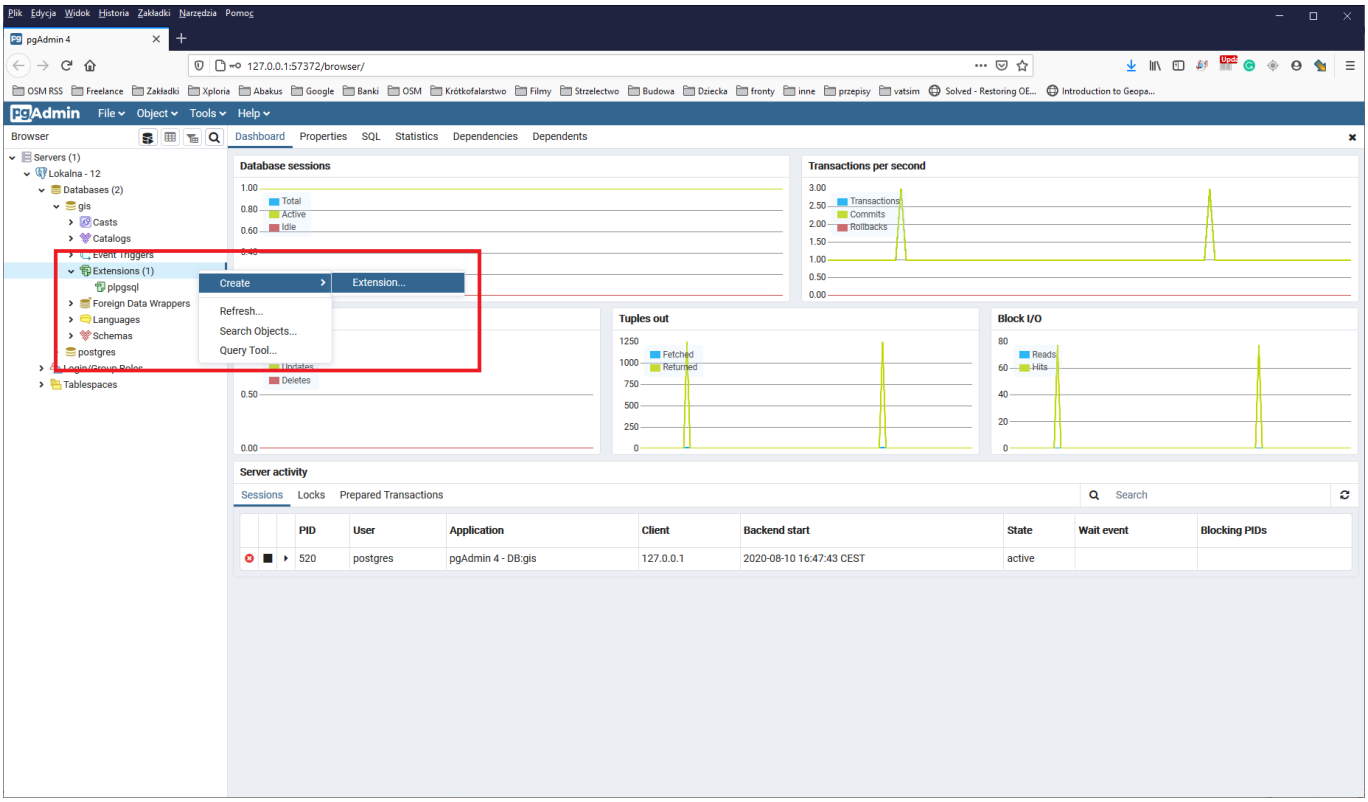
Jeśli podane dane były prawidłowe aplikacja powinna połączyć się do serwera bazy i wyświetlić w drzewie dostępne na tym serwerze bazy danych. Po instalacji dostępna jest tylko jedna baza o nazwie `postgres` i jest to baza serwisowa na której nie powinno się pracować, dlatego teraz utworzymy bazę na której będziemy prowadzili dalsze szkolenie. W tym celu klikamy prawym przyciskiem myszy na pozycji `databases` i z menu kontekstowego wybieramy `create > database`



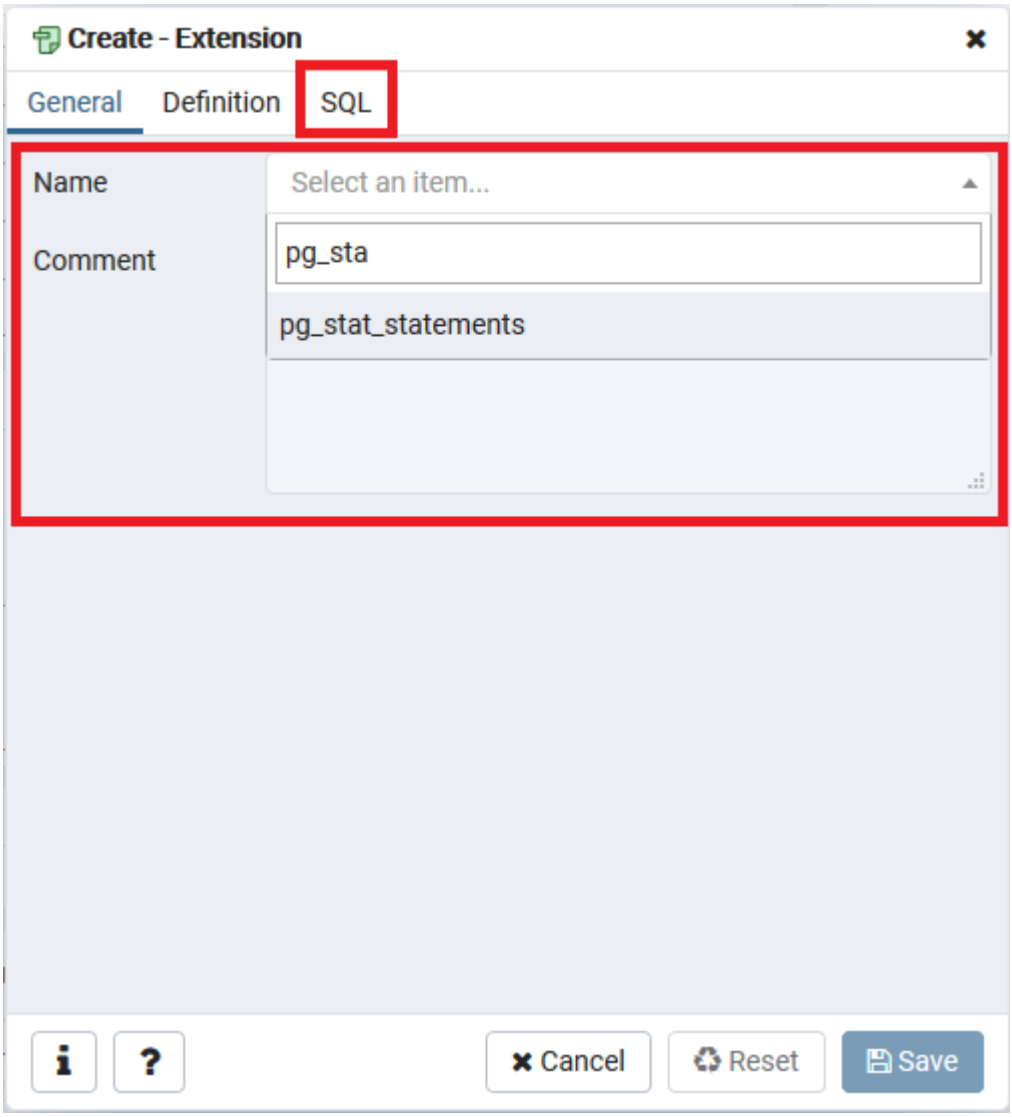
W oknie tworzenia bazy danych w zakładce `General` podajemy jej nazwę - na potrzebę szkolenia sugerujemy nazwę `gis`, i wybieramy `save`



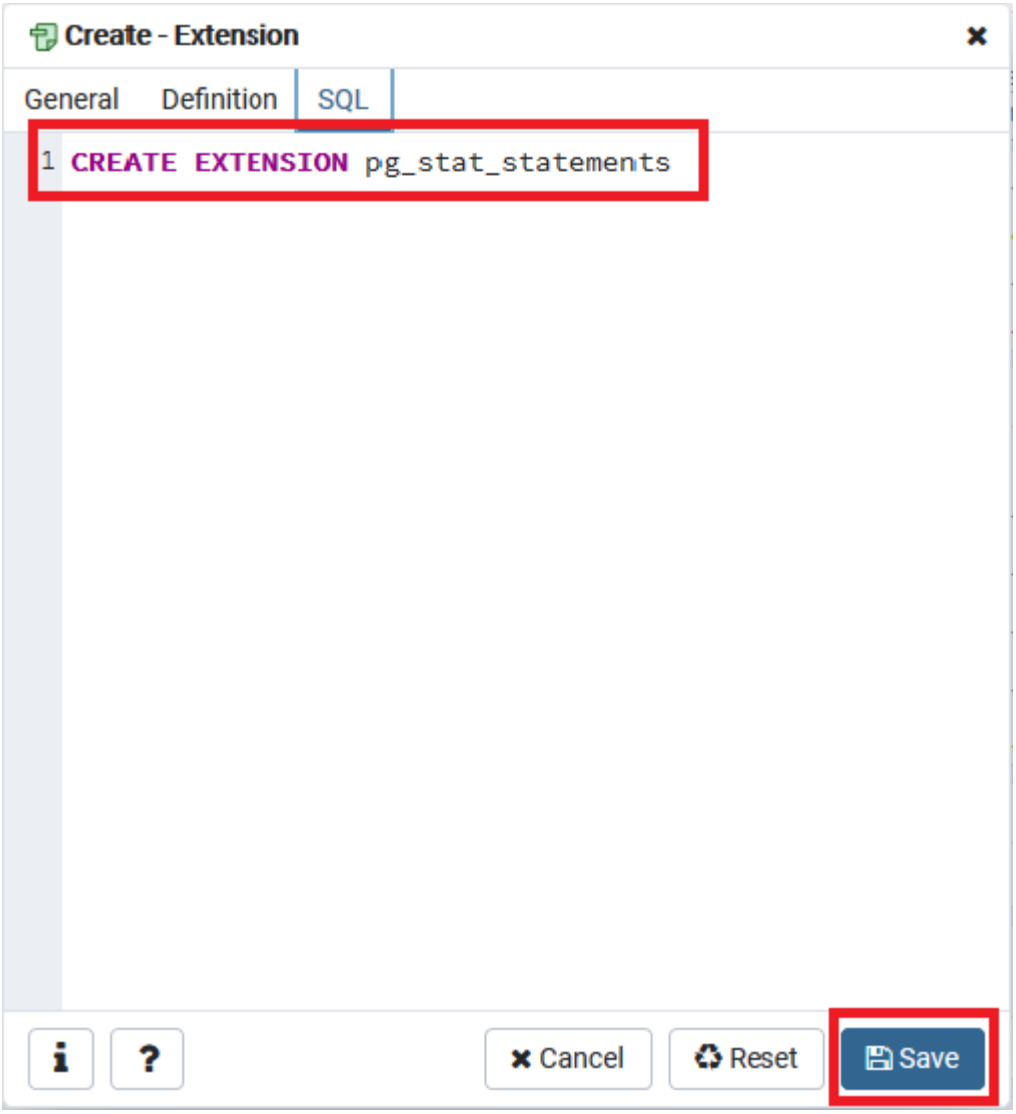
W serwerze powinna zostać utworzona kolejna baza danych - dwuklik w nazwę powinien połączyć aplikację i wyświetlić jej komponenty. Zgodnie z informacjami zawartymi w poprzednim rozdziale do pracy będziemy potrzebowali jeszcze kilku rozszerzeń dlatego odnajdujemy na liście `extensions`, klikamy prawym przyciskiem myszy i wybieramy opcję `create > extension`.



W zakładce `General`, polu `name` wyszukujemy `pg_stat_statements` i wybieramy z listy.



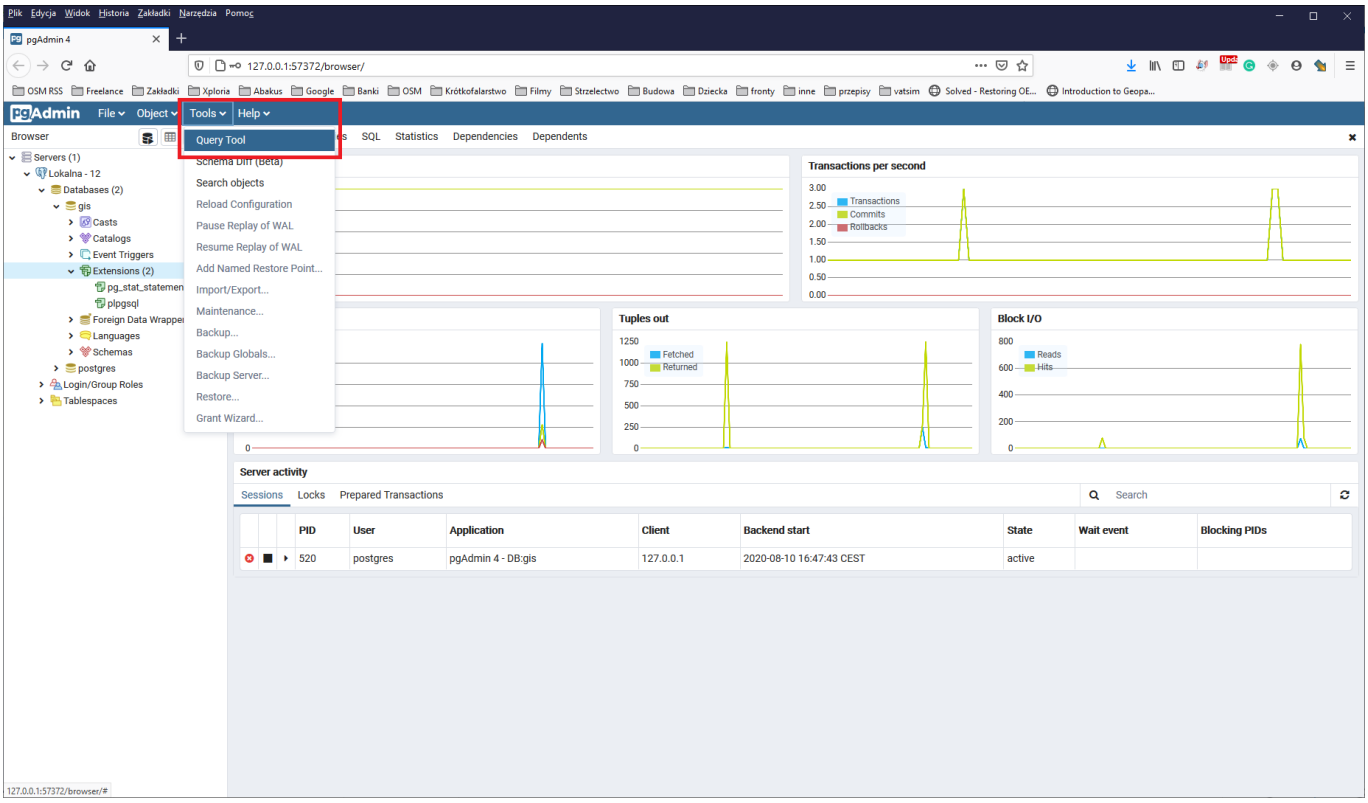
Zasada działania aplikacji pgAdmin jest taka, że pozwala ona za pomocą interfejsu graficznego wygenerować zapytania SQL, które później wysyłane są do bazy. Każde wygenerowane zapytanie możemy podejrzeć na zakładce SQL.



W naszym przypadku jest to zapytanie

```
CREATE EXTENSION pg_stat_statements;
```

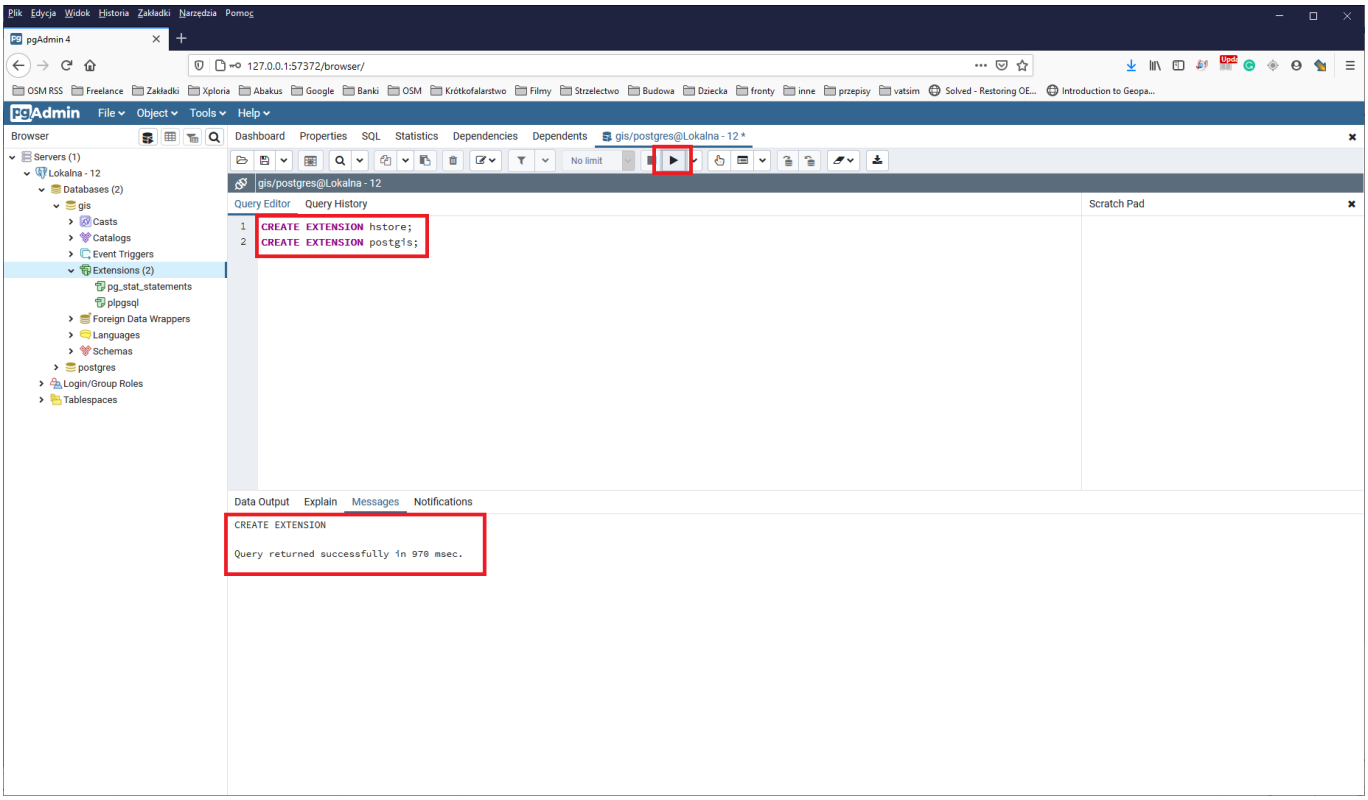
Aby potwierdzić tę zasadę kolejne rozszerzenia zainstalujemy już za pomocą poleceń SQL. W tym celu z menu programu wybieramy `tools > query tool`



W oknie zapytań wpisujemy komendy instalujące kolejne rozszerzenia

```
CREATE EXTENSION hstore;
CREATE EXTENSION postgis;
```

po czym wybieramy ikonę `play` z górnego paska narzędzi. Jeśli wszytko przebiegło prawidłowo w dolnym oknie `messages` powinna pojawić się informacja `CREATE EXTENSION`



Jeśli udało nam się wykonać wszystkie powyższe kroki oznacza to, że:

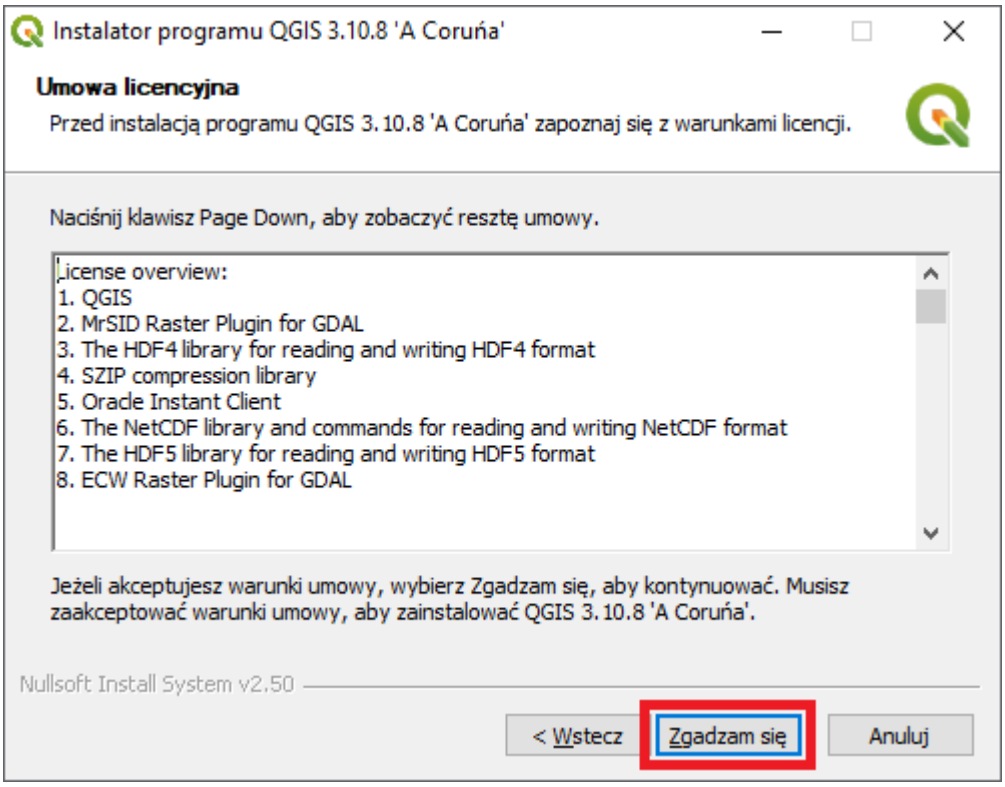
- serwer bazy danych i rozszerzenia zostały poprawnie zainstalowane
- baza danych została utworzona
- wszystkie konieczne rozszerzenia zostały dodane do bazy
- mamy prawidłowo skonfigurowaną aplikację pgAdmin

Instalacja QGIS

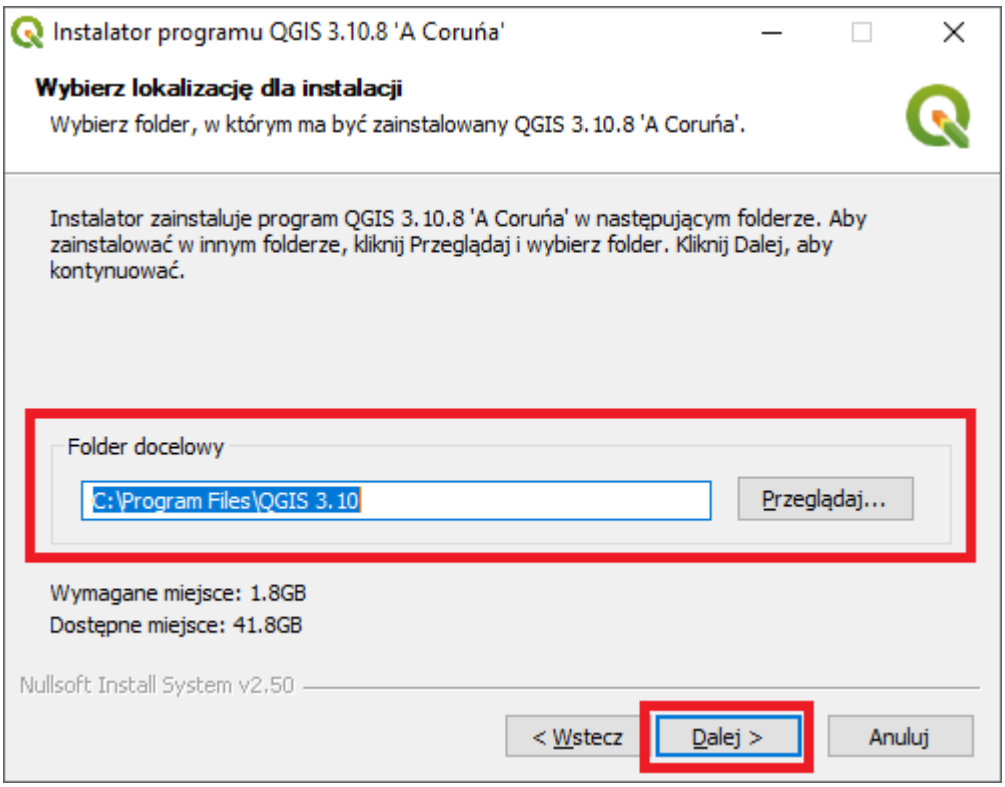
Instalator QGIS dostępny jest na stronie głównej projektu oraz w materiałach szkoleniowych (QGIS-OSGeo4W-3.10.8-1-Setup-x86_64.exe). Twórcy oprogramowania zalecają używanie wersji LTR, dlatego podczas szkolenia będziemy używali wersji 3.10, a nie najnowszej.

Po uruchomieniu instalatora w pierwszym kroku wybieramy `dalej`.

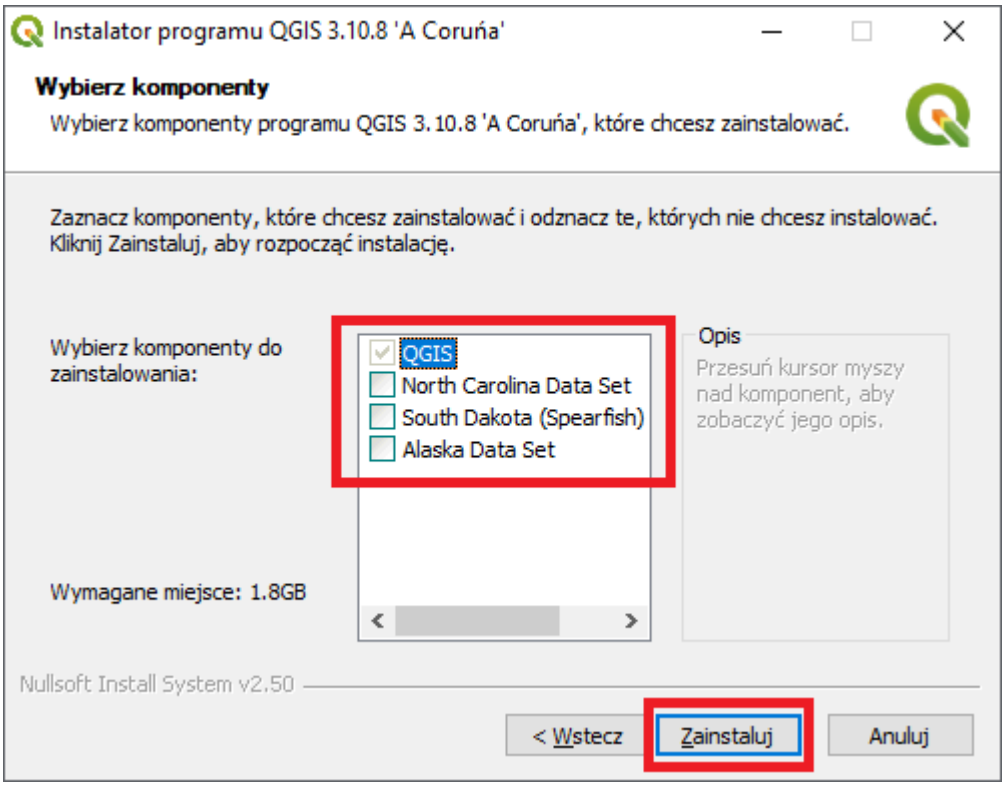
W kolejnym kroku potwierdzamy zapoznanie się z licencją.



Trzeci krok pozwala na zmianę folderu instalacji - zalecamy pozostawienie wartości domyślnej.



W czwartym kroku mamy możliwość wybrania dodatkowych komponentów - w tym przypadku testowych danych. Na potrzeby szkolenia nie będą nam one potrzebne.



Po kilku minutach instalator kończy instalację aplikacji i QGIS jest gotowy do użycia.

Blok 3 - Wprowadzenie do GIS i PostGIS

Obsługiwane reprezentacje danych

Open Geospatial Consortium - międzynarodowa organizacja non-profit zrzeszająca ponad 450 firm, organizacji i uczelni i odpowiedzialna za rozwijanie i implementację otwartych standardów dla danych i

usług przestrzennych - w specyfikacji odnośnie obiektów prostych (dostępnej na stronie organizacji oraz w materiałach szkoleniowych) definiuje dwie standardowe reprezentacje obiektów przestrzennych:

- WKT - Well Known Text - Reprezentacja tekstowa, która może być używana zarówno do tworzenia nowych wystąpień typu, jak i do konwersji istniejących wystąpień do postaci tekstowej w celu wyświetlenia alfanumerycznego.
- WKB - Well Known Binary - binarna reprezentacja geometrii zapewniająca przenośną reprezentację obiektu geometrycznego jako ciągły strumień bajtów.

Każda z tych reprezentacji definiuje zarówno typ obiektu jak i zestaw współrzędnych koniecznych do jego utworzenia. Przykładowe obiekty przestrzenne w reprezentacji WKT:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOINT((0 0),(1 2))
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))

Prawidłowe zapytanie SQL, które tworzy obiekt przestrzenny i wstawia go do bazy może wyglądać następująco:

```
INSERT INTO geotable ( the_geom, the_name )
  VALUES ( ST_GeomFromText('POINT(21.32 45.32)', 4326), 'Jakiś punkt');
```

Reprezentacje zdefiniowane przez OGC są zawsze dwuwymiarowe i nie definiują układów odniesienia. PostGIS rozszerza te reprezentacje o kolejne wymiary oraz układy odniesienia dodając dwie nowe:

- EWKT - Extended WKT
- EWKB - Extended WKB

Przykładowe obiekty w reprezentacji EWKT:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- XY with SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM with SRID
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
- MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
- POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))

- TRIANGLE ((0 0, 0 9, 9 0, 0 0))
- TIN(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))

Prawidłowe zapytanie SQL, które tworzy obiekt przestrzenny i wstawia go do bazy może wyglądać następująco:

```
INSERT INTO geotable ( the_geom, the_name )
  VALUES ( ST_GeomFromEWKT('SRID=4326;POINT(21.32 45.32)'), 'Jakiś punkt'
);
```

Typy danych

Postgis definiuje dwa typy danych przestrzennych:

- geometry - używany do przechowywania danych w płaskich (euklidesowych) układach współrzędnych
- geography - używany do przechowywania danych w kątowych układach współrzędnych

Typy danych możemy definiować ogólnie (np: `geometry`) i dokładnie (np: `geometry(point,4326)`). W pierwszym przypadku baza danych dopuści zapisanie w polu każdej geometrii, którą będziemy chcieli wstawić, w drugim przypadku zezwoli tylko na punkty w odwzorowaniu EPSG:4326

Kiedy użyć geometry a kiedy geography

Typ danych GEOGRAPHY pozwala przechowywać dane używając długości i szerokości geograficznej - bez wchodzenia w zawiłości odwzorowań i układów współrzędnych. Uproszczenie to niesie ze sobą również pewne wady:

- nie wszystkie funkcje obsługują format GEOGRAPHY
- te, które obsługują działają wolniej niż na danych typu GEOMETRY

W takim razie kiedy użyć którego typu?

Typu GEOMETRY lepiej użyć jeśli:

- przetwarzamy dane przestrzenne z niewielkiego obszaru (średniej wielkości państwo)
- wiemy co to odwzorowania i wiemy jak ich używać
- zależy nam na wydajności funkcji przestrzennych

Typu GEOGRAPHY lepiej użyć jeśli:

- przetwarzamy dane na większym obszarze - kontynent lub cała planeta
- nie wiemy co to odwzorowania i nie mamy zamiaru się ich uczyć
- wydajność funkcji przestrzennych nie jest dla nas najistotniejsza

Omówienie grup funkcji wprowadzanych przez postgis

Postgis rozszerza bazę PostgreSQL o około 300 funkcji, które funkcjonalnie możemy podzielić na kilka grup:

Funkcje zarządzania tabelami

Funkcje umożliwiające dodawanie, usuwanie tabel oraz kolumn w tabelach, zwracanie i zmianę zdefiniowanego układu współrzędnych

Przykładowe funkcje:

AddGeometryColumn

Dodaje nową kolumnę z geometrią do wskazanej tabeli

```
SELECT AddGeometryColumn
('my_schema','my_spatial_table','geom',4326,'POINT',2);
```

DropGeometryColumn

Usuwa kolumnę z geometrią z tabeli

```
SELECT DropGeometryColumn ('my_schema','my_spatial_table','geom');
```

Find_SRID

Zwraca numer kodu EPSG układu współrzędnych zdefiniowanego dla wskazanej kolumny w tabeli.

```
SELECT Find_SRID('public', 'tiger_us_state_2007', 'the_geom_4269');
```

UpdateGeometrySRID

Aktualizuje SRID wszystkich elementów w kolumnie geometrii, aktualizując wiązania i odniesienia w geometry_columns. Jeśli kolumna została wymuszona przez definicję typu, definicja typu zostanie

Przykładowe funkcje:

ST_MakePoint

Tworzy geometrię punktową.

```
SELECT ST_MakePoint(-71.1043443253471, 42.3150676015829);
```

ST_MakeEnvelope

Tworzy prostokąt z minimalnych i maksymalnych wartości X i Y. Wartości wejściowe muszą znajdować się w przestrzennym układzie odniesienia określonym przez SRID. Jeśli nie określono SRID, używany jest nieznany system odniesień przestrzennych (SRID 0).

```
SELECT ST_AsText( ST_MakeEnvelope(10, 10, 11, 11, 4326) );
```

ST_Collect

Zbiera geometrie do kolekcji geometrii. Wynikiem jest `Multi*` lub `GeometryCollection`, w zależności od tego, czy geometrie wejściowe mają te same lub różne typy (jednorodne lub niejednorodne). Geometrie wejściowe pozostają niezmienione w kolekcji.

```
SELECT ST_AsText(
  ST_Collect(
    ST_GeomFromText('POINT(1 2)'),
    ST_GeomFromText('POINT(-2 3)')
    )
  );
```

Akcesory

Zwracają informacje o geometriach

Przykładowe funkcje:

ST_X, ST_Y, ST_Z

Zwracają wartość wskazanej współrzędnej dla punktu

```
SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
```

ST_IsClosed

Zwraca wartość TRUE, jeśli punkty początkowe i końcowe LINESTRING pokrywają się. W przypadku powierzchni wielościennych podaje, czy powierzchnia jest płaska (otwarta), czy wolumetryczna (zamknięta).

```
SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)'::geometry);
```

ST_DumpPoints

Ta funkcja zwracająca zbiór (SRF) zwraca zestaw wierszy `geometry_dump` utworzonych przez geometrię (`geom`) i tablicę liczb całkowitych (`path`).

```
SELECT edge_id, (dp).path[1] As index, ST_AsText((dp).geom) As wktnode
FROM (
  SELECT 1 As edge_id, ST_DumpPoints(ST_GeomFromText('LINESTRING(1 2, 3 4,
```

```
10 10)')) AS dp
  UNION ALL
  SELECT 2 As edge_id, ST_DumpPoints(ST_GeomFromText('LINESTRING(3 5, 5 6,
9 10)')) AS dp
) As foo;

 edge_id | index |    wktnode
---------+-------+--------------
       1 |     1 | POINT(1 2)
       1 |     2 | POINT(3 4)
       1 |     3 | POINT(10 10)
       2 |     1 | POINT(3 5)
       2 |     2 | POINT(5 6)
       2 |     3 | POINT(9 10)
```

ST_Envelope

Zwraca minimalną obwiednię dla dostarczonej geometrii jako geometrię. Wielokąt jest zdefiniowany przez punkty narożne ramki ograniczającej ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS doda również współrzędne ZMIN / ZMAX).

```
SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)'::geometry));
```

Edytory

Funkcje pozwalające na edycje geometrii.

Przykładowe funkcje:

ST_AddPoint

Dodaje punkt do linii

```
-- zapytanie zamknie wszystkie geometrie w tabeli
-- dodając do każdej linii na końcu jej punkt startowy
-- tylko dla geometrii niezamkniętych

UPDATE sometable
SET the_geom = ST_AddPoint(the_geom, ST_StartPoint(the_geom))
FROM sometable
WHERE ST_IsClosed(the_geom) = false;
```

ST_SnapToGrid

Przyciągaj wszystkie punkty geometrii wejściowej do siatki zdefiniowanej przez jej początek i rozmiar komórki. Usuń kolejne punkty przypadające na tę samą komórkę, ostatecznie zwracając NULL, jeśli punkty

wyjściowe nie są wystarczające do zdefiniowania geometrii danego typu. Zwinięte geometrie w kolekcji są z niej usuwane. Przydatne do zmniejszania precyzji.

```
--Zmniejsza precyzję geometrii do 3 miejsc po przecinku

UPDATE mytable
   SET the_geom = ST_SnapToGrid(the_geom, 0.001);
```

ST_Reverse

Może być używany na dowolnej geometrii i odwraca kolejność wierzchołków.

```
SELECT
  ST_AsText(the_geom) as line,
  ST_AsText(ST_Reverse(the_geom)) As reverseline
FROM
  (SELECT ST_MakeLine(ST_MakePoint(1,2),ST_MakePoint(1,10)) As the_geom) as
foo;
```

ST_Segmentize

Zwraca zmodyfikowaną geometrię, która nie ma segmentu dłuższego niż podana `max_segment_length`. Obliczanie odległości jest wykonywane tylko w 2d. W przypadku typu `geometry` jednostki długości są jednostkami układu współrzędnych. W przypadku typu `geography` jednostki są w metrach.

```
SELECT
  ST_AsText(
    ST_Segmentize(
      ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),
(-45 -33,-46 -32))')
    ,5)
);
```

Walidatory

Sprawdzanie czy geometria jest poprawna, zwracanie błędów

ST_IsValid

Sprawdza, czy wartość ST_Geometry jest poprawnie sformułowana w 2D zgodnie z regułami OGC. W przypadku geometrii, które są nieprawidłowe `NOTE` PostgreSQL zawiera szczegółowe informacje o tym, dlaczego jest nieprawidłowa. W przypadku geometrii z 3 i 4 wymiarami poprawność nadal jest sprawdzana tylko w 2 wymiarach.

```
SELECT
  ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
  ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As
bad_poly;
```

```
NOTICE:  Self-intersection at or near point 0 0
 good_line | bad_poly
-----------+----------
 t         | f
```

ST_IsValidDetail

Zwraca wiersz `valid_detail`, utworzony przez wartość logiczną (`valid`) określającą, czy geometria jest prawidłowa, varchar (`reason`) podający powód, dla którego jest niepoprawna, oraz geometrię (`location`) wskazującą, gdzie jest nieprawidłowa.

Przydatne do zastępowania i ulepszania kombinacji ST_IsValid i ST_IsValidReason w celu wygenerowania szczegółowego raportu nieprawidłowych geometrii.

```
SELECT gid, reason(ST_IsValidDetail(the_geom)),
ST_AsText(location(ST_IsValidDetail(the_geom))) as location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As
the_geom, gid
FROM (SELECT ST_Buffer(ST_MakePoint(x1*10,y1), z1) As buff, x1*10 + y1*100
+ z1*1000 As gid
  FROM generate_series(-4,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,8) z1
  WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
  INNER JOIN (SELECT
ST_Translate(ST_ExteriorRing(ST_Buffer(ST_MakePoint(x1*10,y1), z1)),y1*1,
z1*2) As line
  FROM generate_series(-3,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,10) z1
  WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(the_geom) = false
ORDER BY gid
LIMIT 3;
```

```
 gid  |      reason       |  location
------+-------------------+------------
```

```
 5330 | Self-intersection | POINT(32 5)
 5340 | Self-intersection | POINT(42 5)
 5350 | Self-intersection | POINT(52 5)
```

ST_IsValidReason

Zwraca tekst określający, czy geometria jest prawidłowa, czy nie, a jeśli nie, powód.

```
SELECT gid, ST_IsValidReason(the_geom) as validity_info
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As
the_geom, gid
FROM (SELECT ST_Buffer(ST_MakePoint(x1*10,y1), z1) As buff, x1*10 + y1*100
+ z1*1000 As gid
  FROM generate_series(-4,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,8) z1
  WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
  INNER JOIN (SELECT
ST_Translate(ST_ExteriorRing(ST_Buffer(ST_MakePoint(x1*10,y1), z1)),y1*1,
z1*2) As line
  FROM generate_series(-3,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,10) z1
  WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(the_geom) = false
ORDER BY gid
LIMIT 3;
```

```
 gid  |      validity_info
------+-------------------------
 5330 | Self-intersection [32 5]
 5340 | Self-intersection [42 5]
 5350 | Self-intersection [52 5]
```

Funkcje układów współrzędnych

Umożliwiają ustawianie, sprawdzanie układu współrzędnych oraz reprojekcje.

ST_SetSRID

Ustawia SRID geometrii na określoną wartość kodu EPSG.

Uwaga - ta funkcja w żaden sposób nie przekształca współrzędnych geometrii - po prostu ustawia metadane definiujące układ współrzędnych, w którym zakłada się, że znajduje się geometria. Jeśli chcesz

przekształcić geometrię w nową projekcję użyj `ST_Transform`.

```
SELECT
  ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
```

ST_SRID

Zwraca kod EPSG dla podanej geometrii.

```
SELECT
  ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
```

```
4326
```

ST_Transform

Zwraca nową geometrię ze współrzędnymi przekształconymi do innego układu współrzędnych. Docelowy układ `to_srid` może być zidentyfikowane przez parametr będący liczbą całkowitą SRID (tj. musi istnieć w tabeli `spatial_ref_sys`). Alternatywnie może być użyta definicja układu współrzędnych PROJ.4, jednak metoda ta nie jest zoptymalizowane. Jeśli docelowy układ współrzędnych jest wyrażony za pomocą łańcucha PROJ.4 zamiast SRID, SRID geometrii wyjściowej zostanie ustawiony na zero. Z wyjątkiem funkcji z from_proj, geometrie wejściowe muszą mieć zdefiniowany SRID.

ST_Transform jest często mylony z ST_SetSRID. ST_Transform faktycznie zmienia współrzędne geometrii z jednego przestrzennego układu odniesienia na inny, podczas gdy ST_SetSRID () po prostu zmienia identyfikator SRID geometrii.

```
SELECT
  ST_AsText(
    ST_Transform(
      ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265
2967450,743265.625 2967416,743238 2967416))',2249)
    ,4326)) As wgs_geom;
```

```
 wgs_geom
---------------------------
 POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231
42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)
```

Funkcje wprowadzania geometrii

Pozwalają na tworzenie geometrii na podstawie znanych reprezentacji.

ST_PointFromText

Konstruuje obiekt punktowy PostGIS ST_Geometry na podstawie Well-Known text. Jeśli nie podano SRID, domyślnie jest ustawiane nieznane (obecnie 0). Jeśli geometria nie jest reprezentacją punktu WKT, zwraca wartość null. Jeśli całkowicie niepoprawny WKT, zgłasza błąd.

Istnieją 2 warianty funkcji ST_PointFromText, pierwszy nie przyjmuje identyfikatora SRID i zwraca geometrię bez zdefiniowanego układu współrzędnych. Drugi przyjmuje identyfikator układu jako drugi argument i zwraca obiekt ST_Geometry, który zawiera ten srid jako część jego metadanych. Srid musi być zdefiniowany w tabeli `spatial_ref_sys`.

Jeśli masz absolutną pewność, że wszystkie geometrie WKT są punktami, nie używaj tej funkcji. Jest wolniejsza niż ST_GeomFromText, ponieważ dodaje dodatkowy krok walidacji. Jeśli budujesz punkty z długich współrzędnych i zależy Ci bardziej na wydajności i dokładności niż na zgodności z OGC, użyj ST_MakePoint lub aliasu ST_Point zgodnego z OGC.

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

ST_PointFromWKB

Funkcja ST_PointFromWKB przyjmuje reprezentację WKB geometrii oraz identyfikator układu współrzędnych (SRID) i tworzy instancję odpowiedniego typu geometrii - w tym przypadku geometrię POINT. Ta funkcja pełni rolę fabryki geometrii w języku SQL.

Jeśli SRID nie jest określony, domyślnie przyjmuje wartość 0. Jeśli ciąg wejściowy nie reprezentuje geometrii POINT zwracane jest NULL.

```
SELECT
  ST_AsText(
    ST_PointFromWKB(
      ST_AsEWKB('POINT(2 5)'::geometry)
  )
  );
```

```
 st_astext
------------
 POINT(2 5)
(1 row)
```

ST_GeomFromGeoJSON

Konstruuje obiekt geometrii PostGIS z reprezentacji GeoJSON.

ST_GeomFromGeoJSON działa tylko dla fragmentów geometrii JSON. Zgłasza błąd, jeśli spróbujesz użyć go na całym dokumencie JSON.

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":
[-48.23456,20.12345]}')) As wkt;
```

```
wkt
------
POINT(-48.23456 20.12345)
```

Funkcje zwracające geometrie

Zwracają geometrie w zdefiniowanych reprezentacjach.

ST_AsEWKT

Zwraca reprezentację WKT dla geometrii poprzedzoną kodem EPSG.

```
SELECT
ST_AsEWKT('0103000020E6100000010000000500000000000000000000000000000000000000
0000000000000000000000000000F03F000000000000F03F000000000000F03F000000000
000F03F000000000000000000000000000000000000000000000000000000000000000'::geometry);
```

```
        st_asewkt
--------------------------------
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)
```

ST_AsEWKB

Zwraca reprezentację EWKB dla geometrii.

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
        st_asewkb
--------------------------------
```

```
\001\003\000\000 \346\020\000\000\001\000
\000\000\005\000\000\000\000
\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\360?\000\000\000\000\000\000\360?
\000\000\000\000\000\000\360?\000\000\000\000\000
\000\360?\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

ST_AsGeoJSON

Zwraca geometrię jako obiekt GeoJSON „geometry” lub wiersz jako obiekt „feature” GeoJSON. Obsługiwane są geometrie 2D i 3D. GeoJSON obsługuje tylko typy geometrii SFS 1.1 (na przykład brak obsługi krzywych).

```
select json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*)::json)
    )
from ( values (1, 'one', 'POINT(1 1)'::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
     ) as t(id, name, geom);
```

```
{
   "type":"FeatureCollection",
   "features":[
      {
         "type":"Feature",
         "geometry":{
            "type":"Point",
            "coordinates":[1,1]
         },
         "properties":{
            "id":1,
            "name":"one"
         }
      },
      {
         "type":"Feature",
         "geometry":{
            "type":"Point",
            "coordinates":[2,2]
         },
         "properties":{
            "id":2,
            "name":"two"
```

```
            }
         },
         {
            "type":"Feature",
            "geometry":{
               "type":"Point",
               "coordinates":[3,3]
            },
            "properties":{
               "id":3,
               "name":"three"
            }
         }
      ]
   }
```

Operatory

Słowa kluczowe pozwalające na badanie zależności między geometriami.

&&

Zwraca TRUE, jeśli obwiednia 2D geometrii A przecina obwiednię 2D geometrii B.

```
SELECT
  tbl1.column1,
  tbl2.column1,
  tbl1.column2 && tbl2.column2 AS overlaps
FROM
  (
    VALUES
      (1, 'LINESTRING(0 0, 3 3)'::geometry),
      (2, 'LINESTRING(0 1, 0 5)'::geometry)
  ) AS tbl1,
  (
    VALUES
      (3, 'LINESTRING(1 2, 4 6)'::geometry)
  ) AS tbl2;
```

```
 column1 | column1 | overlaps
---------+---------+----------
       1 |       3 | t
       2 |       3 | f
(2 rows)
```

<->

Zwraca odległość 2D między dwiema geometriami. Użyty w klauzuli `ORDER BY` zapewnia indeksowane zestawy wyników najbliższego sąsiada. Dla PostgreSQL poniżej 9,5 podaje tylko odległość środka ciężkości obwiedni, a dla PostgreSQL 9.5+, wykonuje prawdziwe wyszukiwanie odległości KNN, podając rzeczywistą odległość między geometriami.

```
SELECT
  st_distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as
d,edabbr, vaabbr
FROM
  va2005
ORDER BY
  geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry limit 10;
```

```
        d         | edabbr | vaabbr
------------------+--------+--------
                0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Funkcje relacji przestrzennych

Funkcje badające przecięcia, nakładanie, stykanie, zawieranie się geometrii.

ST_Contains

Geometria A zawiera geometrię B wtedy i tylko wtedy, gdy żadne punkty B nie leżą na zewnątrz A i co najmniej jeden punkt wnętrza B leży we wnętrzu A.

Zwraca TRUE, jeśli geometria B jest całkowicie wewnątrz geometrii A. Aby ta funkcja miała sens, obie geometrie źródłowe muszą mieć ten sam układ współrzędnych. ST_Contains jest odwrotnością ST_Within. Tak więc ST_Contains (A, B) implikuje ST_Within (B, A) z wyjątkiem przypadków nieprawidłowych geometrii, w których wynik jest zawsze `false`.

```
SELECT ST_Contains(
               ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10),
               ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20)
           )

-- false
```

```
SELECT ST_Contains(
               ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20),
               ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10)
           )

-- true
```

ST_Intersects

Jeśli `geometry` lub `geography` dzieli jakąkolwiek część przestrzeni, to się przecinają. W przypadku geografii - tolerancja wynosi 0,00001 metra (więc wszystkie punkty, które są blisko, są traktowane jako przecinające się)

Funkcje `ST_Overlaps`, `ST_Touches`, `ST_Within` implikują przecięcie przestrzenne. Jeśli którakolwiek z wyżej wymienionych zwraca `true`, wówczas geometrie również przecinają się przestrzennie.

```
SELECT
  ST_Intersects(
    'POINT(0 0)'::geometry,
    'LINESTRING ( 0 0, 0 2 )'::geometry
  );

-- true
```

Funkcje pomiarowe

Funkcje pozwalające na pomiary geometrii i zależności między nimi.

ST_Area

Zwraca obszar geometrii wielokątnej. W przypadku typu `geometry` obliczany jest obszar kartezjański 2D (płaski) w jednostkach określonych przez SRID. Dla typu `geography` domyślnie obszar jest określany na sferoidzie w metrach kwadratowych. Aby obliczyć obszar przy użyciu szybszego, ale mniej dokładnego modelu sferycznego, użyj `ST_Area (geog, false)`.

```
select ST_Area(geom) sqft,
    ST_Area(geom) * 0.3048 ^ 2 sqm
from (
         select 'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265
2967450,743265.625 2967416,743238 2967416))' :: geometry geom
     ) subquery;
```

```
┌──────────┬──────────────────┐
│   sqft   │       sqm        │
```

```
┌──────────┬──────────────┐
│ 928.625  │ 86.27208552  │
└──────────┴──────────────┘
```

ST_Perimeter

Zwraca obwód 2D geometrii. W przypadku geometrii nieobszarowych zwracana jest wartość 0. Dla typu `geomatry` jednostki miary obwodu są określane przez układ współrzędnych geometrii. Dla typu `geography` jednostkami obwodu są metry.

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238
2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
```

```
st_perimeter
---------
 122.630744000095
(1 row)
```

ST_Distance

Dla typu `geometry` zwraca minimalną odległość kartezjańską (płaską) 2D między dwiema geometriami w jednostkach układu współrzędnych.Dla typu `geography` domyślnie zwracana jest minimalna odległość geodezyjna między dwoma obiektami geograficznymi w metrach, obliczana na sferoidzie określonej przez SRID. Jeśli `use_spheroid` ma wartość `false`, stosowane są szybsze obliczenia sferyczne.

```
SELECT ST_Distance(
  'SRID=4326;POINT(-72.1235 42.3521)'::geometry,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry
  );
```

```
st_distance
-----------------
0.00150567726382282
```

Procesory

Funkcje umożliwiające zmiany geometrii.

ST_Buffer

Zwraca geometrię lub geografię reprezentującą wszystkie punkty, których odległość od tej geometrii / geografii jest mniejsza lub równa podanej odległości.

```
select
  ST_AsText(
    ST_Buffer(
      (ST_SetSRID(ST_MakePoint(0,0),4326))::geography,
      10000
    )
  );
```

```
 POLYGON((0.089747155982037 0,0.0880224840241
-0.017627488756351,0.082914776598006 -0.034577404987677,0.074620406584809
-0.050198238110987,0.063458252536095 -0.063889596633595,0.049857425648444
-0.075125294725721,0.034340756571451 -0.083473578931769,0.017504679745175
-0.088613716712858,-0.000003708297878 -0.090348309601953,-0.01751153178455
-0.088610859377427,-0.034346000891237 -0.083468299265945,-0.049860263848495
-0.075118396513058,-0.063458252528508 -0.063882130066058,-0.074617568374027
-0.0501913399058,-0.082909532278218 -0.034572125332427,-0.088015631995455
-0.017624631428396,-0.089739739401455 0,-0.088015631995455
0.017624631428396,-0.082909532278218 0.034572125332426,-0.074617568374027
0.050191339905799,-0.063458252528508 0.063882130066058,-0.049860263848495
0.075118396513057,-0.034346000891238 0.083468299265945,-0.01751153178455
0.088610859377427,-0.000003708297878 0.090348309601953,0.017504679745174
0.088613716712858,0.03434075657145 0.083473578931769,0.049857425648443
0.075125294725721,0.063458252536095 0.063889596633595,0.074620406584808
0.050198238110987,0.082914776598006 0.034577404987678,0.0880224840241
0.017627488756351,0.089747155982037 0))
(1 row)
```

ST_Centroid

Oblicza geometryczny środek geometrii lub środek ciężkości geometrii jako PUNKT.

```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0
1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6 )'));
```

```
          st_astext
----------------------------------------
 POINT(2.30769230769231 3.30769230769231)
(1 row)
```

ST_Simplify

Zwraca „uproszczoną” wersję podanej geometrii. Uproszczenie odbywa się za pomocą algorytmu Douglasa-Peuckera.

Uwaga - upraszczając za bardzo koło stanie się ośmiokątem lub trójkątem.

```
SELECT ST_Npoints(the_geom) AS np_before,
       ST_NPoints(ST_Simplify(the_geom,0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(the_geom,0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(the_geom,1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(the_geom,10)) AS np10_triangle,
       (ST_Simplify(the_geom,100) is null) AS  np100_geometrygoesaway
  FROM
    (SELECT ST_Buffer('POINT(1 3)', 10,12) As the_geom) AS foo;

 -- np_before: 49
 -- np01_notbadcircle: 33
 -- np05_notquitecircle:17
 -- np1_octagon: 9
 -- np10_triangle: 4
 -- np100_geometrygoesaway: f
```

ST_Union

Łączy geometrie zwracając nową geometrię bez przecinających się regionów.

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' ::
geometry))
```

```
st_astext
----------
MULTIPOINT(-2 3,1 2)
```

Może być również użyta jako funkcja agregująca:

```
SELECT
  stusps,
  ST_Union(f.geom) as singlegeom
FROM sometable f
GROUP BY stusps
```

Przekształcenia afiniczne

Funkcje umożliwiające przekształcenia afiniczne (pokrewne) geometrii.

ST_Rotate

Obraca geometrię o zadaną wartość podaną w radianach przeciwnie do ruchu wskazówek zegara wokół zadanego punktu obrotu.

```
--Obrót o 180 stopni
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
```

```
               st_asewkt
---------------------------------------
 LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)
```

ST_Scale

Skaluje geometrię do nowego rozmiaru, mnożąc rzędne przez zadany współczynnik.

```
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5,
0.75, 0.8));
```

```
        st_asewkt
-------------------------------------
 LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)
```

ST_Translate

Zwraca nową geometrię, której współrzędne są przesunięte o wartości delta x, delta y, delta z. Jednostki są oparte na jednostkach zdefiniowanych w układzie współrzędnych (SRID) dla tej geometrii.

```
SELECT
  ST_AsText(
    ST_Translate(
      ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As
wgs_transgeomtxt;
```

```
wgs_transgeomtxt
---------------------
POINT(-70.01 42.37)
```

Funkcje obwiedni

Funkcje pozwalające na tworzenie, agregację i pracę z obwiedniami.

ST_Extent

Zwraca obwiednię, która obejmuje zestaw geometrii. Funkcja `ST_Extent` jest funkcją agregującą w terminologii SQL. Oznacza to, że działa na listach danych, w ten sam sposób, w jaki działają funkcje `SUM()` i `AVG()`.

Ponieważ zwraca ramkę ograniczającą, jednostki przestrzenne są zgodne z używanym układem współrzędnych danych.

```
SELECT ST_Extent(the_geom) as bextent FROM sometable;
```

```
                    st_bextent
-----------------------------------
BOX(739651.875 2908247.25,794875.8125 2970042.75)
```

ST_Expand

Funkcja zwraca obwiednie uzyskaną przez rozszerzenie obwiedni wejścia. Stopień rozszerzenia może się odbyć przez określenie pojedynczej odległości na jaką prostokąt powinien zostać rozszerzony we wszystkich kierunkach, albo przez określenie odległości rozwinięcia dla każdego kierunku.

```
SELECT
  CAST(
    ST_Expand(
      ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892
110714)', 2163)
      ,10)
    As box2d);
```

```
                    st_expand
-----------------------------------
 BOX(2312882 110666,2312990 110724)
```

ST_XMax, ST_XMin, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

Zestaw funkcji zwracających maksymalne i minimalne wartości współrzędnych obwiedni w poszczególnych osiach wymiarów.

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
st_xmax
-------
5
```

Funkcje referencji liniowej

Funkcje pozwalające na pracę z liniami.

ST_LineLocatePoint

Zwraca liczbę zmiennoprzecinkową z przedziału od 0 do 1, reprezentującą położenie najbliższego punktu na linii do zadanego Punktu, jako ułamek całkowitej długości 2d linii.

Możesz użyć zwróconej lokalizacji, aby wyodrębnić `Point` (`ST_LineInterpolatePoint`) lub fragment linii (`ST_LineSubstring`).

```
select st_LineLocatePoint(
  ST_MakeLine(ST_MakePoint(0,0), ST_MakePoint(0,2)),
  ST_MakePoint(0,1)
);
```

```
 st_linelocatepoint
--------------------
                0.5
(1 row)
```

ST_LineInterpolatePoint

Zwraca punkt interpolowany wzdłuż linii. Pierwszy argument musi być `LINESTRING`. Drugi argument to liczba zmiennoprzecinkowa między 0 a 1 reprezentująca ułamek całkowitej długości linii.

```
select
  ST_AsText(
    ST_LineInterpolatePoint(
      ST_MakeLine(ST_MakePoint(0,0), ST_MakePoint(0,2)),
      0.5
    )
  );
```

```
 st_astext
------------
 POINT(0 1)
(1 row)
```

Wykaz wszystkich funkcji wraz z ich definicjami można znaleźć w dokumentacji PostGIS

Omówienie odwzorowań używanych w Polsce

Odwzorowanie kartograficzne (geograficzne) to określony matematycznie sposób dwuwymiarowego i przeskalowanego przedstawiania powierzchni części lub całości kuli ziemskiej lub innego ciała niebieskiego na płaszczyźnie.

Ze względu na zniekształcenia wynikające z odwzorowania, odwzorowania mogą być:

- wiernoodległościowe,
- wiernopowierzchniowe,
- wiernokątne.

Zależnie od powierzchni, na którą odwzorowuje się siatkę geograficzną, rozróżnia się odwzorowania kartograficzne:

- klasyczne: płaszczyznowe, stożkowe, walcowe
- umowne (pseudoklasyczne) – powstają w wyniku modyfikacji siatek klasycznych: pseudopłaszczyznowe, siatki globuralne, siatki koliste, siatki azymutoidalne, pseudostożkowe, pseudowalcowe, wielościenne i inne.

Zależnie od położenia powierzchni odwzorowania w stosunku do kuli ziemskiej rozróżnia się odwzorowania kartograficzne: normalne, poprzeczne (równikowe), ukośne.

Ze względu na położenie środka rzutu klasyfikuje się siatki: centralne, stereograficzne, ortograficzne.

Ze względu na odległość powierzchni rzutu od kuli, odwzorowania mogą być: styczne, sieczne, odległe.

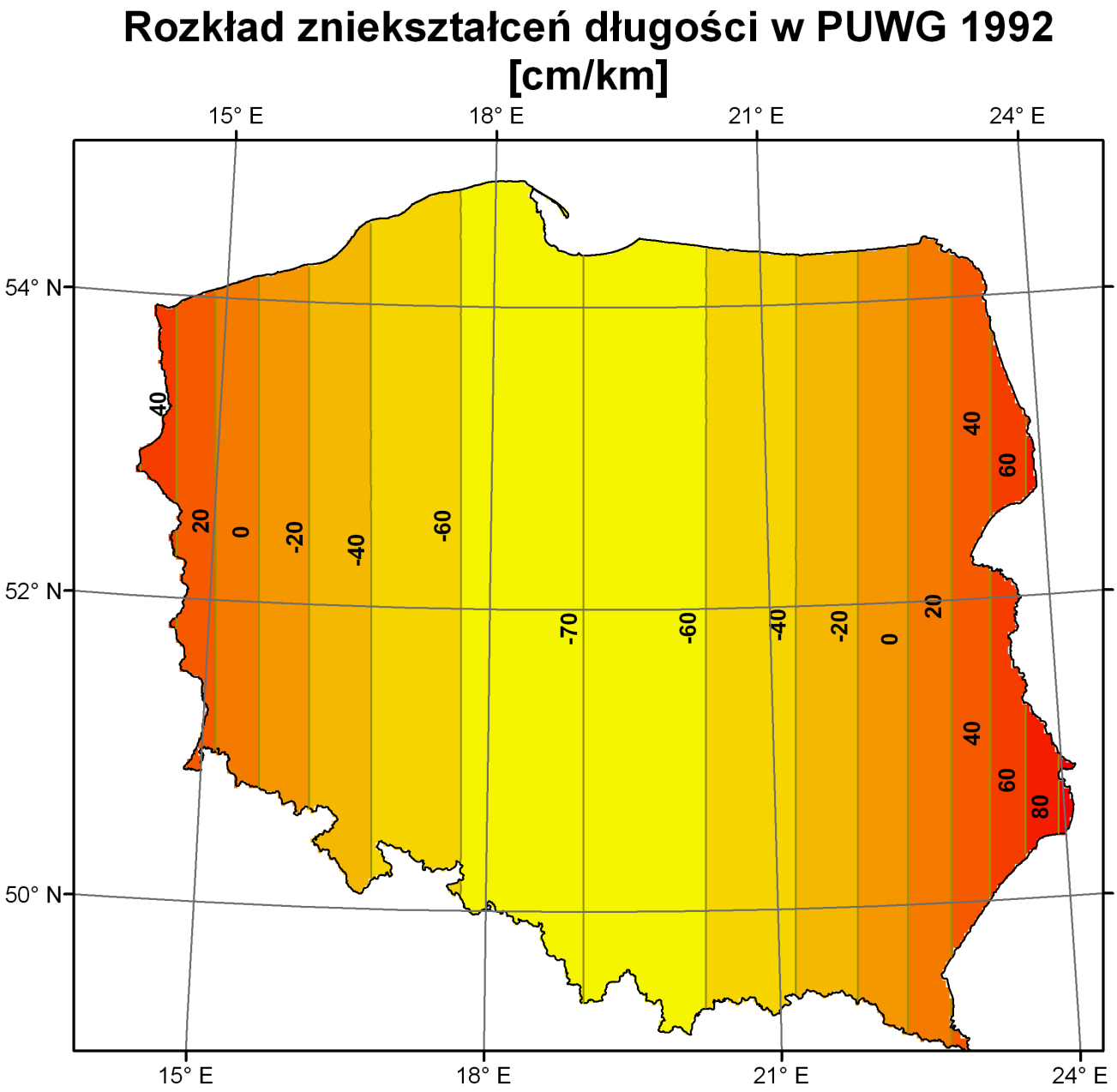
Układ współrzędnych PL-1992

Kod EPSG:2180

Układ współrzędnych 1992 (Państwowy Układ Współrzędnych Geodezyjnych 1992) – układ współrzędnych płaskich prostokątnych oparty na odwzorowaniu Gaussa-Krügera dla elipsoidy GRS80 w jednej dziesięciostopniowej strefie.

Początkiem układu jest punkt przecięcia południka 19°E z obrazem równika. Południk środkowy odwzorowuje się na linię prostą w skali m0 = 0,9993, na południku środkowym zniekształcenie wynosi –70 cm/km i rośnie do +90 cm/km na skrajnych wschodnich obszarach Polski. Układ stanowi podstawę do sporządzania map w skalach 1:10 000 i mniejszych, ze względu na duże zniekształcenia. Układ ten jest wykorzystywany m.in. do sporządzania Leśnych Map Numerycznych w Lasach Państwowych. Do opracowań w większych skalach (1:5000 i większe) stosuje się układ współrzędnych 2000.

Zgodnie z aktualnym stanem prawnym jest to jedyny układ dla opracowań małoskalowych obowiązujący w Polsce.



Układ współrzędnych PL-2000

Kody EPSG:

- strefa 5 - EPSG:2176
- strefa 6 - EPSG:2177
- strefa 7 - EPSG:2178
- strefa 8 - EPSG:2179

Układ współrzędnych 2000 (Państwowy Układ Współrzędnych Geodezyjnych 2000, PL-2000) – układ współrzędnych płaskich prostokątnych zwany układem „2000”, powstały w wyniku zastosowania odwzorowania Gaussa-Krügera dla elipsoidy GRS 80 w czterech trzystopniowych strefach o południkach osiowych 15°E, 18°E, 21°E i 24°E, oznaczone odpowiednio numerami – 5, 6, 7 i 8. Skala długości

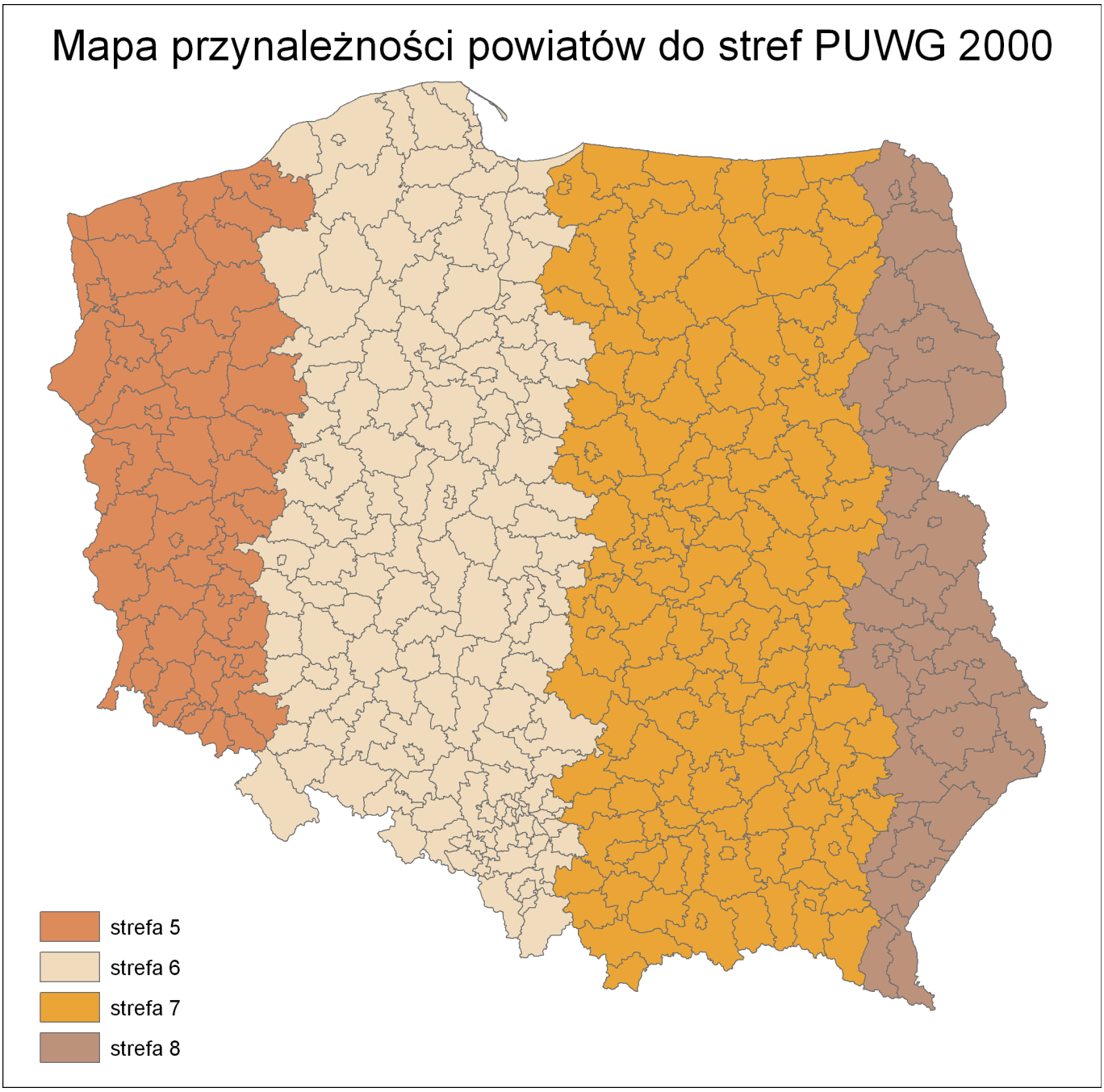
odwzorowania na południkach osiowych wynosi m0 = 0,999923. Zniekształcenia na południku osiowym wynoszą −7,7 cm/km zaś na styku stref +7 cm/km.

Układ 2000 został wprowadzony Rozporządzeniem Rady Ministrów w sprawie państwowego systemu odniesień przestrzennych z 8 sierpnia 2000, od 1 stycznia 2010 jest to jedyny układ współrzędnych geodezyjnych obowiązujący w Polsce. Układ stosuje się na potrzeby wykonywania map w skalach większych od 1:10 000 – w szczególności mapy ewidencyjnej i mapy zasadniczej. Zastąpił on układ współrzędnych „1965".

Od roku 2012 zgodnie z nowym rozporządzeniem o państwowym systemie odniesień przestrzennych układ posiada oznaczenie PL-2000. Zastosowanie, elementy oraz parametry techniczne m.in. układu 2000 reguluje rozporządzenie Rady Ministrów z 15 października 2012 roku.

W układzie tym koncepcja nawiązuje do dawnego układu współrzędnych „1942". Różnica polega jednak na odmienności przyjętych elipsoid odniesienia oraz na zastosowaniu dodatkowej skali podobieństwa (skali kurczenia na południkach środkowych).

Przynależność poszczególnych powiatów do stref przedstawia poniższa mapa



European Terrestial Reference System 1989

Jest to geodezyjny europejski ziemski system odniesienia, przyjęty rezolucją nr 7 na XVII Zgromadzeniu Generalnym Międzynarodowej Unii Geodezji i Geofizyki w Canberzew 1979r., zatwierdzony rezolucją nr 1 na zgromadzeniu podkomisji EUREF(IAG Reference Frame Sub-Commission for Europe) we Florencji w 1990r. jako identyczny z Międzynarodowym Ziemskim Systemem Odniesienia ITRS (International Terrestrial Reference System) na epokę 1989.0.

Jako system jest pozbawiony odwzorowania kartograficznego. Implementowany wprost opatrzony jest kodem EPSG:4258, a oparty o niego układ współrzędnych geodezyjnych o takiej samej nazwie kodem EPSG:6258.

Zgodnie z wytycznymi dyrektywy INSPIRE układ ten jest właściwy do publikacji i udostępniania usług i zasobów danych przestrzennych objętych tą dyrektywą.

W Polsce matematyczną i fizyczną realizacją europejskiego ziemskiego systemu odniesienia jest system o nazwie PL-ETRF89 i jest on systemem referencyjnym dla układu współrzędnych PL-2000.

Web Mercator

Kod EPSG:3857

Odwzorowanie walcowe równokątne (odwzorowanie Merkatora) to odwzorowanie walcowe Ziemi. Południkom i równoleżnikom odpowiadają odcinki, kąty między nimi są zachowane.

Odwzorowanie na równiku jest dokładne, ale wraz z oddalaniem się od niego błędy rosną, gdyż na odwzorowaniu wszystkie równoleżniki mają te same długości. Prowadzi to do ogromnych deformacji wyglądu obszarów w okolicach bieguna. Z tych powodów jej używanie ma sens tylko w nawigacji, gdyż bardzo łatwo znaleźć na takiej mapie dowolny punkt o zadanych współrzędnych geograficznych oraz wyznaczać azymuty. Do celów geodezyjnych, kartograficznych, katastralnych i gospodarczych, powszechnie wykorzystywane jest zmodyfikowane odwzorowanie Merkatora – wiernokątne walcowe poprzeczne np. w układzie UTM.

Jest to jedno z najstarszych odwzorowań kartograficznych, wynalezione w XVI w. przez flamandzkiego kartografa Gerarda Merkatora, w czasach wielkich odkryć geograficznych, związanych z długimi wyprawami morskimi, kiedy szczególnie ważne było wyznaczanie azymutów, a zatem też wiernokątność mapy.

Jest to odwzorowanie najczęściej używane w mapach cyfrowych dostępnych w sieci Internet:

WGS-84

Kod EPSG:4326

Jest to układ współrzędnych, który został zaprojektowany jako jednolity dla całego świata. Jest powszechnie wykorzystywany w urządzeniach do nawigacji oraz przez NATO do sporządzania map wojskowych. Jego elementem charakterystycznym jest zapis w stopniach długości i szerokości geograficznej północnej i południowej. Sposobów zapisów współrzędnych jest sporo. Inną wizytówka tego układu jest charakterystyczne równoleżnikowe rozciągnięcie danych.

Tradycyjny sposób zapisu oparty na stopniach, minutach i sekundach np 15° 16′ 32” E oraz 53° 18′ 22” N jest wypierany przez nowocześniejszy (łatwiejszy do obliczeń maszynowych) np. 18,34522 oraz 40,7654674. Oznaczenie N,S,E,W zostały zastąpione poprzez znaki „minus”. (półkula zachodnia i południowa otrzymują

minus przez wartością. i tak punkt w Ameryce Południowej może mieć współrzędne -100,0000 oraz -40,0000 zamiast 100° 00’00” W oraz 40° 00’00” S

Blok 4 - Importy danych do bazy

Różnice między formatami plikowymi a bazą danych

Przechowywanie danych przestrzennych w bazach danych daje nam wiele nowych możliwości, takich jak:

- tworzenie modeli relacyjnych unikających nadmiarowości
- dostęp do nowych typów danych
- praca równoległa na jednym zbiorze danych
- wykonywanie analiz przestrzennych za pomocą języka SQL
- automatyzacja procesów z użyciem funkcji i wyzwalaczy
- dużo wyższa wydajność dzięki indeksowaniu danych

Różnice między bazą danych a formatami plikowymi są na tyle istotne, że do wydajnej pracy na danych przestrzennych nie wystarczy jedynie ich import - konieczne jest dodatkowe ich przetworzenie i normalizacja.

Tak samo nie da się wyeksportować danych z bazy do formatu plikowego bez utraty części funkcjonalności choćby z powodu ograniczonej ilości typów danych czy ograniczenia długości nazwy kolumny w plikach ESRI Shapefile. Baza danych ma również możliwość przechowywania różnych typów geometrii w jednej tabeli, na co większość formatów plikowych nie pozwala. Mając powyższe na uwadze każdorazowo przed eksportem danych z bazy konieczne jest odpowiednie przygotowanie zasobu.

Omówienie użytego oprogramowania

W bieżącym bloku użyjemy aplikacji `ogr2ogr` oraz `ogrinfo`, które są elementami pakietu GDAL.

GDAL to biblioteka translacji dla formatów danych przestrzennych, zarówno rastrowych i wektorowych, wydana na licencji Open Source przez Open Source Geospatial Foundation. Jako biblioteka przedstawia pojedynczy abstrakcyjny model danych rastrowych i pojedynczy abstrakcyjny model danych wektorowych w aplikacji wywołującej dla wszystkich obsługiwanych formatów. Zawiera również szereg przydatnych narzędzi wiersza poleceń do translacji i przetwarzania danych.

GDAL/OGR jest używane przez niektóre systemy GIS, m.in.: GRASS GIS, OpenEV, FME, Google Earth, i ESRI (od ArcGIS 9.2).

Pakiet GDAL jest instalowany zarówno z serwerem bazy danych jak i z aplikacją QGIS - w szkoleniu użyjemy instalacji QGIS, ponieważ instalacja PostGIS nie posiada wszystkich potrzebnych sterowników.

Użyjemy również aplikacji `shp2pgsql` oraz `pgsql2shp` - programów terminalowych pozwalających na konwersję plików .shp do skryptu SQL zgodnego z PostGIS, oraz do zapisu w plikach ESRI Shapefile danych bezpośrednio z bazy danych PostgreSQL. Programy zostały zainstalowane wraz z serwerem bazy danych i umieszczone w folderze instalacji (domyślnie: `"C:\Program Files\PostgreSQL\12\bin\"`).

Import wykonamy za pomocą `psql` - terminalowego klienta bazy danych PostgreSQL. Jest to interakcyjne narzędzie do wykonywania zapytań (Interactive Query Tools – ISQL). Narzędzie to jest podobne do SQLPlus w bazie danych Oracle, czy ISQL w bazie danych Sybase. Program ten może być także wykorzystywany do zapisywania wyników zapytań do plików lub ewentualnego wykorzystania przez skrypty powłoki.

Uzyskanie informacji o pliku shp do importu

Do importu użyjemy pliku .shp z nazwami miejscowości (`gis_osm_places_free_1.shp`) pochodzącej z danych OpenStreetMap pobranych z serwera Geofabrik, dostępnych w folderze z danymi szkolenia.

Zawartość pliku możemy sprawdzić za pomocą dowolnego narzędzia GIS, lub aplikacji `ogrinfo.exe`, która jak i cały pakiet `GDAL` została zainstalowana razem z rozszerzeniem PostGIS. Skróconą instrukcją użycia otrzymamy uruchamiając program bez żadnych parametrów:

```
"C:\Program Files\QGIS 3.10\bin\ogrinfo.exe"
```

```
Usage: ogrinfo [--help-general] [-ro] [-q] [-where
restricted_where|@filename]
               [-spat xmin ymin xmax ymax] [-geomfield field] [-fid fid]
               [-sql statement|@filename] [-dialect sql_dialect] [-al] [-
rl] [-so] [-fields={YES/NO}]
               [-geom={YES/NO/SUMMARY}] [[-oo NAME=VALUE] ...]
               [-nomd] [-listmdd] [-mdd domain|`all`]*
               [-nocount] [-noextent]
               datasource_name [layer [layer ...]]
```

Podając jako parametr plik .shp otrzymamy krótką informację na jego temat:

```
"C:\Program Files\QGIS 3.10\bin\ogrinfo.exe" gis_osm_places_free_1.shp
```

```
INFO: Open of `gis_osm_places_free_1.shp'
      using driver `ESRI Shapefile' successful.
1: gis_osm_places_free_1 (Point)
```

Plik zawiera jedną warstwę typu `Point` o nazwie `gis_osm_places_free_1`, spróbujmy więc uzyskać ogólną informację (przełącznik `-so`) na temat tej warstwy

```
"C:\Program Files\QGIS 3.10\bin\ogrinfo.exe" -so gis_osm_places_free_1.shp
gis_osm_places_free_1
```

```
INFO: Open of `gis_osm_places_free_1.shp'
      using driver `ESRI Shapefile' successful.

Layer name: gis_osm_places_free_1
Geometry: Point
Feature Count: 7666
Extent: (19.753938, 50.185000) - (21.852383, 51.346319)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
osm_id: String (10.0)
code: Integer (4.0)
fclass: String (28.0)
population: Integer64 (10.0)
name: String (100.0)
```

Z otrzymanej informacji zapamiętujemy liczbę obiektów, czyli `7666`.

Import danych wektorowych za pomocą shp2pgsql

Opcje programu `shp2pgsql` możemy otrzymać wpisując w wierszu poleceń:

```
"c:\Program Files\PostgreSQL\12\bin\shp2pgsql.exe"
```

```
RELEASE: 2.4.3 (r16312)
USAGE: shp2pgsql [<options>] <shapefile> [[<schema>.]<table>]
OPTIONS:
  -s [<from>:]<srid> Set the SRID field. Defaults to 0.
      Optionally reprojects from given SRID (cannot be used with -D).
 (-d|a|c|p) These are mutually exclusive options:
     -d  Drops the table, then recreates it and populates
         it with current shape file data.
     -a  Appends shape file into current table, must be
         exactly the same table schema.
     -c  Creates a new table and populates it, this is the
         default if you do not specify any options.
     -p  Prepare mode, only creates the table.
  -g <geocolumn> Specify the name of the geometry/geography column
      (mostly useful in append mode).
  -D  Use postgresql dump format (defaults to SQL insert statements).
```

```
  -e  Execute each statement individually, do not use a transaction.
      Not compatible with -D.
  -G  Use geography type (requires lon/lat data or -s to reproject).
  -k  Keep postgresql identifiers case.
  -i  Use int4 type for all integer dbf fields.
  -I  Create a spatial index on the geocolumn.
  -m <filename>  Specify a file containing a set of mappings of (long)
column
     names to 10 character DBF column names. The content of the file is one
or
     more lines of two names separated by white space and no trailing or
     leading space. For example:
         COLUMNNAME DBFFIELD1
         AVERYLONGCOLUMNNAME DBFFIELD2
  -S  Generate simple geometries instead of MULTI geometries.
  -t <dimensionality> Force geometry to be one of '2D', '3DZ', '3DM', or
'4D'
  -w  Output WKT instead of WKB.  Note that this can result in
      coordinate drift.
  -W <encoding> Specify the character encoding of Shape's
      attribute column. (default: "UTF-8")
  -N <policy> NULL geometries handling policy (insert*,skip,abort).
  -n  Only import DBF file.
  -T <tablespace> Specify the tablespace for the new table.
      Note that indexes will still use the default tablespace unless the
      -X flag is also used.
  -X <tablespace> Specify the tablespace for the table's indexes.
      This applies to the primary key, and the spatial index if
      the -I flag is used.
  -?  Display this help screen.

  An argument of `--' disables further option processing.
  (useful for unusual file names starting with '-')
```

Dane konwertujemy za pomocą polecenia:

```
"c:\Program Files\PostgreSQL\12\bin\shp2pgsql.exe" -c -g way -I -S
gis_osm_places_free_1.shp public.places_1 > places.sql
```

Analiza polecenia:

- **c:\Program Files\PostgreSQL\12\bin\shp2pgsql.exe** - pełna ścieżka do programu shp2pgsql.exe
- **-c** - tworzy nową tabelę w bazie
- **-g way** - kolumna z geometrią będzie miała nazwę `way`
- **-I** - tworzy indeks przestrzenny
- **-S** - wymusza utworzenie geometrii prostych zamiast złożonych
- **gis_osm_places_free_1.shp** - nazwa pliku, który będziemy przetwarzać
- **public.places_1** - nazwa schematu i tabeli w bazie do której będziemy importować

- **> places.sql** - przekierowanie wyjścia aplikacji do pliku `places.sql`

Jeśli wszytko przebiegło prawidłowo polecenie powinno wyświetlić na konsoli następujący komunikat:

```
Shapefile type: Point
Postgis type: POINT[2]
```

W bieżącym folderze powinien również zostać utworzony plik `places.sql` zawierający skrypt sql

```
SET CLIENT_ENCODING TO UTF8;
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "public"."places_1" (
  gid serial,
  "osm_id" varchar(10),
  "code" int2,
  "fclass" varchar(28),
  "population" int8,
  "name" varchar(100)
);
ALTER TABLE "public"."places_1" ADD PRIMARY KEY (gid);
SELECT AddGeometryColumn('public','places_1','way','0','POINT',2);
INSERT INTO "public"."places_1"
("osm_id","code","fclass","population","name",way) VALUES
('31174958','1002','town','47538','Skarżysko-
Kamienna','01010000005CCB64389EDB3440C68C4BB0938E4940');
INSERT INTO "public"."places_1"
("osm_id","code","fclass","population","name",way) VALUES
('31532321','1003','village','0','Dąbrowa','01010000004447BC862A123440BCBA1
92433954940');
...
INSERT INTO "public"."places_1"
("osm_id","code","fclass","population","name",way) VALUES
('7795318056','1050','locality','0','Kobyłka','01010000009D685721E57934401A
B2704859614940');
CREATE INDEX ON "public"."places_1" USING GIST ("way");
COMMIT;
ANALYZE "public"."places_1";
```

Powstały w ten sposób skrypt zaimportujemy do bazy za pomocą aplikacji `psql`.

```
"c:\Program Files\PostgreSQL\12\bin\psql.exe" -f places.sql -U postgres gis
```

Analiza polecenia:

- **c:\Program Files\PostgreSQL\12\bin\psql.exe** - pełna ścieżka do aplikacji
- **-f places.sql** - uruchamia skrypt z pliku `places.sql`

- **-U postgres** - połącz używając użytkownika `postgres`
- **gis** - nazwa bazy danych

Aplikacja poprosi o podanie hasła dla użytkownika `postgres` po czym uruchomi skrypt. Jeśli wszystko przebiegnie prawidłowo wynik działania programu powinien wyglądać następująco:

```
SET
SET
BEGIN
CREATE TABLE
ALTER TABLE
               addgeometrycolumn
-----------------------------------------------
 public.places_1.way SRID:0 TYPE:POINT DIMS:2
(1 row)

INSERT 0 1
(...)
INSERT 0 1
CREATE INDEX
COMMIT
ANALYZE
```

Poprawność importu możemy sprawdzić w aplikacji `pgAdmin` - w bazie danych `gis`, w schemacie `public` powinna pojawić się tabela `places_1` zawierająca `7666` rekordów, lub za pomocą tej samej aplikacji którą dokonaliśmy importu, używając przełącznika `-c` i podając zapytanie SQL, które chcemy wykonać na bazie. Całe polecenie wygląda następująco:

```
"C:\Program Files\PostgreSQL\12\bin\psql.exe" -U postgres -c "select
count(*) from places_1;" gis
```

W odpowiedzi otrzymamy:

```
 count
-------
  7666
(1 row)
```

co jest wartością zgodną z ilością obiektów w warstwie, którą otrzymaliśmy za pomocą aplikacji `ogrinfo`.

Import danych wektorowych za pomocą ogr2ogr

Aplikacja `ogr2ogr` to potężne narzędzie pozwalające na konwersję danych przestrzennych między kilkudziesięcioma formatami danych, jak i na wykonywanie na nich innych operacji jak filtrowanie, zmiana kodowania czy reprojekcje. W tym kroku zaimportujemy z tego samego pliku wyłącznie rekordy wskazujące

na miasta. Zgodnie z dokumentacją klucza place=* w OpenStreetMap miasta oznaczamy jako `city` oraz `town`. Sprawdźmy najpierw ile takich obiektów mamy w naszym pliku .shp:

```
"C:\Program Files\QGIS 3.10\bin\ogrinfo.exe" -so -where "fclass='city' or
fclass='town'" gis_osm_places_free_1.shp gis_osm_places_free_1
```

```
INFO: Open of `gis_osm_places_free_1.shp'
      using driver `ESRI Shapefile' successful.

Layer name: gis_osm_places_free_1
Geometry: Point
Feature Count: 45
Extent: (19.753938, 50.185000) - (21.852383, 51.346319)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
osm_id: String (10.0)
code: Integer (4.0)
fclass: String (28.0)
population: Integer64 (10.0)
name: String (100.0)
```

OGRInfo twierdzi, że takich obiektów mamy 45. Dzięki temu, że w poprzednim kroku zaimportowaliśmy cały ten plik do bazy możemy również sprawdzić, czy baza danych zwróci taką samą wartość:

```
"C:\Program Files\PostgreSQL\12\bin\psql.exe" -U postgres -c "select
count(*) from places_1 where fclass in ('city', 'town');" gis
```

```
 count
-------
    45
(1 row)
```

Użyjmy więc `ogr2ogr` aby zaimportować tylko te 45 obiektów do nowej tabeli w bazie:

```
"C:\Program Files\QGIS 3.10\bin\ogr2ogr.exe" -f "PostgreSQL"
"PG:host=127.0.0.1 user=postgres dbname=gis password=gis"
"gis_osm_places_free_1.shp" -lco GEOMETRY_NAME=way -nln places_2 -overwrite
-where "fclass='city' or fclass='town'"

Jeśli polecenie zostanie wykonane prawidłowo na konsole nie zostanie
wyświetlony żaden komunikat.

Prawidłowość importu jak poprzednio możemy sprawdzić za pomocą `psql`:

```cmd
"C:\Program Files\PostgreSQL\12\bin\psql.exe" -U postgres -c "select
count(*) from places_2 ;" gis
```

```
 count
-------
    45
(1 row)
```

Jak widzimy w bazie danych pojawiła się nowa tabela o nazwie `places_2`, w której mamy 45 obiektów, co zgadza się z liczbą obiektów, które miały zostać zaimportowane z pliku .shp.

Eksport danych wektorowych za pomocą pgsql2shp

Pgsql2shp to aplikacja bliźniacza do omówionej w jednym z poprzednich rozdziałów aplikacji shp2pgsql, jednak o odwrotnym działaniu - pozwala ona na eksport danych z bazy do pliku .shp.

Jak w poprzednim przypadku listę opcji programu możemy uzyskać uruchamiając go bez żadnych parametrów:

```
"C:\Program Files\PostgreSQL\12\bin\pgsql2shp.exe"
```

```
RELEASE: 3.0.1 (3.0.1)
USAGE: pgsql2shp [<options>] <database> [<schema>.]<table>
       pgsql2shp [<options>] <database> <query>

OPTIONS:
  -f <filename>  Use this option to specify the name of the file to create.
  -h <host>  Allows you to specify connection to a database on a
     machine other than the default.
  -p <port>  Allows you to specify a database port other than the default.
  -P <password>  Connect to the database with the specified password.
  -u <user>  Connect to the database as the specified user.
  -g <geometry_column> Specify the geometry column to be exported.
  -b Use a binary cursor.
  -r Raw mode. Do not assume table has been created by the loader. This
```
```

```
would
     not unescape attribute names and will not skip the 'gid' attribute.
  -k Keep PostgreSQL identifiers case.
  -m <filename>  Specify a file containing a set of mappings of (long)
column
     names to 10 character DBF column names. The content of the file is one
or
     more lines of two names separated by white space and no trailing or
     leading space. For example:
         COLUMNNAME DBFFIELD1
         AVERYLONGCOLUMNNAME DBFFIELD2
  -? Display this help screen.
```

Aby wyeksportować dane używamy polecenia:

```
"C:\Program Files\PostgreSQL\12\bin\pgsql2shp.exe" -f places_2.shp -h
127.0.0.1 -p 5432 -u postgres -P gis gis public.places_2
```

Analiza polecenia:

- "C:\Program Files\PostgreSQL\12\bin\pgsql2shp.exe" - ścieżka do aplikacji
- -f places_2.shp - zapisujemy w pliku `places_2.shp`
- -h 127.0.0.1 - adres IP serwera bazy to `127.0.0.1`
- -p 5432 - port na którym baza nasłuchuje to `5432`
- -u postgres - połącz jako użytkownik `postgres`
- -P gis - użyj hasła `gis`
- gis - połącz do bazy danych `gis`
- public.places_2 - użyj tabeli `places_2` w schemacie `public`

Jeśli uruchomienie zakończy się poprawnie na konsoli powinno zostać wyświetlone:

```
Initializing...
Done (postgis major version: 3).
Output shape: Point
Dumping: X [45 rows].
```

Poprawność eksportu możemy sprawdzić za pomocą aplikacji `ogrinfo`:

```
"C:\Program Files\QGIS 3.10\bin\ogrinfo.exe" -so places_2.shp places_2
```

```
INFO: Open of `places_2.shp'
      using driver `ESRI Shapefile' successful.

Layer name: places_2
```

```
Geometry: Point
Feature Count: 45
Extent: (19.966777, 50.242587) - (21.852383, 51.191022)
Layer SRS WKT:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["latitude",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["longitude",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
OGC_FID: Integer64 (11.0)
OSM_ID: String (10.0)
CODE: Real (32.10)
FCLASS: String (28.0)
POPULATION: Real (32.10)
NAME: String (100.0)
```

Wyeksportowanych zostało 45 obiektów, czyli liczba jest zgodna z zawartością bazy danych.

Eksport danych wektorowych za pomocą ogr2ogr

Analogicznie do eksportu z bazy danych możemy użyć również aplikacji `ogr2ogr`, z tą jednak różnicą, że w tym przypadku nie jesteśmy ograniczeni do formatu ESRI Shapefile - aplikacja potrafi odczytywać i zapisywać kilkadziesiąt różnych formatów przestrzennych.

```
ogr2ogr -f "GeoJSON" places_2.json PG:"host=127.0.0.1 dbname=gis
user=postgres password=gis port=5432" "public.places_2"
```

Jeśli uruchomienie przebiegnie prawidłowo aplikacja nie wyświetli na konsoli żadnego komunikatu. Analogicznie do poprzedniej sytuacji poprawność eksportu możemy sprawdzić aplikacją `ogrinfo`:

```
"C:\Program Files\QGIS 3.10\bin\ogrinfo.exe" -so places_2.json places_2
```

```
INFO: Open of `places_2.json'
      using driver `GeoJSON' successful.

Layer name: places_2
```

```
Geometry: Point
Feature Count: 45
Extent: (19.966777, 50.242587) - (21.852383, 51.191022)
Layer SRS WKT:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
osm_id: String (0.0)
code: Integer (0.0)
fclass: String (0.0)
population: Integer (0.0)
name: String (0.0)
```

Tym razem również ilość wyeksportowanych obiektów to 45.

Import danych openstreetmap za pomocą osm2pgsql

osm2pgsql to kolejna aplikacja terminalowa pozwalająca na import danych do bazy PostgreSQL - w jej przypadku importujemy dane projektu OpenStreetMap w ich natywnym formacie.

OpenStreetMap (skrótowo OSM) to globalny projekt mający na celu stworzenie darmowej oraz swobodnie dostępnej mapy świata. Co najważniejsze, może być edytowana przez każdego, przez co nazywana jest także kartograficznym odpowiednikiem Wikipedii.

Wszelkie dane zawarte w projekcie udostępniane są na otwartej licencji Open Database License (ODbL), co umożliwia dalsze ich wykorzystywanie, między innymi w systemach nawigacji GPS, co czyni ją darmową alternatywą dla drogich, niekoniecznie lepszych, map komercyjnych.

Model danych projektu różni się zasadniczo od modeli znanych ogólnie w GIS i opiera się w znacznym stopniu na relacjach, a jedynym nośnikiem geometrii są punkty. Model właściwości obiektów również jest dynamiczny i w znacznym stopniu nieznormalizowany. Powyższy model w przypadku odwzorowania w bazie danych wprost byłby trudny w użyciu dlatego do importu użyjemy narzędzia które na podstawie punktów i relacji zbuduje dla nas gotowe obiekty przestrzenne i umieści je w osobnych tabelach bazy danych.

Wersja importera dla systemu Windows możliwa jest do pobrania pod adresem

https://ci.appveyor.com/project/openstreetmap/osm2pgsql/build/artifacts

Dostępna jest również w formie archiwum w plikach szkoleniowych (osm2pgsql_Release_x64.zip).

Na potrzeby szkolenia sugerujemy aby archiwum rozpakować , po czym folder znajdujący się w środku przenieść do katalogu głównego na dysku C: komputera.

Aby rozpocząć import danych należy przejść do folderu w którym znajduje się pobrany wcześniej ekstrakt danych OSM i uruchomić aplikację:

```
"c:\osm2pgsql-bin\osm2pgsql.exe" -d gis -U postgres -W -H 127.0.0.1 -c -j -
S "C:\osm2pgsql-bin\empty.style" swietokrzyskie-latest.osm.pbf
```

Analiza polecenia:

- **"C:\osm2pgsql-bin\osm2pgsql.exe"** - ścieżka do aplikacji
- **-d gis** - nazwa bazy do której będziemy importowali
- **-U postgres** - nazwa użytkownika
- **-W** - wymuś podanie hasła
- **-H 127.0.0.1** - adres serwera bazy danych
- **-c** - utwórz nowy zestaw danych
- **-j** - umieść wszystkie właściwości w kolumnie typu hstore
- **-S "C:\osm2pgsql-bin\default.style"** - plik konfiguracyjny
- **swietokrzyskie-latest.osm.pbf** - nazwa pliku z ekstraktem danych OSM

Jeśli import przebiegnie prawidłowo na konsoli powinien pojawić się poniższy zestaw komunikatów:

```
Allocating memory for sparse node cache
Node-cache: cache=800MB, maxblocks=12800*65536, allocation method=1
Using built-in tag processing pipeline
Using projection SRS 3857 (Spherical Mercator)
Setting up table: planet_osm_point
Setting up table: planet_osm_line
Setting up table: planet_osm_polygon
Setting up table: planet_osm_roads

Reading in file: ..\swietokrzyskie-latest.osm.pbf
Using PBF parser.
Processing: Node(5990k 2995.0k/s) Way(918k 114.75k/s) Relation(5343
5343.0/s)  parse time: 12s
Node stats: total(5990645), max(7802006353) in 2s
Way stats: total(918432), max(835879893) in 8s
Relation stats: total(5343), max(11487477) in 1s
node cache: stored: 5990645(100.00%), storage efficiency: 50.00% (dense
blocks: 0, sparse nodes: 5990645), hit rate: 100.00%
Sorting data and creating indexes for planet_osm_point
Sorting data and creating indexes for planet_osm_line
Sorting data and creating indexes for planet_osm_polygon
Sorting data and creating indexes for planet_osm_roads
Using native order for clustering
Using native order for clustering
Using native order for clustering
Using native order for clustering
```

```
Copying planet_osm_roads to cluster by geometry finished
Creating geometry index on planet_osm_roads
Creating indexes on planet_osm_roads finished
All indexes on planet_osm_roads created in 1s
Completed planet_osm_roads
Copying planet_osm_point to cluster by geometry finished
Creating geometry index on planet_osm_point
Copying planet_osm_line to cluster by geometry finished
Creating geometry index on planet_osm_line
Creating indexes on planet_osm_point finished
All indexes on planet_osm_point created in 3s
Completed planet_osm_point
Creating indexes on planet_osm_line finished
All indexes on planet_osm_line created in 3s
Completed planet_osm_line
Copying planet_osm_polygon to cluster by geometry finished
Creating geometry index on planet_osm_polygon
Creating indexes on planet_osm_polygon finished
All indexes on planet_osm_polygon created in 17s
Completed planet_osm_polygon

Osm2pgsql took 30s overall
```

W bazie danych powinny pojawić się nowe tabele z zaimportowanymi danymi - domyślnie ich nazwy rozpoczynają się od `planet_osm_`. Możemy to sprawdzić za pomocą `psql`

```
"C:\Program Files\PostgreSQL\12\bin\psql.exe" -U postgres -d gis -c "\dt"
```

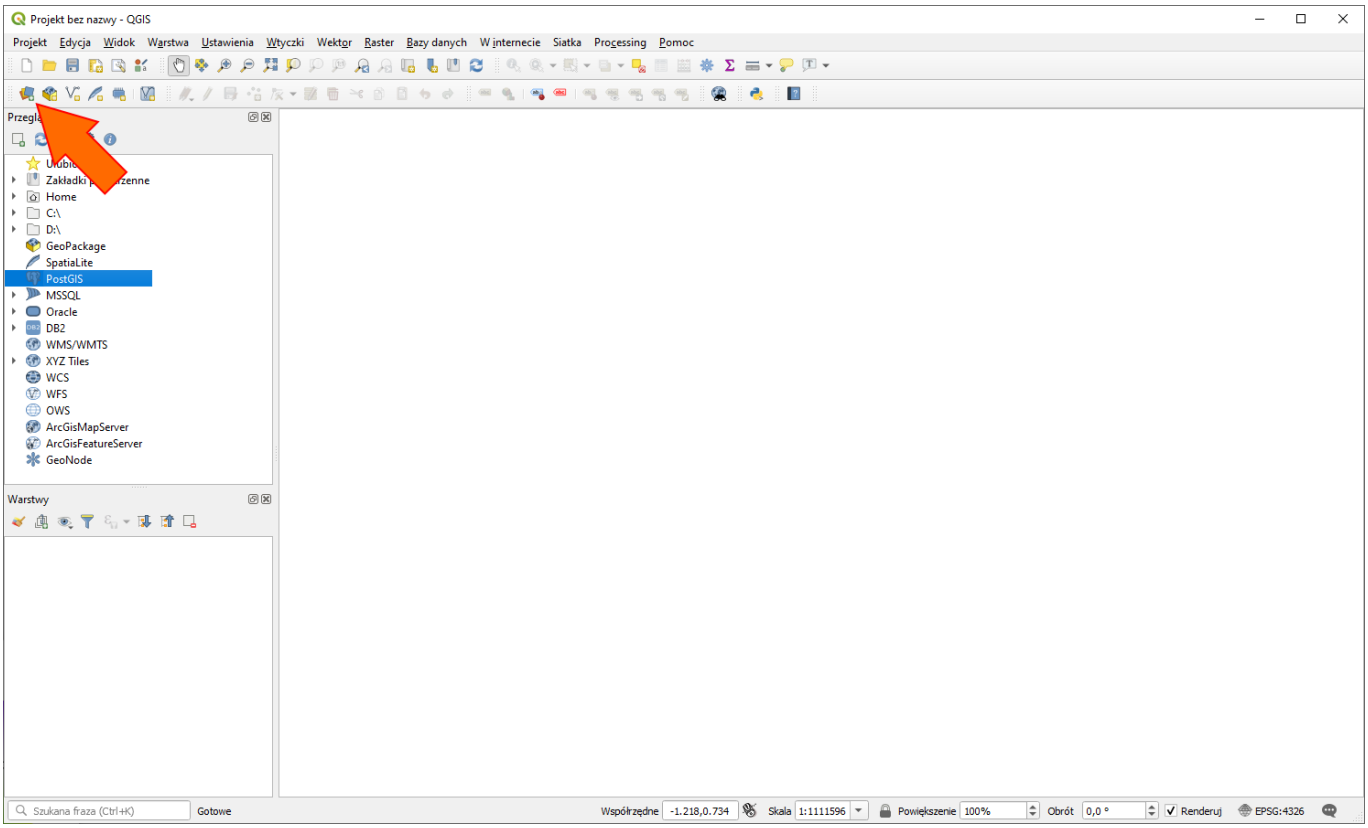
```
              List of relations
 Schema |        Name         | Type  |  Owner
--------+---------------------+-------+----------
 public | places_1            | table | postgres
 public | places_2            | table | postgres
 public | planet_osm_line     | table | postgres
 public | planet_osm_nodes    | table | postgres
 public | planet_osm_point    | table | postgres
 public | planet_osm_polygon  | table | postgres
 public | planet_osm_rels     | table | postgres
 public | planet_osm_roads    | table | postgres
 public | planet_osm_ways     | table | postgres
 public | spatial_ref_sys     | table | postgres
(10 rows)
```

Z naszego punktu widzenia najistotniejsze dane znajdują się w tabelach `planet_osm_point`, `planet_osm_line` oraz `planet_osm_polygon` - odpowiednio punkty, linie i obszary zaimportowane z danych OSM.

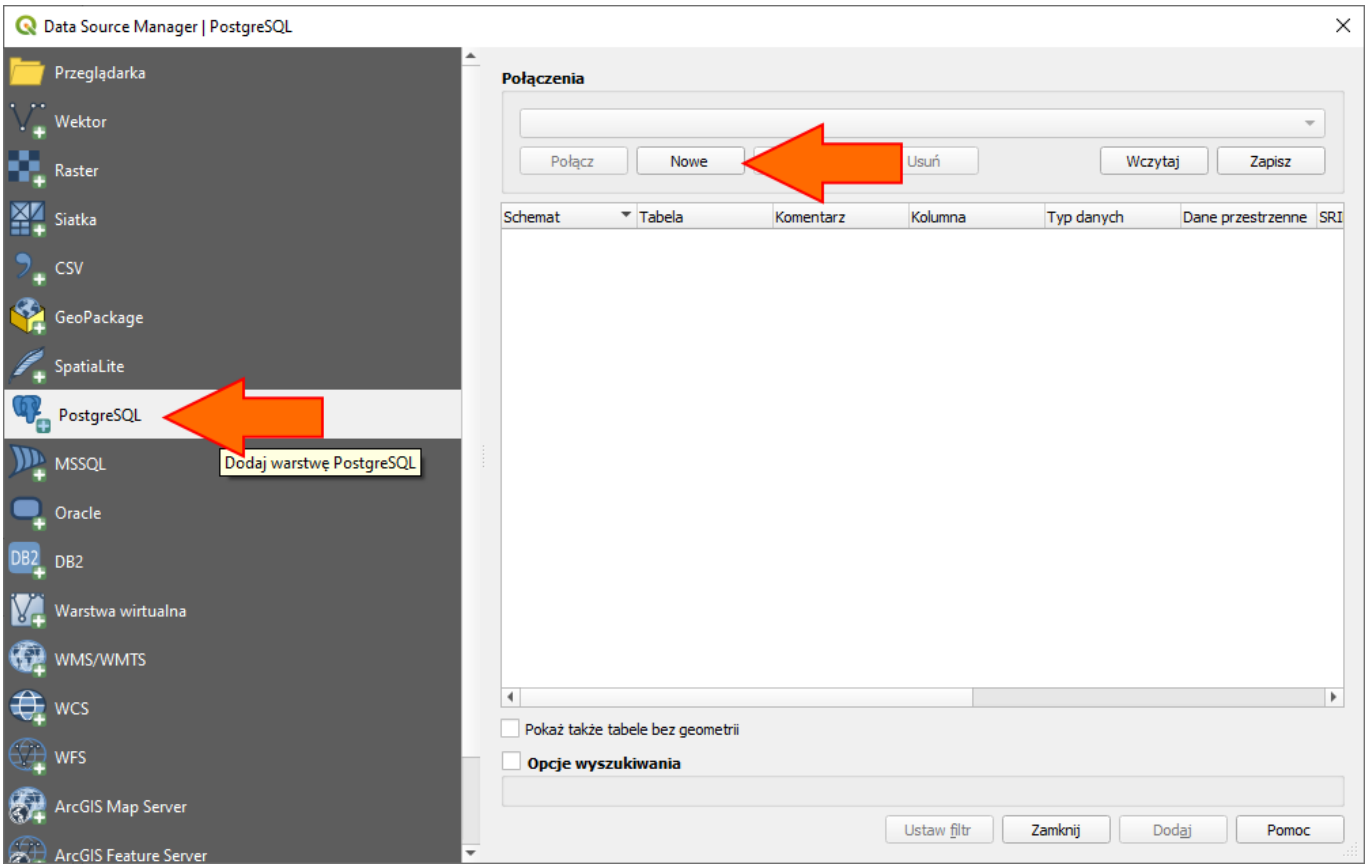
Blok 5 - Obsługa bazy za pomocą QGIS

Połączenie z bazą danych

Aby utworzyć nowe połączenie z bazą danych należy otworzyć okno `Zarządzanie źródłami danych` wybierając `menu > warstwa > zarządzanie źródłami danych` lub przez odpowiednią ikonę na pasku narzędzi



W oknie zarządzania źródłami danych zaznaczamy `PostgreSQL`, po czym klikamy przycisk `Nowe`



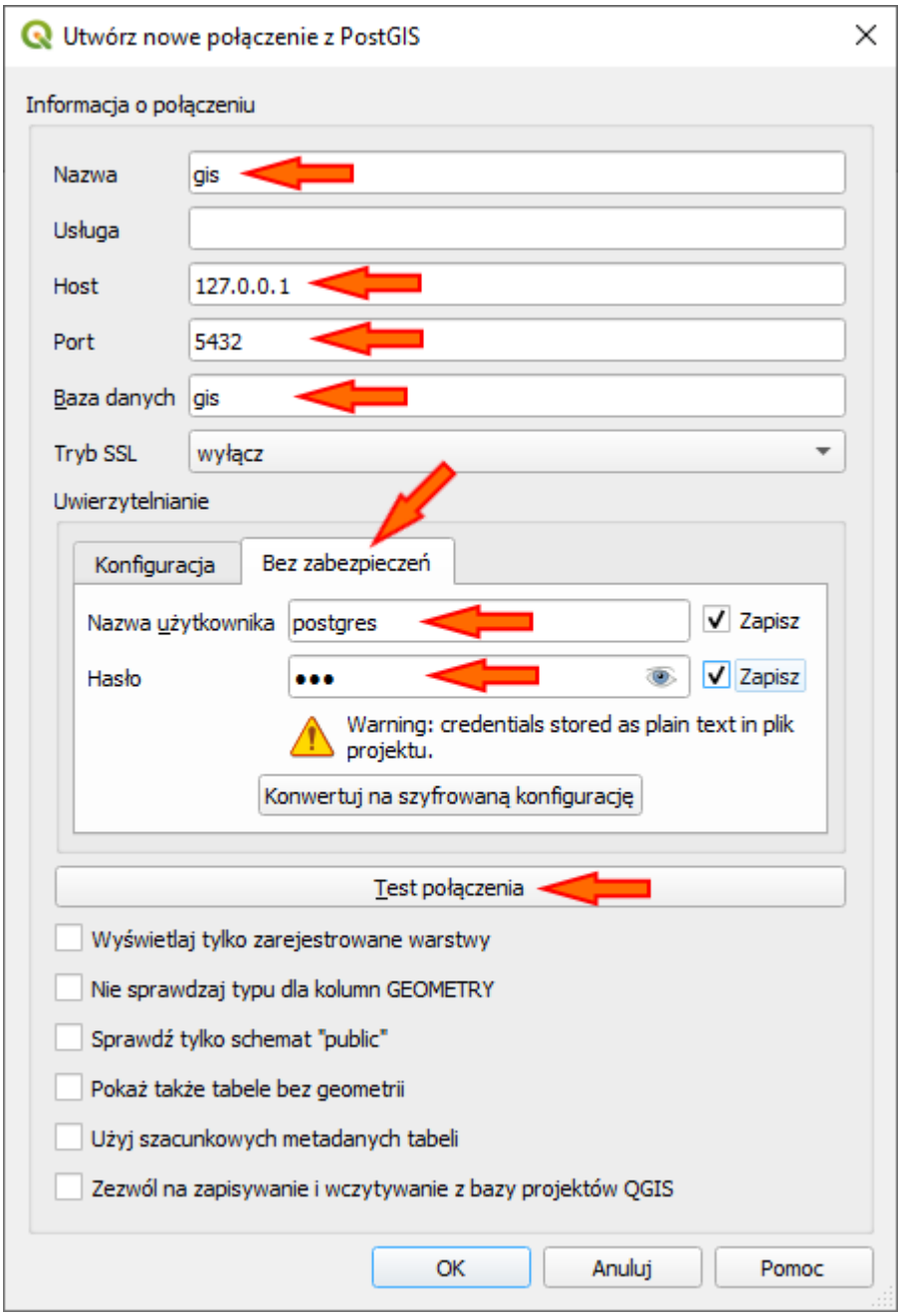
Pojawia się okno konfiguracji nowego połączenia, w którym wypełniamy kolejno następujące pola:

- **Nazwa** - tu podajemy nazwę połączenia (dowolnie)
- **Host** - adres IP serwera bazy danych - w naszym przypadku `127.0.0.1`
- **Port** - port na którym nasłuchuje serwer - w naszym przypadku domyślny `5432`
- **Baza danych** - nazwa bazy danych - `gis`

Następnie w ramce `Uwierzytelnianie` wybieramy zakładkę `Bez zabezpieczeń`, gdzie podajemy:

- **Nazwa użytkownika** - w naszym przypadku `postgres`
- **Hasło** - w naszym przypadku `gis`

Zaznaczamy również pola `Zapisz` przy obu polach tekstowych.

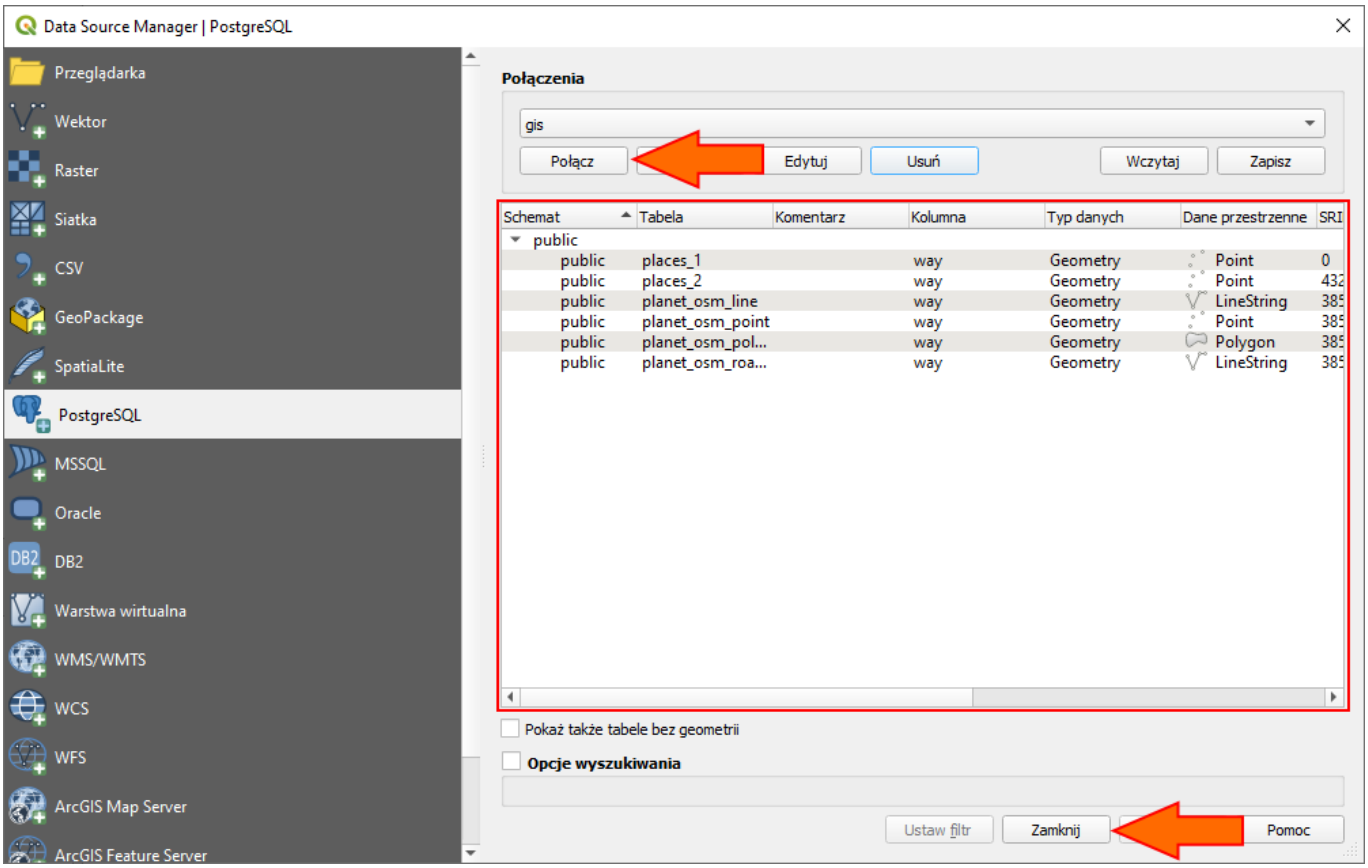


Uwaga !

Przy takiej konfiguracji podane poświadczenia będą zapisane otwartym tekstem w plikach projektu.

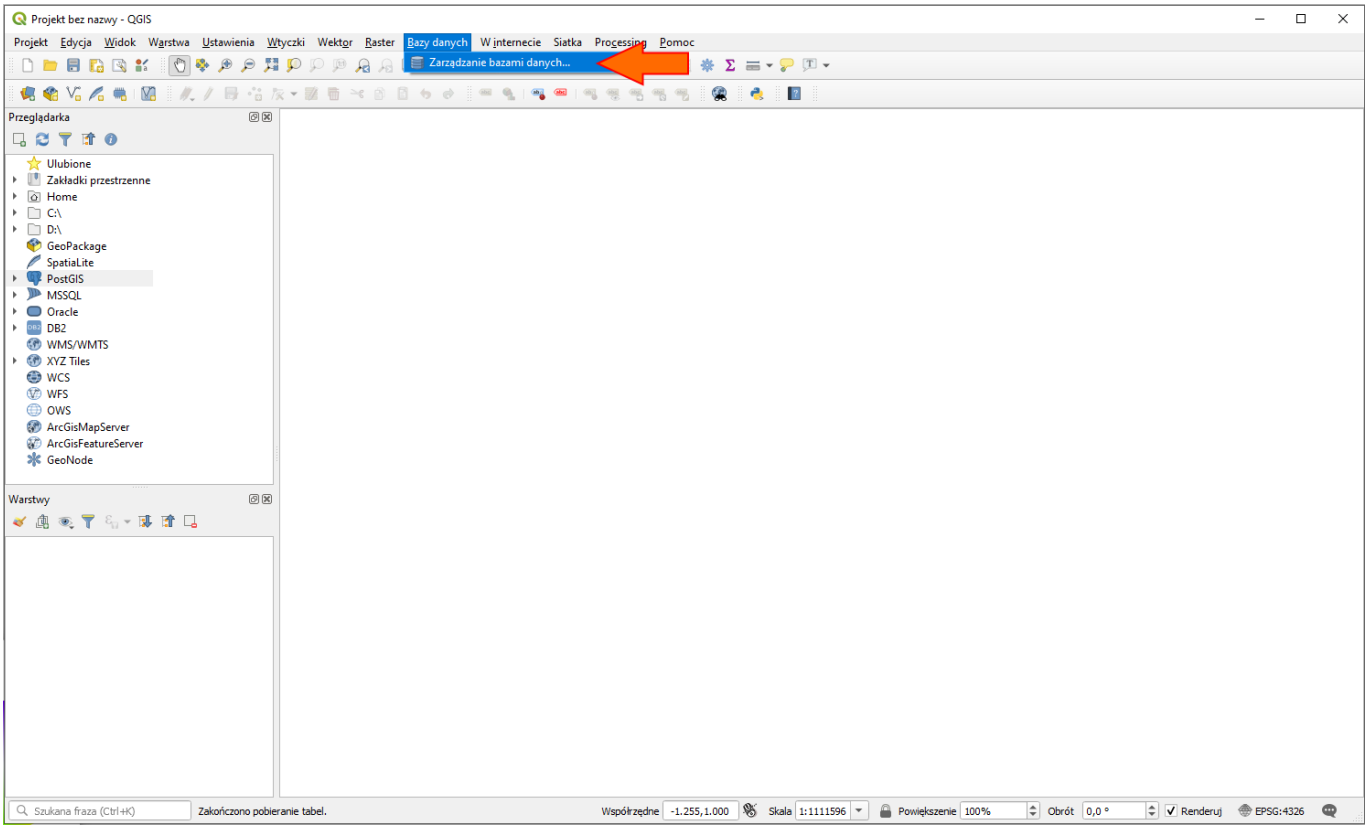
Jeśli `Test połączenia` przebiegł pomyślnie zamykamy okno przyciskiem `OK`.

W oknie `Zarządzanie źródłami danych` wybieramy utworzone połączenie i klikamy `Połącz`. Jeśli wszystko przebiegło prawidłowo w ramce poniżej powinien pojawić się schemat `public` a po jego rozwinięciu powinny być widoczne tabele naszej bazy danych.

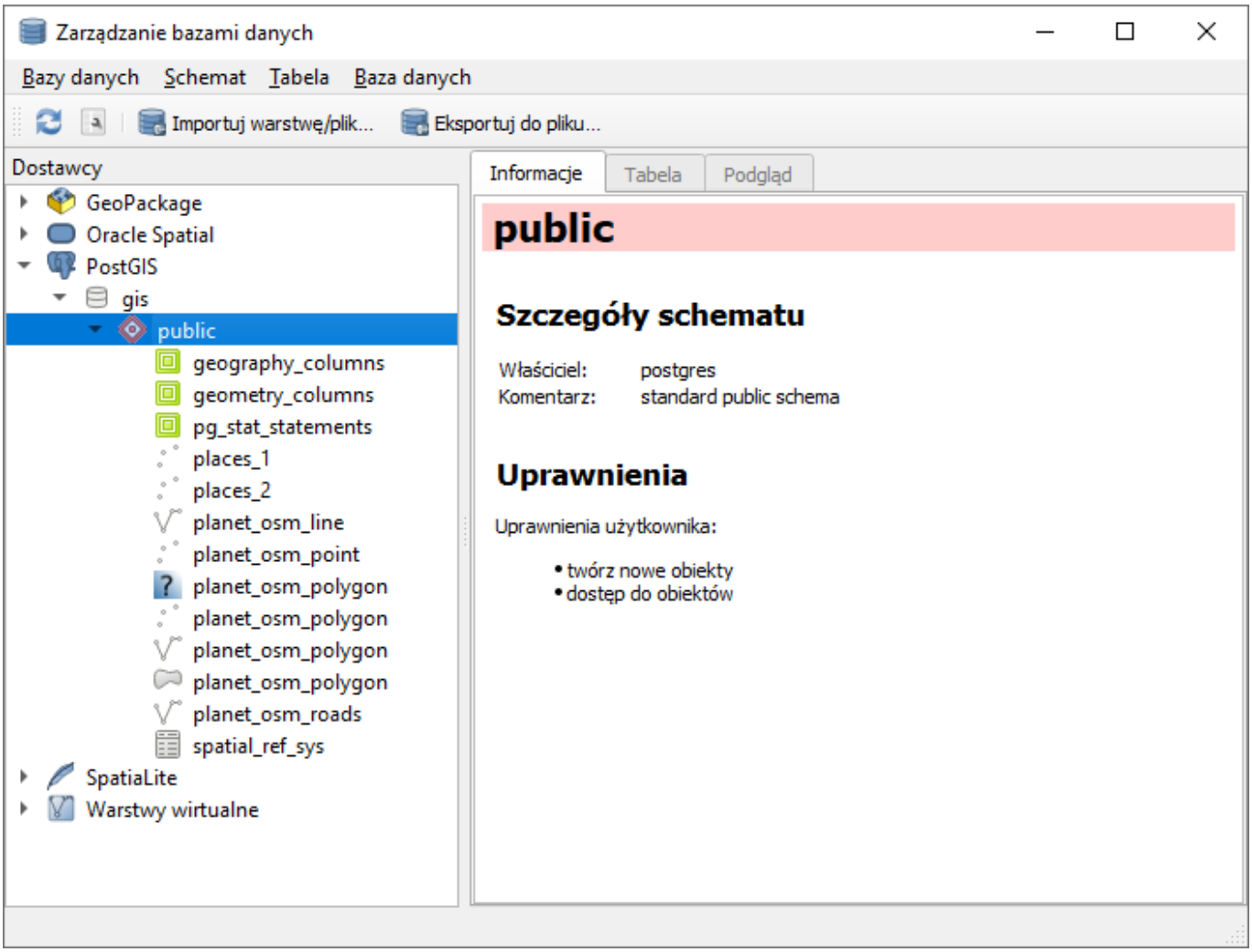


Tworzenie nowego schematu i tabeli

Do zarządzania danymi w bazie danych używamy okna `Zarządzanie bazami danych` dostępnego po wybraniu z `menu > Bazy danych > Zarządzanie bazami danych`

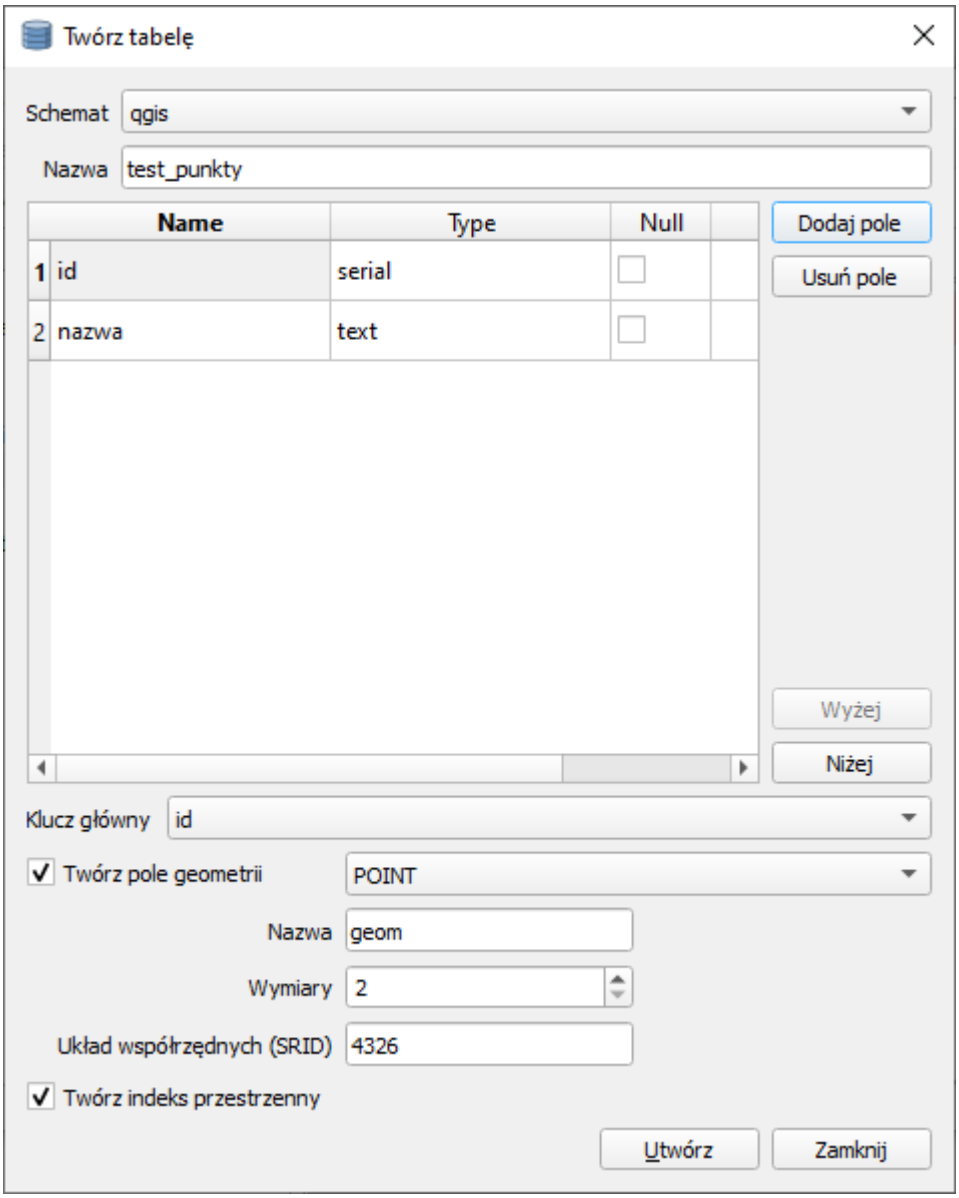


Po otwarciu okna w ramce `Dostawcy` rozwijamy `PostGIS` > `public` i ponownie powinny tam być widoczne tabele z naszej bazy.



Na potrzeby kolejnych kroków szkolenia utworzymy nowy schemat i tabelę.

Nowy schemat tworzymy wybierając `menu > Schemat > Utwórz schemat...` i w kolejnym oknie podajemy nazwę schematu `qgis`. Schemat powinien pojawić się w drzewie, więc zaznaczamy go i wybieramy `menu > Tabela > Utwórz tabelę...`.



W oknie tworzenia tabeli wykonujemy kolejne kroki:

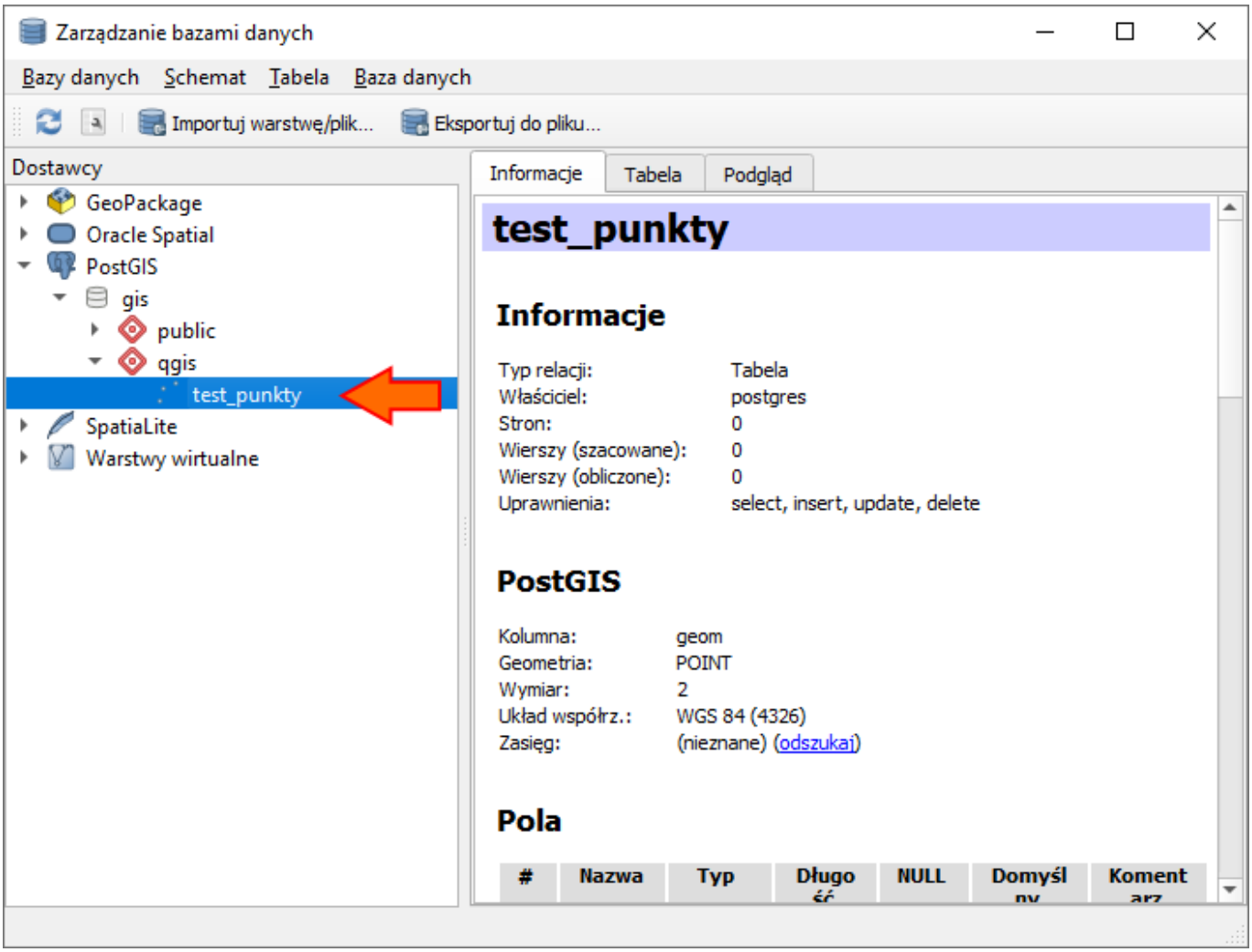
- podajemy nazwę tabeli: `test_punkty`
- wybieramy `Dodaj pole`
- w tabeli zmieniamy nazwę pola na `id`, typ zostawiamy jako `serial`
- wybieramy `Dodaj pole`
- w tabeli zmieniamy nazwę pola na `nazwa` i typ na `text`
- jako `Klucz główny` wybieramy `id`
- zaznaczamy `Twórz pole geometrii`
- jako typ geometrii wybieramy `POINT`
- podajemy nazwę pola geometrii `geom`
- podajemy wymiary `2`
- podajemy układ współrzędnych jako `4326`
- zaznaczamy `Twórz indeks przestrzenny`

Kilka słów wyjaśnienia:

- typ danych `serial` to typ specyficzny dla PostgreSQL - jest oparty na generatorze - dzięki niemu możemy automatycznie i unikalnie numerować rekordy w tabeli

- klucz główny tabeli musi być ustawiony jawnie jeśli dane w tej tabeli mają być edytowane w QGIS
- indeks przestrzenny pozwala na znaczne skrócenie zapytań opartych o warunki przestrzenne

Na zakończenie klikamy przycisk `Utwórz` - nowa tabela powinna pojawić się w drzewie obiektów, a po dwukliku powinna zostać dodana do warstw projektu.

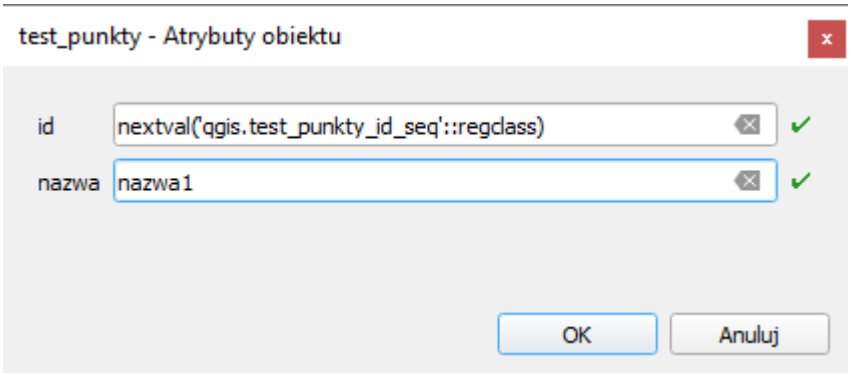


Edycja danych w bazie

Edycja warstwy utworzonej na podstawie bazy danych nie różni się zasadniczo ode edycji każdej innej warstwy - dokonujemy tego używając paska narzędzi `Digitalizacja`. Przed rozpoczęciem należy przełączyć warstwę w tryb edycji.

Obiekty dodajemy za pomocą odpowiedniego narzędzia.

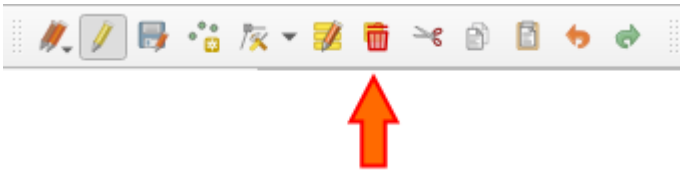
Po wybraniu i kliknięciu na mapie (lub wyrysowaniu kształtu) wyświetlone zostanie okno w którym możemy podać właściwości utworzonego obiektu. Należy zauważyć, że pole `id` o typie `serial` jest już wypełnione wywołaniem funkcji, która pobierze kolejną wartość z generatora. Wartości tej nie należy zmieniać, a w przypadku projektów produkcyjnych można zablokować jej edycję w QGIS.



Kolejne narzędzie pozwala na edycję wierzchołków geometrii.

Jeśli na warstwie zaznaczymy jeden lub kilka obiektów mamy również możliwość edycji ich właściwości

lub usunięcia ich.



Kolejne funkcje, czyli `Wytnij`, `Kopiuj`, `Wklej`, `Cofnij` oraz `Powtórz` działają analogicznie do wszystkich innych aplikacji, więc zakładamy, że omawianie ich funkcjonalności nie jest konieczne.

Po każdej edycji należy pamiętać, aby zapisać warstwę używając przycisku `Zapisz` znajdującego się na pasku `Digitalizacja`

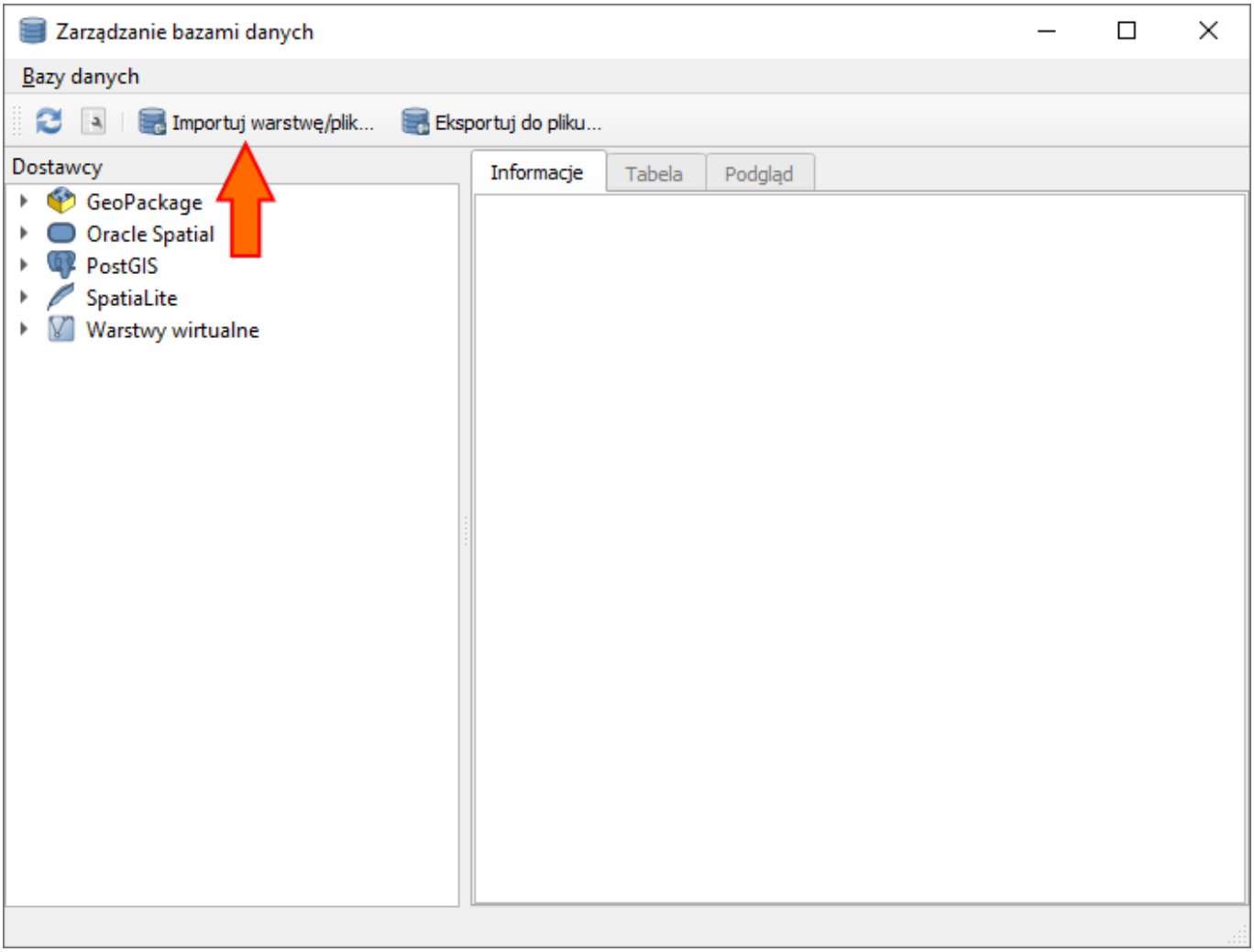
i wyłączenie trybu edycji warstwy.

Jest to o tyle istotne, że **zapis projektu nie powoduje zapisu edycji w warstwach**, co jest częstą przyczyną błędów początkujących użytkowników i utraty wykonanych edycji.

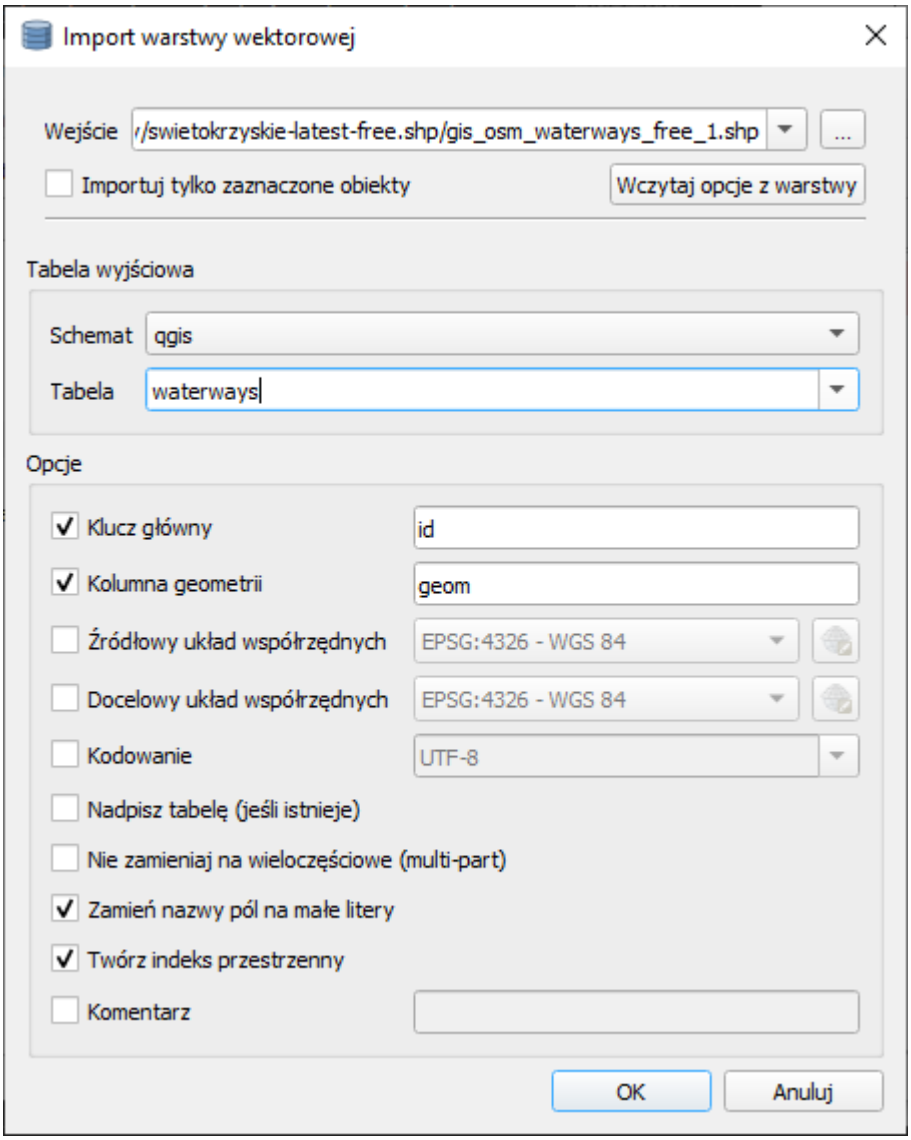
Import danych do bazy

Aplikacja QGIS pozwala również na importy danych do bazy. Są one wolniejsze od przedstawionych metod terminalowych, ale dla użytkowników przyzwyczajonych do pracy z interfejsami graficznymi mogą być wygodniejsze.

Aby zaimportować nową tabelę do bazy w oknie `Zarządzanie bazami danych` klikamy przycisk `Importuj warstwę/plik`



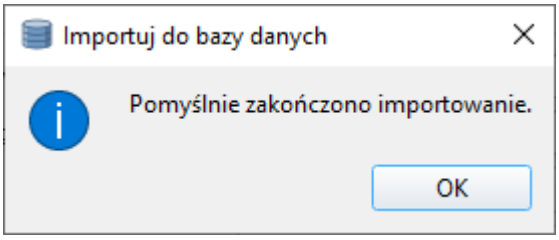
Wyświetlone zostanie okno importu.



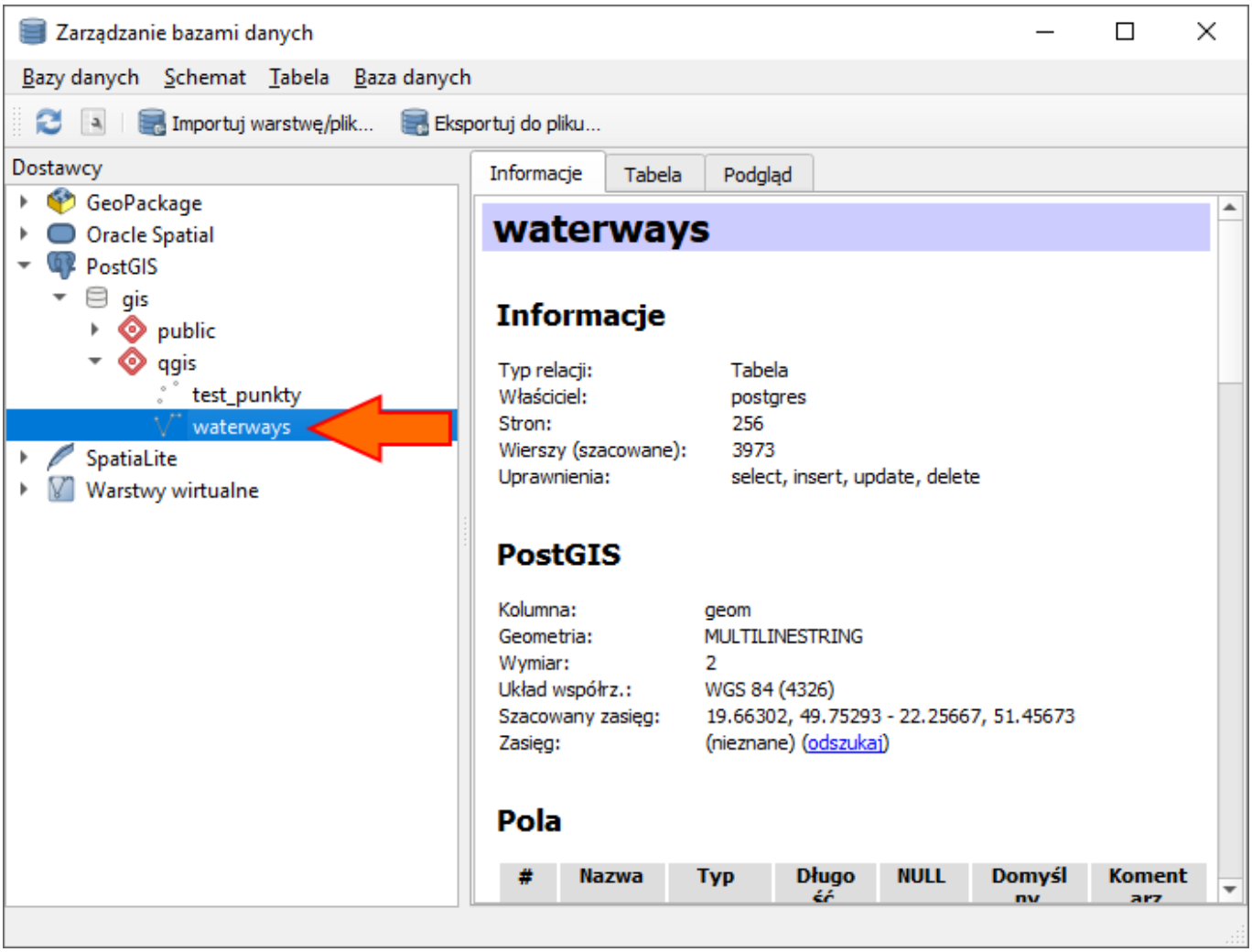
Jako wejście możemy podać dowolny plik z danymi przestrzennymi znajdujący się na dysku, lub warstwę wcześniej dodaną do projektu. W przypadku warstwy mamy również możliwość zaimportowania tylko obiektów, które są na niej zaznaczone. Na potrzeby szkolenia zaimportujemy warstwę `gis_osm_waterways_free_1.shp`

Po wybraniu przycisku `Wczytaj opcje z warstwy` pola w kolejnych ramkach zostaną wypełnione. Mamy możliwość zmiany schematu do którego importujemy oraz nazwy tabeli - tutaj ustawimy `waterways`. Klucz główny będzie miał nazwę `id` a pole geometrii `geom`. Nazwy pól zmienimy na małe litery i utworzymy indeks przestrzenny.

Na koniec wybieramy `OK` i jeśli QGIS nie napotka żadnych problemów powinniśmy otrzymać informację że warstwa została zaimportowana do bazy,

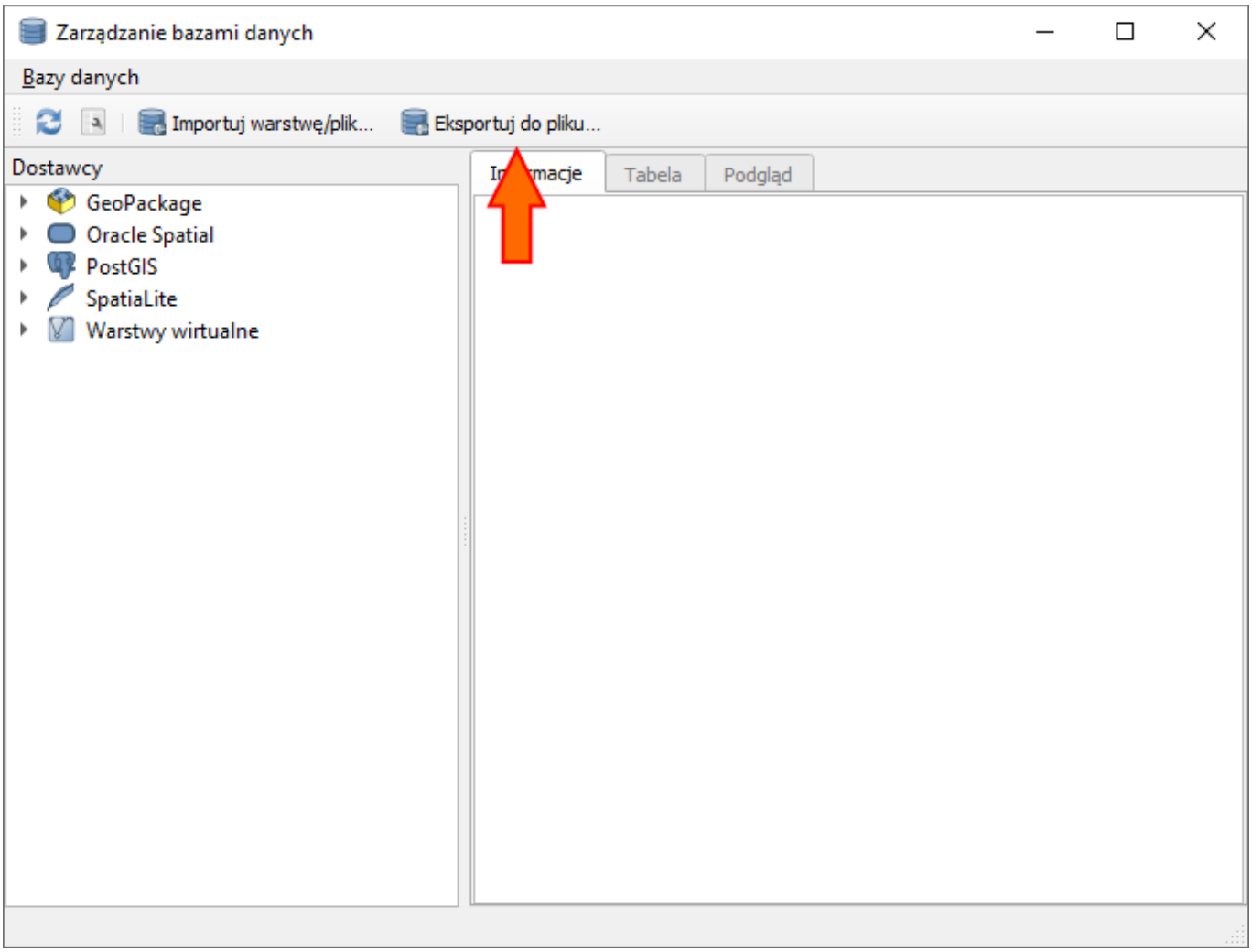


co możemy potwierdzić w oknie `Zarządzania bazami danych`.

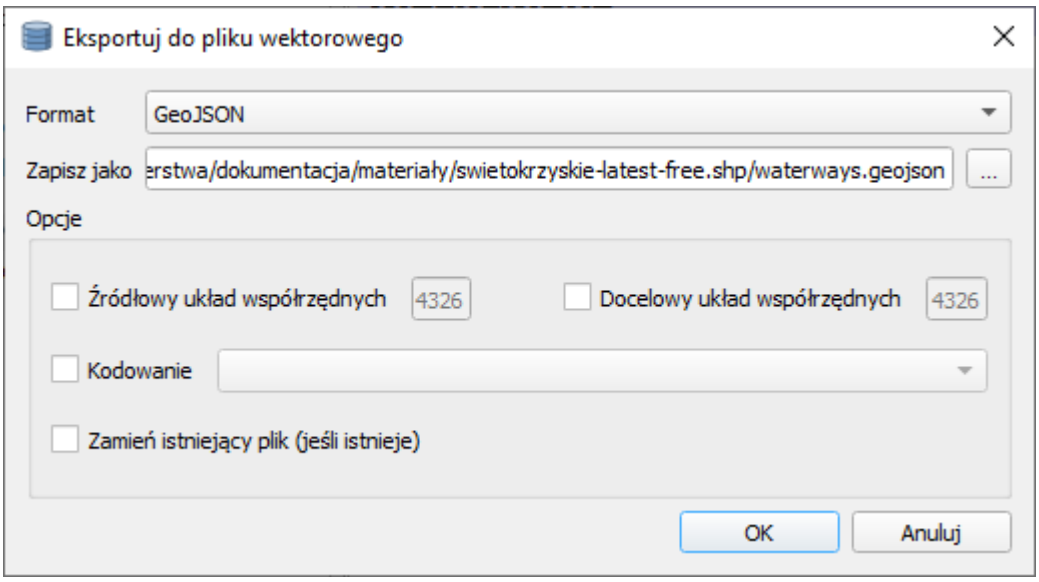


Eksport danych z bazy

Aby wyeksportować dane z bazy w oknie `Zarządzanie bazami danych` zaznaczamy tabelę którą chcemy wyeksportować i klikamy przycisk `Eksportuj do pliku`

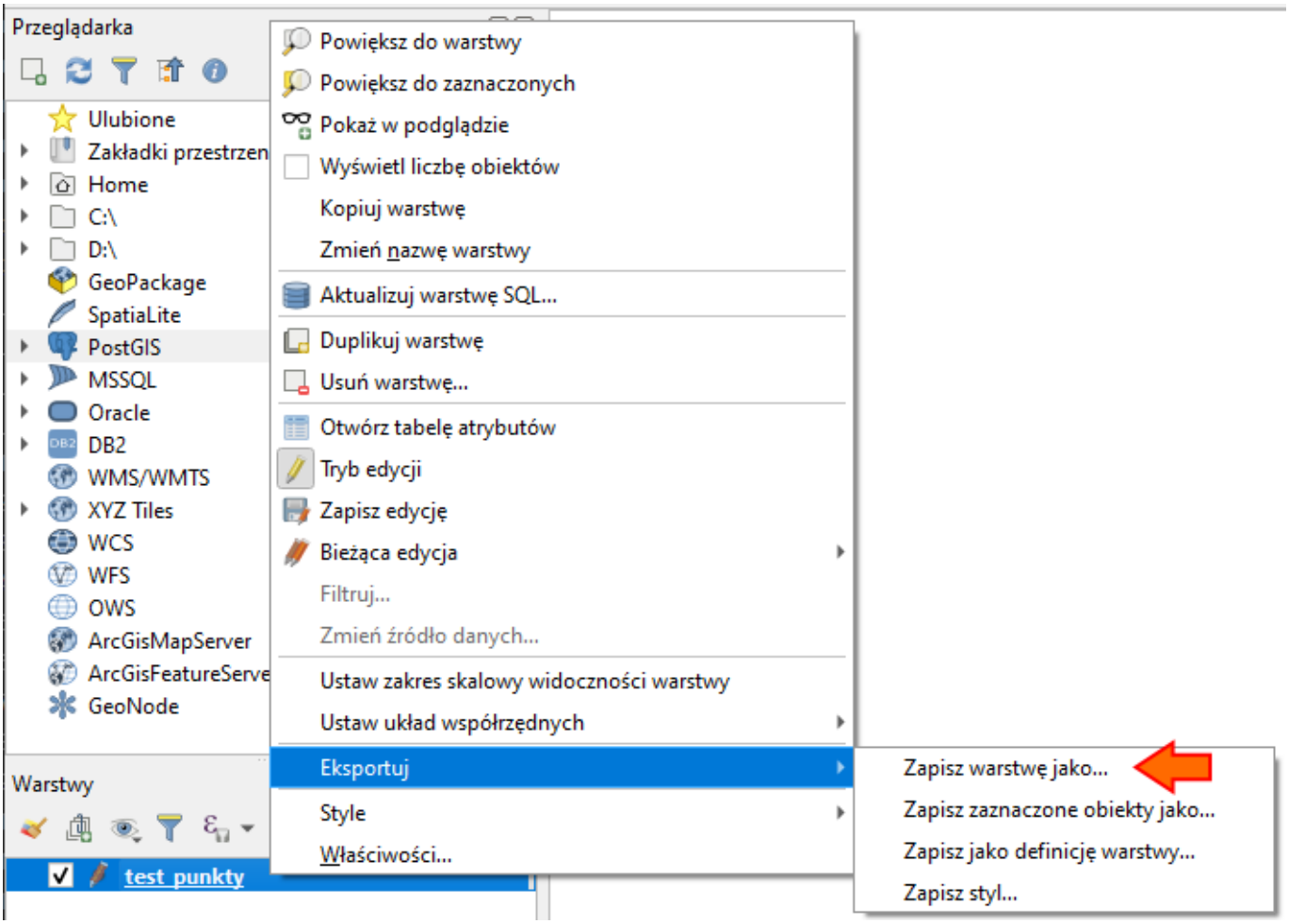


Pojawi się okno eksportu do pliku

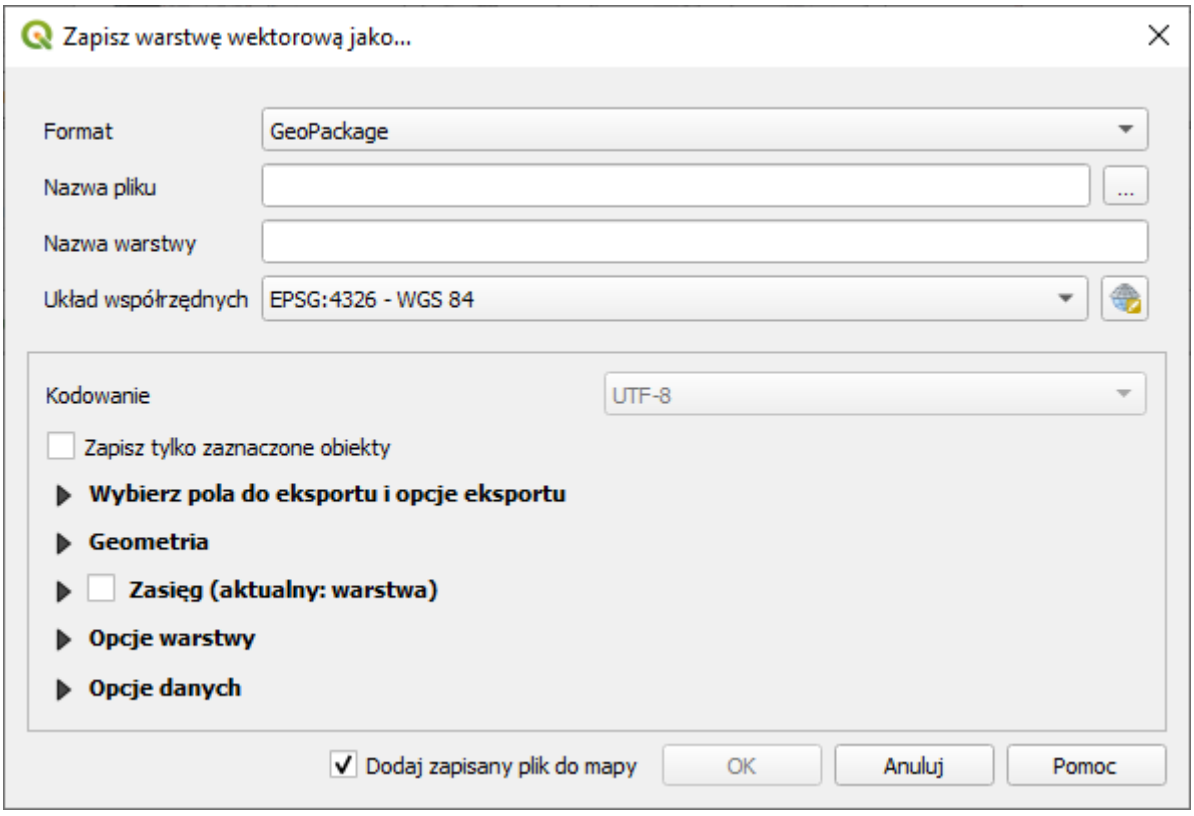


W oknie mamy możliwość wyboru kilkunastu formatów wektorowych - wybieramy `GeoJSON`, po czym podajemy ścieżkę zapisu pliku oraz jego nazwę i zatwierdzamy `OK`.

QGIS daje również możliwość zapisu dowolnej warstwy projektu do pliku, co może być pomocne jeśli potrzebujemy wyeksportować tylko zaznaczone obiekty z warstwy. W tym celu klikamy prawym przyciskiem myszy na warstwie do eksportu i z menu kontekstowego wybieramy `Eksportuj > Zapisz warstwę jako...`



W oknie eksportu jak wcześniej mamy możliwość wybrania formatu i nazwy pliku, ale również zapisania tylko zaznaczonych obiektów, wybrania zakresu danych do eksportu czy pól, które powinny zostać zapisane.



Blok 6 - tworzenie i operacje na danych wektorowych

Utworzenie schematu bazy (budynki, adresy, ulice, granice) i wypełnienie danymi

Na potrzeby dalszej części szkolenia utworzymy w bazie danych nowy schemat, w którym utworzymy kilka tabel i wypełnimy je danymi.

Wszystkie polecenia można wykonywać na bazie za pomocą dowolnego klienta bazy danych.

Do utworzenia modelu bazy oraz wypełnienia go danymi użyjemy poniższego skryptu:

```
-- usuwamy schemat osm w razie gdyby taki już istniał
drop schema if exists osm;

-- tworzymy schemat o nazwie osm
create schema osm;

-- tworzymy tabelę dla budynków
drop table if exists osm.budynki;
create table osm.budynki (
  id serial primary key,
  budynek text,
  geom geometry(polygon, 2180)
);

-- tworzymy tabelę dla punktów adresowych
drop table if exists osm.adresy;
create table osm.adresy (
  id serial primary key,
  kod_poczt text,
  miasto text,
  miejscowosc text,
  ulica text,
  numer text,
  geom geometry(point, 2180)
);

-- tworzymy tabelę dla ulic
drop table if exists osm.ulice;
create table osm.ulice (
  id serial primary key,
  typ_drogi text,
  nazwa text,
  geom geometry(linestring, 2180)
);

-- tworzymy tabelę dla granic administracyjnych
drop table if exists osm.granice;
create table osm.granice (
  id serial primary key,
  poziom text,
  nazwa text,
```

```
  teryt text,
  geom geometry(polygon, 2180)
);
```

Tabele będziemy wypełniali danymi zaimportowanymi za pomocą aplikacji osm2pgsql. Powyżej zdefiniowaliśmy, że nasze tabele będą przechowywały dane w układzie EPSG:2180 (Polska 1992), sprawdźmy więc w jakim układzie są dane źródłowe:

```
select distinct aa.srid from (
  select distinct st_srid(way) as srid from planet_osm_point
  union all
  select distinct st_srid(way) as srid from planet_osm_line
  union all
  select distinct st_srid(way) as srid from planet_osm_polygon
) as aa;
```

W odpowiedzi otrzymujemy:

```
 srid
------
 3857
(1 row)
```

Układy współrzędnych nie są zgodne, więc będziemy musieli zastosować funkcję ST_Transform żeby wykonać reprojekcję

Do oznaczania budynków w OSM używany jest klucz building.

Do wypełnienia tabel danymi użyjemy konstrukcji `insert ... select`.

```
insert into osm.budynki (budynek, geom)
  select
    tags -> 'building' as budynek,
    ST_Transform(way, 2180) as geom
  from
    planet_osm_polygon
  where
    tags ? 'building';
```

Kolumna klucza głównego (`id`) została celowo pominięta, ponieważ zdefiniowaliśmy ją jako `serial` więc sama wypełni się unikalnymi identyfikatorami.

Kolumna `tags` w danych osm jest typu `hstore`, dlatego do operacji na niej potrzebujemy specjalnym operatorów:

- konstrukcja `tags -> 'building'` zwróci wartość klucza building w kolumnie tags

- konstrukcja `tags ? 'building'` zwróci `true` jeśli w kolumnie tags istnieje klucz building.

W odpowiedzi otrzymujemy ilość wstawionych rekordów:

```
INSERT 0 720098
```

Analogicznie do powyższego wypełnimy tabelę z adresami. Adresy w danych OSM oznaczane są zestawem tagów z przestrzeni nazw `addr:`, czyli możemy założyć, że punkt adresowy musi posiadać numer domu oznaczany jako `addr:housenumber`. Dodatkowym utrudnieniem będzie to, że dane adresowe mogą być wypełniane zarówno na punktach adresowych (czyli tabela planet_osm_point)jak i na obrysach budynków, które są obszarami, więc znajdują się w tabeli planet_osm_polygon. Tabela z adresami definiuje geometrię jako punktową, więc dla obrysów budynków będziemy musieli użyć funkcji ST_Centroid która dla każdego obszaru zwróci jego centroid jako punkt.

```
insert into osm.adresy (kod_poczt, miasto, miejscowosc, ulica, numer, geom)
  select
    tags -> 'addr:postcode' as kod_poczt,
    tags -> 'addr:city' as miasto,
    tags -> 'addr:place' as miejscowosc,
    tags -> 'addr:street' as ulica,
    tags -> 'addr:housenumber' as numer,
    ST_Transform(way, 2180) as geom
  from
    planet_osm_point
  where
    tags ? 'addr:housenumber'
  union
  select
    tags -> 'addr:postcode' as kod_poczt,
    tags -> 'addr:city' as miasto,
    tags -> 'addr:place' as miejscowosc,
    tags -> 'addr:street' as ulica,
    tags -> 'addr:housenumber' as numer,
    ST_Centroid(ST_Transform(way, 2180)) as geom
  from
    planet_osm_polygon
  where
    tags ? 'addr:housenumber';
```

Kolejna tabela do wypełnienia to ulice, które oznaczane są za pomocą tagu `highway`.

```
insert into osm.ulice (typ_drogi, nazwa, geom)
  select
    tags -> 'highway' as typ_drogi,
    tags -> 'name' as nazwa,
    ST_Transform(way, 2180) as geom
  from
```

```
    planet_osm_line
  where
    tags ? 'highway';
```

Granice administracyjne definiuje tag `boundary` o wartości `administrative`, a ich typ oznaczamy tagiem `admin_level`.

```
insert into osm.granice (poziom, nazwa, teryt, geom)
  select
    tags -> 'admin_level' as poziom,
    tags -> 'name' as nazwa,
    tags -> 'teryt:terc' as teryt,
    ST_Transform(way, 2180) as geom
  from
    planet_osm_polygon
  where
    tags -> 'boundary' = 'administrative'
    and
    tags ? 'admin_level';
```

Konsolidacja tabeli gmin

Jak wiadomo gminy łączą się w powiaty, a te w województwa, dlatego aby w przyszłości ograniczyć ilość złączeń przestrzennych utworzymy tabelę gdzie każda gmina będzie miała przyporządkowany powiat (`admin_level=6`) oraz województwo (`admin_level=4`). Do utworzenia tabeli użyjemy konstrukcji `create table ... as select ...`. Nową tabelę uzyskamy przez odnalezienie dla centroidu każdej gminy zawierającego go powiatu oraz województwa. Unikalną numerację rekordów zapewni nam funkcja `row_number() over()`

```
create table osm.gminy as (
  select
    row_number() over() as id,
    woj.nazwa as woj,
    pow.nazwa as pow,
    gm.nazwa as gm,
    gm.teryt as teryt,
    gm.geom as geom
  from
    osm.granice gm
    join osm.granice pow
      on ST_Intersects(ST_Centroid(gm.geom), pow.geom) and pow.poziom = '6'
    join osm.granice woj
      on ST_Intersects(ST_Centroid(gm.geom), woj.geom) and woj.poziom = '4'
  where gm.poziom = '7';
```

```
SELECT 104

Query returned successfully in 287 msec.
```

Obliczenie ilości budynków w poszczególnych gminach

Aby obliczyć ilość budynków w poszczególnych gminach musimy użyć złączenia przestrzennego do przyporządkowania gminy do każdego budynku po czym zgrupować po gminach i obliczyć ilość rekordów dla każdej grupy. Wynik ograniczymy to 10 gmin z największą ilością budynków

```
select woj, pow, gm, count(*) as ilosc
from osm.gminy g
join osm.budynki b on st_intersects(g.geom, b.geom)
group by 1,2,3
order by 4 desc nulls last
limit 10;
```

Zapytanie zwróciło wynik, ale czas wykonania był dość długi

```
Successfully run. Total query runtime: 3 min 57 secs.
10 rows affected.
```

Dodajmy więc do tabel biorących udział w zapytaniu indeksy przestrzenne

```
create index gminy_geom_idx on osm.gminy using gist(geom);
create index budynki_geom_idx on osm.budynki using gist(geom);
```

Po dodaniu indeksów zapytanie wykonuje się znacznie szybciej

```
Successfully run. Total query runtime: 15 secs 770 msec.
10 rows affected.
```

Przypisanie gminy, powiatu i województwa do punktów adresowych

Aby przypisać jednostki administracyjne do adresów w pierwszej kolejności musimy dodać do tabeli odpowiednie kolumny

```
alter table osm.adresy
  add column woj text,
  add column pow text,
  add column gm text;
```

Następnie dodamy również indeks przestrzenny który pozwoli nam na dużo szybsze wykonanie kolejnych złączeń przestrzennych

```
create index adresy_geom_idx on osm.adresy using gist(geom);
```

Po czym aktualizujemy tabelę poleceniem

```
update
  osm.adresy
set
  woj = g.woj,
  pow = g.pow,
  gm = g.gm
from
  osm.gminy g
where
  st_intersects(adresy.geom, g.geom);
```

Polecenie powinno zwrócić

```
UPDATE 291012

Query returned successfully in 8 secs 680 msec.
```

Odnalezienie adresów dla poszczególnych budynków

Podobnie jak w poprzednim przypadku do tabeli z budynkami dodamy nową kolumnę, która będzie kluczem obcym do tabeli z adresami.

```
alter table osm.budynki
add column adres_id integer,
add constraint fk_adres_id foreign key (adres_id) references osm.adresy
(id);
```

Następnie możemy wypełnić adresy używając polecenia

```
update osm.budynki
set adres_id = a.id
from osm.adresy a
where st_intersects(budynki.geom, a.geom);
```

Na koniec sprawdzimy dla ilu budynków udało się odnaleźć adres

```
select count(*) from osm.budynki where adres_id is not null;
```

```
 count
--------
 272213
(1 row)
```

Utworzenie geometrii ze współrzędnych geograficznych (warstwa hydrantów)

Kolejne kroki szkolenia obejmują studium przypadku - otrzymaliśmy od lokalnych władz miasta Kielce warstwę z budynkami wyeksportowaną z programu AutoCAD do .shp oraz wykaz hydrantów w postaci listy ich typów i współrzędnych. Naszym zadaniem będzie analiza zabezpieczenia budynków siecią hydrantową.

Dla zadań wykonywanych w kolejnych krokach utworzymy nowy schemat

```
create schema kielce;
```

W plikach szkolenia możemy odnaleźć listę współrzędnych hydrantów kielce_hydranty.csv. Lista nie jest standardowym formatem danych przestrzennych ale dzięki możliwościom aplikacji `ogr2ogr` możemy zaimportować ją do bazy tworząc od razu geometrie punktowe ze współrzędnych, za pomocą komendy:

```
"c:\Program Files\QGIS 3.10\bin\ogr2ogr.exe" -f "PostgreSQL"
"PG:host=127.0.0.1 user=postgres dbname=gis password=gis"
kielce_hydranty.csv -oo X_POSSIBLE_NAMES=Lon* -oo Y_POSSIBLE_NAMES=Lat* -oo
KEEP_GEOM_COLUMNS=NO -nln kielce.kielce_hydranty
```

Zaimportowaną tabelę musimy trochę posprzątać, więc na początek zmienimy nazwę kolumny z danymi przestrzennymi

```
alter table kielce.kielce_hydranty rename column wkb_geometry to geom;
```

Analiza będzie wymagała dokonywania pomiarów odległości, więc zmienimy układ współrzędnych danych na `Polska 1992`:

```
update kielce.kielce_hydranty set geom =
st_transform(ST_SetSRID(geom,4326), 2180);
```

Na koniec usuniemy kolumnę `id` po czym istniejącą kolumnę `ogc_fid` przemianujemy na `id`

```
alter table kielce.kielce_hydranty drop column id;
alter table kielce.kielce_hydranty rename column ogc_fid to id;
```

W ten sposób tabela jest zgodna ze strukturami tabel w innych schematach - uniknie to późniejszych błędów w zapytaniach wynikających ze stosowania różnych nazw dla kolumn o tym samym zastosowaniu.

Łączenie linii w obszary (warstwa budynków)

Kolejnym przekazanym zasobem jest warstwa budynków w formacie ESRI Shapefile - kielce_budynki.shp. Warstwę zaimportujemy również za pomocą `ogr2ogr`.

```
"c:\Program Files\QGIS 3.10\bin\ogr2ogr.exe" -f "PostgreSQL"
"PG:host=127.0.0.1 user=postgres dbname=gis password=gis"
kielce_budynki.shp -nln kielce.kielce_budynki
```

Niestety po sprawdzeniu jej zawartości okazuje się że nie zawiera ona obszarów budynków jedynie linie ich obrysów - dodatkowo podzielone w ten sposób że każda linia ma dokładnie dwa wierzchołki - żeby uzyskać obszary budynków musielibyśmy wszystkie te linie połączyć w obrysy po czym przekształcić w obszary - w tym przypadku z pomocą przyjdzie nam PostGIS - uruchamiamy poniższe polecenie:

```
create table kielce_budynki_fin as
select row_number() over() as id, geom from (
  select (st_dump(st_polygonize(geom))).geom from kielce_budynki) as aa;
```

W ten sposób PostGIS utworzył dla nas nową tabelę z budynkami - tym razem już w formie obszarów. Dalej usuwamy tabelę z liniami:

```
drop table if exists kielce.kielce_budynki;
```

Nowej tabeli zmieniamy nazwę:

```
alter table kielce.kielce_budynki_fin rename to kielce_budynki;
```

Uzgadniamy układ współrzędnych z tabelą z hydrantami:

```
update kielce.kielce_budynki set geom = st_transform(geom, 2180);
```

Jak wcześniej zmieniamy nazwę kolumny zawierającej identyfikatory:

```
alter table kielce.kielce_budynki rename column ogc_fid to id;
```

Dodatkowo utworzymy mechanizm wypełniający dla nas automatycznie identyfikatory. Tworzymy nowy generator:

```
create sequence kielce.kielce_budynki_id_seq;
```

Ustawiamy aktualną wartość generatora na maksymalne `id` z tabeli + 1:

```
SELECT setval('kielce.kielce_budynki_id_seq', COALESCE((SELECT MAX(id)+1
FROM kielce.kielce_budynki), 1), false);
```

Po czym ustawiamy aby wartość pola `id` była automatycznie pobierana z generatora:

```
alter table kielce.kielce_budynki alter column id set default
nextval('kielce.kielce_budynki_id_seq');
```

I zakładamy na tabeli klucz unikalny - potrzebny w przypadku edycji jej przez QGIS.

```
alter table kielce.kielce_budynki add constraint pk_kielce_kielce_budynki
primary key (id);
```

Mamy kolejną tabelę konieczną do naszej analizy.

Utworzenie obszaru zabezpieczonego siecią hydrantową

Budynek uznajemy za zabezpieczony jeśli hydrant znajduje się nie dalej niż 150 metrów od niego, dlatego teraz utworzymy dla każdego hydrantu nowe pole, w którym utworzymy obszar, który chroni używając funkcji ST_Buffer

```
alter table kielce.kielce_hydranty add column buffer geometry;
update kielce.kielce_hydranty set buffer=st_buffer(geom, 150);
```

Odnalezienie budynków zabezpieczonych siecią hydrantową

W kolejnym kroku dla każdego budynku odnajdziemy identyfikator hydrantu, który go chroni. W tym celu dodajemy nową kolumnę `hydrant_id`, która będzie kluczem obcym do tabeli z hydrantami.

Aby utworzyć klucz obcy tabela hydrantów musi mieć klucz główny, który musimy utworzyć:

```
ALTER TABLE kielce.kielce_hydranty
    ADD CONSTRAINT kielce_hydranty_pkey PRIMARY KEY (id);
```

Następnie do tabeli z budynkami dodajemy kolumnę z kluczem obcym

```
alter table kielce.kielce_budynki
  add column hydrant_id integer,
  add constraint fk_hydrant_id foreign key (hydrant_id) references
kielce.kielce_hydranty (id);
```

Aby zapytanie działało wydajnie utworzymy kilka indeksów przestrzennych

```
create index kielce_hydranty_geom_idx on kielce.kielce_hydranty using
gist(geom);
create index kielce_hydranty_buffer_idx on kielce.kielce_hydranty using
gist(buffer);
create index kielce_budynki_geom_idx on kielce.kielce_budynki using
gist(geom);
```

Teraz możemy już uruchomić właściwe zapytanie - w tym przypadku użyjemy instrukcji `UPDATE` z niestandardową dla niej klauzulą `FROM` oraz warunkiem opartym na funkcji `st_intersects` sprawdzającej czy dwie geometrie przecinają się.

```
update
  kielce.kielce_budynki
set
  hydrant_id = h.id
from
  kielce.kielce_hydranty h
where
  st_intersects(h.buffer, kielce_budynki.geom)
```

Teraz możemy już sprawdzić zgrubny wynik naszej analizy

```
select zabezpieczony, count(*) from (
  select
    case
      when hydrant_id is not null then 'tak'::text
      else 'nie'::text
    end as zabezpieczony
  from
    kielce.kielce_budynki
```

```
) as aa
group by 1;
```

Możemy też użyć QGIS, aby wyświetlić na mapie zabezpieczone i niezabezpieczone budynki.

Blok 7 - Operacje na danych rastrowych

PostGIS od wersji 3.0 ma wydzielone rozszerzenie obsługujące dane rastrowe - w poprzedniej wersji włączenie rozszerzenia postgis włączało również obsługę danych rastrowych - od wersji 3.0 trzeba ją włączyć osobno używając polecenia:

```
create extension postgis_raster;
```

Import danych rastrowych za pomocą raster2pgsql

Importy danych rastrowych do bazy wykonywane są za pomocą narzędzia raster2pgsql. Krótką pomoc na temat programu uzyskamy uruchamiając go bez żadnych parametrów.

```
"c:\Program Files\PostgreSQL\12\bin\raster2pgsql.exe"
```

```
RELEASE: 3.0.1 GDAL_VERSION=24 (3.0.1)
USAGE: raster2pgsql [<options>] <raster>[ <raster>[ ...]] [[<schema>.]
<table>]
  Multiple rasters can also be specified using wildcards (*,?).

OPTIONS:
  -s <srid> Set the SRID field. Defaults to 0. If SRID not
     provided or is 0, raster's metadata will be checked to
     determine an appropriate SRID.
  -b <band> Index (1-based) of band to extract from raster. For more
      than one band index, separate with comma (,). Ranges can be
      defined by separating with dash (-). If unspecified, all bands
      of raster will be extracted.
  -t <tile size> Cut raster into tiles to be inserted one per
      table row. <tile size> is expressed as WIDTHxHEIGHT.
      <tile size> can also be "auto" to allow the loader to compute
      an appropriate tile size using the first raster and applied to
      all rasters.
  -P Pad right-most and bottom-most tiles to guarantee that all tiles
     have the same width and height.
  -R  Register the raster as an out-of-db (filesystem) raster. Provided
      raster should have absolute path to the file
 (-d|a|c|p) These are mutually exclusive options:
     -d  Drops the table, then recreates it and populates
         it with current raster data.
     -a  Appends raster into current table, must be
         exactly the same table schema.
     -c  Creates a new table and populates it, this is the
         default if you do not specify any options.
     -p  Prepare mode, only creates the table.
  -f <column> Specify the name of the raster column
  -F  Add a column with the filename of the raster.
  -n <column> Specify the name of the filename column. Implies -F.
  -l <overview factor> Create overview of the raster. For more than
      one factor, separate with comma(,). Overview table name follows
      the pattern o_<overview factor>_<table>. Created overview is
      stored in the database and is not affected by -R.
  -q  Wrap PostgreSQL identifiers in quotes.
  -I  Create a GIST spatial index on the raster column. The ANALYZE
      command will automatically be issued for the created index.
  -M  Run VACUUM ANALYZE on the table of the raster column. Most
      useful when appending raster to existing table with -a.
  -C  Set the standard set of constraints on the raster
      column after the rasters are loaded. Some constraints may fail
      if one or more rasters violate the constraint.
```

```
  -x  Disable setting the max extent constraint. Only applied if
      -C flag is also used.
  -r  Set the constraints (spatially unique and coverage tile) for
      regular blocking. Only applied if -C flag is also used.
  -T <tablespace> Specify the tablespace for the new table.
      Note that indices (including the primary key) will still use
      the default tablespace unless the -X flag is also used.
  -X <tablespace> Specify the tablespace for the table's new index.
      This applies to the primary key and the spatial index if
      the -I flag is used.
  -N <nodata> NODATA value to use on bands without a NODATA value.
  -k  Skip NODATA value checks for each raster band.
  -E <endian> Control endianness of generated binary output of
      raster. Use 0 for XDR and 1 for NDR (default). Only NDR
      is supported at this time.
  -V <version> Specify version of output WKB format. Default
      is 0. Only 0 is supported at this time.
  -e  Execute each statement individually, do not use a transaction.
  -Y  Use COPY statements instead of INSERT statements.
  -G  Print the supported GDAL raster formats.
  -?  Display this help screen.
```

W zależności od zastosowania i planowanych operacji na rastrach możemy zaimportować go do bazy danych w całości (jeśli planujemy jego przekształcenia) oraz w podziale na mniejsze kafelki (jeśli planujemy złączenia przestrzenne z jego udziałem). W ramach szkolenia będziemy wykonywali obie te operacje dlatego zaimportujemy nasz raster dwukrotnie.

Do celu szkolenia użyjemy rastra z wysokością terenu powstałego przez reprojekcję i przycięcie zasobu EU_DEM utworzonego w ramach projektu Copernicus. Plik dostępny jest w plikach szkolenia pod nazwą EUDEM_swietokrzyskie_2180.tif

Aby zaimportować w całości uruchamiamy komendę:

```
"c:\Program Files\PostgreSQL\12\bin\raster2pgsql.exe" -c -s 2180 -f rast -I
EUDEM_swietokrzyskie_2180.tif kielce.dem_solid >
EUDEM_swietokrzyskie_2180_solid.sql
```

```
Processing 1/1: EUDEM_swietokrzyskie_2180.tif
```

Wynikowy plik sql importujemy za pomocą aplikacji `psql`:

```
"c:\Program Files\PostgreSQL\12\bin\psql.exe" -U postgres -f
EUDEM_swietokrzyskie_2180_solid.sql gis
```

```
BEGIN
CREATE TABLE
INSERT 0 1
CREATE INDEX
ANALYZE
COMMIT
```

Aby utworzyć tabelę z rastrem podzielonym na fragmenty używamy polecenia,które od powyższego polecenia różni się jedynie parametrem `-t 100x100` wymuszającym podzielenie rastra na kafle o wymiarach 100x100px.

```
"c:\Program Files\PostgreSQL\12\bin\raster2pgsql.exe" -c -s 2180 -t 100x100
-f rast -I EUDEM_swietokrzyskie_2180.tif kielce.dem >
EUDEM_swietokrzyskie_2180.sql
```

```
Processing 1/1: EUDEM_swietokrzyskie_2180.tif
```

Wynikowy skrypt jak wcześniej importujemy za pomocą `psql`.

```
"c:\Program Files\PostgreSQL\12\bin\psql.exe" -U postgres -f
EUDEM_swietokrzyskie_2180.sql gis
```

```
BEGIN
CREATE TABLE
INSERT 0 1
...
INSERT 0 1
CREATE INDEX
ANALYZE
COMMIT
```

Zaimportowane dane możemy obejrzeć importując je do QGIS.

Reprojekcja rastra

Pierwszą operacją na rastrze będzie jego reprojekcja, którą wykonamy za pomocą funkcji ST_Transform

```
drop table if exists kielce.dem_4326;
create table kielce.dem_4326 as
select rid, ST_Transform(rast,4326, 'Lanczos') as rast
from kielce.dem_solid;
```

```
UWAGA:  tabela "dem_4326" nie istnieje, pominięto
SELECT 1

Query returned successfully in 8 secs 721 msec.
```

Powyższy skrypt utworzy w bazie nową tabelę o nazwie dem_4326 znajdującą się w schemacie `kielce`. Wynikowy raster możemy obejrzeć w aplikacji QGIS.

Odnalezienie wysokości dla każdego budynku w bazie danych

Obsługa rastrów w bazie danych PostGIS poza podstawowymi przekształceniami pozwala również na złączenia przestrzenne z danymi wektorowymi oraz na wyciąganie z rastrów wartości poszczególnych pikseli. W bieżącym kroku odnajdziemy wartość wysokości dla każdego budynku w bazie danych. W tym celu użyjemy funkcji ST_VALUE.

Powyższa funkcja zwraca wartość z rastra dla wskazanych punktów, dlatego, żeby nie tworzyć punktów w geometrii budynków 'w locie' utworzymy sobie nową kolumnę i wypełnimy ją centroidami budynków. Na powstałej kolumnie założymy również indeks.

```
alter table kielce.kielce_budynki add column centroid geometry;
update kielce.kielce_budynki set centroid = ST_Centroid(geom);
create index kielce_budynki_cent_idx on kielce.kielce_budynki using
gist(centroid);
```

Dodajemy również kolumnę dla wysokości i wypełniamy ją danymi

```
alter table kielce.kielce_budynki add column elevation numeric;

update kielce.kielce_budynki set elevation = ST_Value(r.rast, centroid,
true)
from kielce.dem r
where ST_Intersects(centroid, rast);
```

Na koniec możemy sprawdzić czy dla wszystkich budynków udało się uzyskać wysokość.

```
select count(*) from kielce.kielce_budynki where elevation is null;
```

Wygenerowanie cieniowania terenu

```
drop table if exists kielce.dem_hs;
create table kielce.dem_hs as
```

```
select rid, ST_HillShade(rast, 1, '32BF', 315, 45, 255, 1, false) as rast
from kielce.dem_solid;
```

```
UWAGA:  tabela "dem_hs" nie istnieje, pominięto
SELECT 3172

Query returned successfully in 6 min 9 secs.
```

Wygenerowanie pochyłości terenu

PostGIS zawiera również funkcje dedykowane do analiz numerycznych modeli terenu. W kolejnym kroku z zaimportowanego rastra z wysokością wygenerujemy raster określający pochyłość terenu za pomocą funkcjiST_Slope

```
drop table if exists kielce.dem_sl;
create table kielce.dem_sl as
select rid, ST_Slope(rast, 1, '32BF', 'DEGREES', 111120, false) as rast
from kielce.dem_solid;
```

```
UWAGA:  tabela "dem_sl" nie istnieje, pominięto
SELECT 1

Query returned successfully in 5 min 31 secs.
```

Wygenerowany raster będziemy chcieli użyć do wypełnienia danych w tabeli z budynkami, dlatego wygenerujemy kolejny zasób powstały przez podzielenie za pomocą funkcji ST_Tile

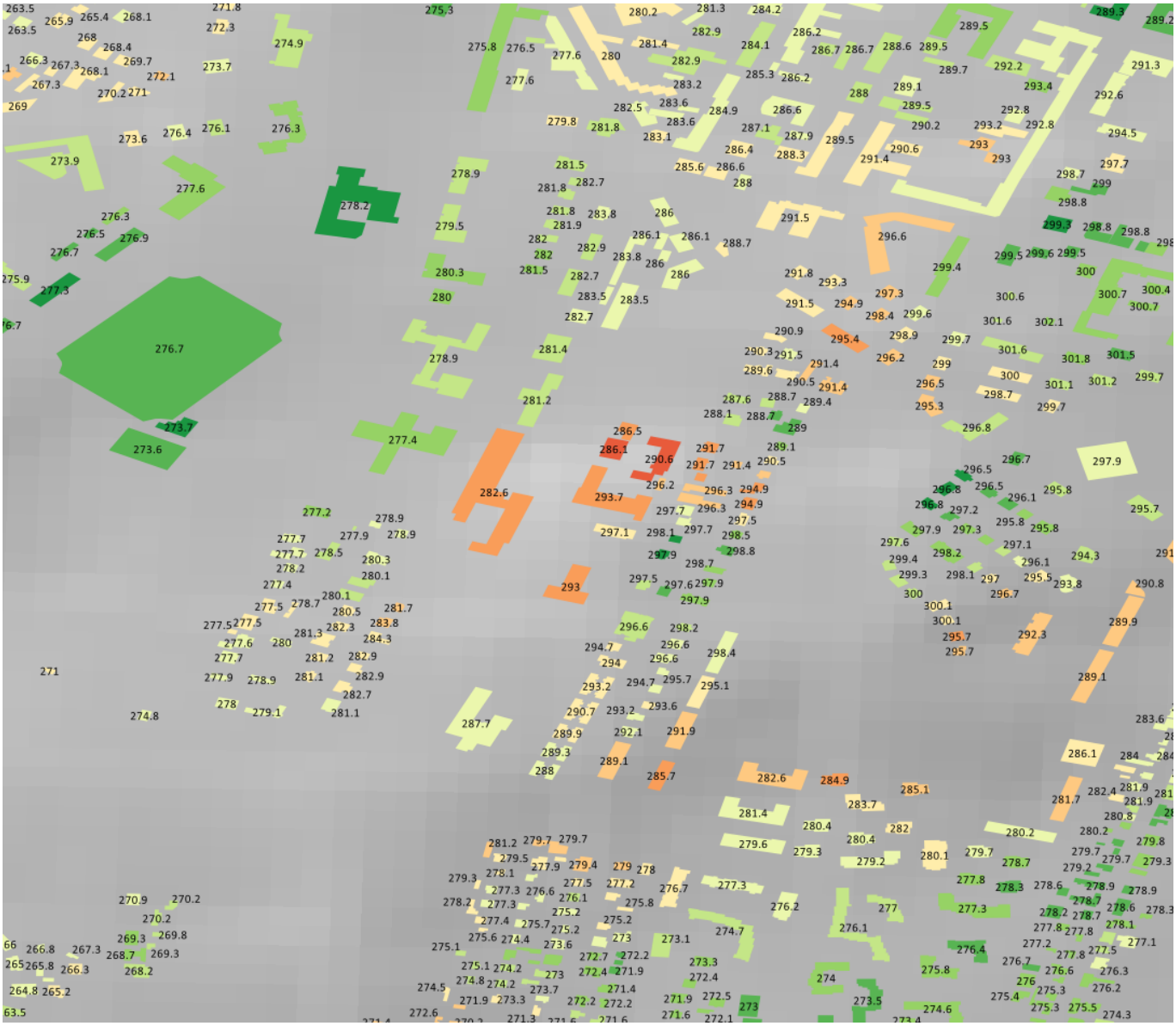
```
drop table if exists kielce.dem_sl_tile;
create table kielce.dem_sl_tile as select ST_Tile(rast, 1, 100, 100, false,
null) as rast from kielce.dem_sl;
```

Do tabeli z budynkami dodajemy nową kolumnę w wstawiamy w nią wartości pobrane z rastra jak w poprzednim kroku

```
alter table kielce.kielce_budynki add column slope numeric;

update kielce.kielce_budynki set slope = ST_Value(r.rast, centroid, true)
from kielce.dem_sl_tile r
where ST_Intersects(centroid, rast);
```

Dzięki wykonanym operacjom możemy w QGIS wygenerować mapę prezentującą wysokość budynków oraz pochyłość gruntu na którym zostały one posadowione.



Blok 8 - Procedury składowane

Wprowadzenie do procedur

Procedury składowane mogą być pisane w różnych językach, głównie PL/pgSQL oraz SQL, ale również Perl, Python, PHP, Lua, C, C++. Oprócz tradycyjnego przetwarzania danych, umożliwiają także definiowanie:

- ograniczeń sprawdzających (CHECK),
- wyzwalaczy,
- funkcji agregujących i okienkowych (w rodzaju MIN, MAX, AVG),
- operatorów, w tym
    - operatorów konwersji typów ((CAST x AS typ)),
    - dziedzin (ang. domain)

Zastosowanie procedur składowanych są różnorakie, mogą w najprostszych przypadkach ułatwiać administrację bazą danych, zarządzaniem użytkownikami itp., mogą również zostać użyte jako dodatkowa

warstwa dostępu do danych w aplikacji użytkowej.

PostgreSQL obsługuje przeciążanie nazw funkcji - w bazie może istnieć wiele procedur o tej samej nazwie pod warunkiem, że mają różne sygnatury. Sygnaturą jest nazwa funkcji oraz typy argumentów wejściowych, ale już typy wynikowe nie grają roli. Nie mogą więc istnieć dwie funkcje o tej samej nazwie i argumentach, ale różnych typach wynikowych.

Należy pamiętać, że w przypadku utworzenia funkcji w języku PL/pgSQL lub SQL w bazie danych pamiętany jest również jej pełny kod źródłowy, łącznie z komentarzami.

W PostgreSQL nową procedurę do bazy wprowadza polecenie o składni:

```
CREATE [ OR REPLACE ] FUNCTION nazwa(lista argumentów) RETURNS typ wyniku
AS
      'treść procedury'
LANGUAGE nazwa języka
[ jak planer ma traktować procedurę? ]
[ reguły dostępu ]
```

- **lista argumentów** - `[rodzaj[IN/OUT/INOUT]] [nazwa] typ [ { DEFAULT | = } wartość domyślna ]`
- **typ wyniku** - procedura może zwracać między innymi:
    - void (wartość NULL)
    - liczbę
    - jeden rekord z tabeli
    - jeden rekord o podanej strukturze
    - setof (ciągi liczb i znaków)
    - ciąg rekordów z tabeli
    - ciąg rekordów o określonej strukturze
- **treść procedury**
- **rodzaj języka**: plpgsql lub SQL
- **jak planer ma traktować procedurę?**
    - rodzaj funkcji
        - IMMUTABLE - procedura nie zmienia bazy danych i wynik przez nią zwracany nie zależy od zawartości bazy. Np. funkcja wykonująca operacje arytmetyczne ma taką cechę.
        - STABLE - procedura nie zmienia bazy danych i zawsze zwraca ten sam wynik pod warunkiem, że dane się nie zmieniły. Np. funkcja, która wyłącznie wyszukuje pracowników wg numeru PESEL zwróci zawsze ten sam zestaw rekordów jeśli pomiędzy jej wywołaniami nie zmieniono tabeli pracowników.
        - VOLATILE - procedura ma efekty uboczne, tj. zmienia bazę danych, np. używane są wprost zapytania w rodzaju UPDATE, DELETE, bądź wywoływane inne procedury tego typu. Domyślnie, pesymistycznie, przyjmowany jest taki rodzaj.
    - czy wywoływać, gdy jeden z argumentów jest NULL?
        - CALLED ON NULL INPUT - wywoływana nawet jeśli parametr wejściowy ma wartość null
        - RETURNS NULL ON NULL INPUT lub STRICT - gdy przynajmniej jeden argument jest pusty, automatycznie zwracany jest NULL, bez wywoływania funkcji.
    - szacowany czas wykonania

        - COST czas - Wartość COST czas podawana przy tworzeniu procedury musi być liczbą rzeczywistą nieujemną.Jeśli procedurę charakteryzuje duży czas wykonania, optymalizator zapytań może przekształcić tak warunki logiczne - ale tylko w klauzuli WHERE - żeby procedura nie była wywoływana za każdym razem. Wykorzystuje się tutaj leniwe wartościowanie wyrażeń logicznych (ang. short circuit evaluation) zawierających operator AND.
    - szacowana liczba danych wyjściowych
        - ROWS liczba wierszy - musi być liczbą całkowitą.Od liczby wierszy zależy plan zapytania, np. zostanie wykorzystany indeks (dla dużej liczby wierszy) lub też sekwencyjne skanowanie (dla małej).
- **reguły dostępu**
    - SECURITY INVOKER [domyślnie] - procedura wywoływana jest z takimi uprawnieniami, jak aktualnie zalogowany użytkownik;
    - SECURITY DEFINER - procedura wywoływana jest z takimi uprawnieniami, jak użytkownik, który ją zdefiniował. Dzięki tej opcji można częściowo lub całkowicie odizolować zwykłych użytkowników od tabel, i umożliwić modyfikację lub dostęp wyłącznie przez dedykowane funkcje. W szczególności mogą to być np. jakieś krytyczne ustawienia aplikacji, dane finansowe itp.
    - SEARCH_PATH - W chwili tworzenia funkcji można określić ścieżkę wyszukiwania ograniczając tym samym dostęp do innych schematów.

Utworzenie prostej funkcji geokodowania

Geokodowanie to proces odnajdywania na mapie (zwracania współrzędnych) obiektów o wskazanych właściwościach - w tym kroku zajmiemy się geokodowaniem adresów, czyli konwersją adresów w formie tekstowej na dane przestrzenne.

Funkcja, którą utworzymy będzie przyjmowała 4 parametry wejściowe - `miasto`, `miejscowość`, `ulicę` i `numer`. Używając tych parametrów pobierze odpowiednie dane z tabeli z adresami i zwróci dla nich geometrię punktową w odwzorowaniu EPSG:4326.

```
drop function if exists osm.geocode(text, text, text, text);

create or replace
  function osm.geocode (miasto text, miejscowosc text, ulica text, numer
text)
  returns geometry(point, 4326)
  as
  $$
    select
      ST_Transform(a.geom, 4326)
    from
      osm.adresy a
    where
      a.miasto = miasto
      and a.miejscowosc = miejscowosc
      and a.ulica = ulica
      and a.numer = numer
    limit 1;
```

```
  $$
  language sql
  stable
  cost 0.02;
```

Działanie funkcji możemy sprawdzić uruchamiając zapytanie

```
select * from osm.geocode('Maluszyn', null, 'Wolności', '12');
```

Utworzenie prostej funkcji rev-geokodowania

Rev-Geokodowanie (odwrotne geokodowanie) to proces odwrotny do geokodowania, czyli dla zadanych współrzędnych zwraca z bazy parametry obiektu - w naszym przypadku będzie się starał znaleźć adres najbliższy wskazanych współrzędnych. W ciele funkcji użyjemy dwóch funkcji - najpierw w celu poprawienia wydajności ograniczymy wyszukiwanie do 200 metrów od wskazanego punktu używając funkcji ST_Dwithin, po czym otrzymany podzbiór posortujemy po odległości od wskazanego punktu używając funkcji ST_Distance i zwrócimy pierwszy rekord - najbliższy wskazanych współrzędnych.

```
drop function if exists osm.revgeocode(double precision, double precision);

create or replace
  function osm.revgeocode (lon double precision, lat double precision)
  returns table (miasto text, miejscowosc text, ulica text, numer text)
  as
  $$
    select
      miasto,
      miejscowosc,
      ulica,
      numer
    from
      osm.adresy a
    where
      ST_Dwithin(ST_Transform(ST_SetSRID(ST_MakePoint(lon,
lat),4326),2180), a.geom, 200)
    order by
      ST_Distance(ST_Transform(ST_SetSRID(ST_MakePoint(lon,
lat),4326),2180), a.geom)
      asc
    limit 1;
  $$
  language sql stable cost 0.02;
```

Działanie funkcji możemy sprawdzić uruchamiając zapytanie

```
select * from osm.revgeocode(19.7968403, 50.9129479001876);
```

Powinno zwrócić adres `Maluszyn, Wolności 12`

Geokodowanie adresów dla testowej listy współrzędnych

Utworzone funkcje możemy użyć do uzupełniania danych w bazie - w tym celu utworzymy testowy zestaw danych. Poniższe polecenie dla każdego miasta w bazie wyciągnie jeden unikalny adres

```
drop table if exists osm.geocode_test;

create table osm.geocode_test as
  select distinct on (miasto)
    miasto,
    miejscowosc,
    ulica,
    numer
  from
    osm.adresy;
```

Do powstałej tabeli testowej dodajemy nową kolumnę

```
alter table osm.geocode_test add column geom geometry(point, 4326);
```

Po czym uruchamiamy polecenie które wypełni tę kolumnę danymi za pomocą utworzonej wcześniej procedury.

```
update osm.geocode_test
set geom = osm.geocode(miasto, miejscowosc, ulica, numer);
```

Odwrotne geokodowanie z użyciem bloku anonimowego

Analizy przestrzenne są czasochłonne, dlatego często potrzebujemy metody, aby wykonywać je krok po kroku nie tracąc wyniku wykonanych już kroków. Jednym z rozwiązań tego problemu jest użycia anonimowego bloku kodu w języku plpgsql. Test procedury odwrotnego geokodowania przeprowadzimy właśnie z użyciem takiego bloku kodu. N a początek - jak wcześniej - muśimy utworzyć testowy zestaw danych - będzie to tabela ze współrzędnymi jednego adresu z każdego miasta z tabeli adresów:

```
drop table if exists osm.revgeocode_test;

create table osm.revgeocode_test as
  select
    row_number() over() as id, *
  from (
    select distinct on (miasto)
```

```
      st_X(st_transform(geom,4326)) as lon,
      st_Y(st_transform(geom,4326)) as lat
    from
      osm.adresy
    ) as aa;
```

Do powstałej tabeli dodajemy kolumny, które będziemy wypełniali

```
alter table osm.revgeocode_test add column miasto text;
alter table osm.revgeocode_test add column miejscowosc text;
alter table osm.revgeocode_test add column ulica text;
alter table osm.revgeocode_test add column numer text;
```

Po czym tworzymy blok kodu - jego działanie opisane jest za pomocą komentarzy w kodzie.

```
do
$$
  declare --sekcja pozwalające deklarować zmienne
    rev record; --zmienna typu `record` w której przechowamy to, co zwróci
funkcja rev-geokodowania
    test record; --zmienna typu `record` w której przechowamy rekord z
tabeli testowej
    row_cnt integer; --- zmienna typu integer w której przechowamy ilość
rekordów w przetwarzanej tabeli
  begin --tutaj zaczyna się właściwy kod
    select count(*) from osm.revgeocode_test where miejscowosc is null into
row_cnt; -- wstawiamy do zmiennej row_cnt ilość rekordów zliczoną z tabeli
testowej
    for test in select * from osm.revgeocode_test where miejscowosc is null
    loop -- pętla for która pobiera z tabeli testowej puste rekordy i
wstawia do zmiennej test
      raise notice 'rekord: %/%',test.id,row_cnt; -- zwracamy na konsolę
komunikat który wiersz przetwarzamy
      select * from osm.revgeocode(test.lon, test.lat) into rev; --tutaj
uruchamiamy funkcję i jej wynik przekazujemy do zmiennej rev
      --kolejne polecenie dodaje do tabeli testowej dane pobrane z
procedury
      update osm.revgeocode_test set miasto = rev.miasto, miejscowosc =
rev.miejscowosc, ulica = rev.ulica, numer = rev.numer
      where id = test.id;
      commit; --tutaj clue całego kodu - 'zapisujemy' dane w bazie
    end loop; -- kończymy pętlę i wracamy do początku
  end;
$$
language plpgsql;
```

Blok 9 - Wyzwalacze

Wprowadzenie - metodyka tworzenia wyzwalaczy

Wyzwalacze mogą być używane do wielu różnych celów, np.

- bardziej zaawansowana kontrola danych przed wstawieniem/aktualizacją,
- kontrola dostępu, np. wyłącznie możliwości kasowanie wybranych danych,
- automatyczne logowanie akcji,
- automatyczne tworzenie wpisów historycznych,
- realizacja widoków zmaterializowanych.

Składnia tworzenia wyzwalacza:

```
CREATE TRIGGER nazwa_wyzwalacza { BEFORE | AFTER } { zdarzenie [ OR ... ] }
    ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( warunek ) ]
    EXECUTE PROCEDURE PROCEDURE nazwa_procedury ( argumenty )
```

Zdarzenie to INSERT, UPDATE, DELETE, TRUNCATE.

Procedura może zostać wywołana na dwóch poziomach: albo dla każdego przetwarzanego wiersza (`ROW`) albo dla pojedynczej instrukcji SQL (`STATEMENT`). Wywołanie może nastąpić przed (`BEFORE`) lub po (`AFTER`) operacji.

Opcjonalny warunek `WHEN`, wprowadzony w wersji 9.0, pozwala odpalać procedurę warunkowo; można uzyskać w ten sposób wyzwalacze działające na poziomie kolumn.

Informacja - Pojedyncza procedura składowana może być używana w wielu wyzwalaczach.

W PostgreSQL wyzwalacze wywołują procedury składowane, które:

- nie mogą przyjmować argumentów
- zwracają psuedotyp TRIGGER.

Argumenty dla procedury są określane podczas definiowania wyzwalacza i dostępne przez zmienne TG_NARGS i TG_ARGV. Procedura musi zwrócić albo rekord o tej samej strukturze co zmienna NEW, albo NULL, albo zgłosić wyjątek. Procedur wyzwalaczy nie można wywoływać wprost.

Procedura ma dostęp do wielu danych dotyczących wyzwalacza i tabeli dla której została wywołana, więc można jedną funkcją obsłużyć np. logowanie zmian w wielu tabelach.

Procedura wyzwalacza ma w chwili wywołania dostęp do następujących zmiennych:

- **NEW** - zmienna typu `RECORD` zawiera wiersz, który ma zostać wstawiony (`INSERT`) lub zaktualizowany (`UPDATE`). Ma wartość NULL dla operacji DELETE/TRUNCATE i dla wszystkich operacji wykonywanych na poziomie instrukcji SQL.
- **OLD** - zmienna typu `RECORD` zawiera wiersz który ma zostać zastąpiony (`UPDATE`) lub skasowany (`DELETE`). Ma wartość `NULL` dla operacji `INSERT`/`TRUNCATE` i dla wszystkich operacji wykonywanych na poziomie instrukcji SQL.
- **TG_NAME** - nazwa_wyzwalacza (napis)

- **TG_WHEN** - w zależności od chwili odpalenia wyzwalacza, napis `BEFORE` - przed lub `AFTER` - po zdarzeniu
- **TG_LEVEL** - w zależności od poziomu działania wyzwalacza, napis `ROW` - przetwarzanie wiersza lub `STATEMENT` - wykonywanie instrukcji
- **TG_OP** - napis określający dla jakiej operacji został wywołany: `INSERT`, `UPDATE`, `DELETE` lub `TRUNCATE.
- **TG_RELID** - oid tabeli `TG_TABLE_NAME`
- **TG_TABLE_NAME** - nazwa tabeli dla której wyzwalacz został wywołany
- **TG_TABLE_SCHEMA** - nazwa schematu, w którym umieszczona jest tabela
- **TG_TABLE_NAME** - nazwa tabeli
- **TG_NARGS** - liczba argumentów, które zostały zdefiniowane dla wyzwalacza
- **TG_ARGV[]** - lista argumentów w formie napisów, które zostały zdefiniowane dla wyzwalacza; sięgnięcie poza zakres tablicy nie powoduje błędu, jedynie zwrócenie NULL.

Kolejność wykonywania wyzwalaczy PostgreSQL nie trzyma się tutaj standardu SQL, który przewiduje, że wyzwalacze są uruchamiane zgodnie z kolejnością tworzenia. W PostgreSQL decyduje porządek alfabetyczny nazw wyzwalaczy, co ułatwia kontrolę nad kolejnością.

Utworzenie wyzwalacza uzupełniającego powierzchnię dodanego budynku

```
-- ALTER TABLE kielce.kielce_budynki DROP COLUMN powierzchnia;

ALTER TABLE kielce.kielce_budynki
    ADD COLUMN powierzchnia double precision;
```

```
-- DROP FUNCTION kielce.trp_powierzchnia();

CREATE FUNCTION kielce.trp_powierzchnia()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS
$BODY$
begin

if tg_op = 'INSERT' then
  if new.geom is not null then
    new.powierzchnia = st_area(new.geom);
  end if;
elseif tg_op = 'UPDATE' then
  if new.geom is not null and not st_equals(old.geom, new.geom) then
    new.powierzchnia = st_area(new.geom);
  end if;
end if;

return new;
```

```
end
$BODY$;
```

```
-- DROP TRIGGER tr_powierzchnia ON kielce.kielce_budynki;

CREATE TRIGGER tr_powierzchnia
    BEFORE INSERT OR UPDATE
    ON kielce.kielce_budynki
    FOR EACH ROW
    EXECUTE PROCEDURE kielce.trp_powierzchnia();
```

Utworzenie wyzwalacza uzupełniającego adres dodawanego budynku

```
-- ALTER TABLE kielce.kielce_budynki DROP COLUMN powierzchnia;

ALTER TABLE kielce.kielce_budynki ADD COLUMN miasto text;
ALTER TABLE kielce.kielce_budynki ADD COLUMN miejscowosc text;
ALTER TABLE kielce.kielce_budynki ADD COLUMN ulica text;
ALTER TABLE kielce.kielce_budynki ADD COLUMN numer text;
```

```
-- DROP FUNCTION kielce.trp_adres();

CREATE or replace FUNCTION kielce.trp_adres()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS
$BODY$

declare
  adr record;

begin

  select * from osm.revgeocode(ST_X(ST_Transform(ST_Centroid(new.geom),
4326)), ST_Y(ST_Transform(ST_Centroid(new.geom), 4326))) into adr;
  new.miasto = adr.miasto;
  new.miejscowosc = adr.miejscowosc;
  new.ulica = adr.ulica;
  new.numer = adr.numer;
  return new;

end

$BODY$;
```

```
-- DROP TRIGGER tr_powierzchnia ON kielce.kielce_budynki;

CREATE TRIGGER tr_adres
    BEFORE INSERT OR UPDATE
    ON kielce.kielce_budynki
    FOR EACH ROW
    EXECUTE PROCEDURE kielce.trp_adres();
```

Blok 10 - Ustawienia i strojenie bazy danych

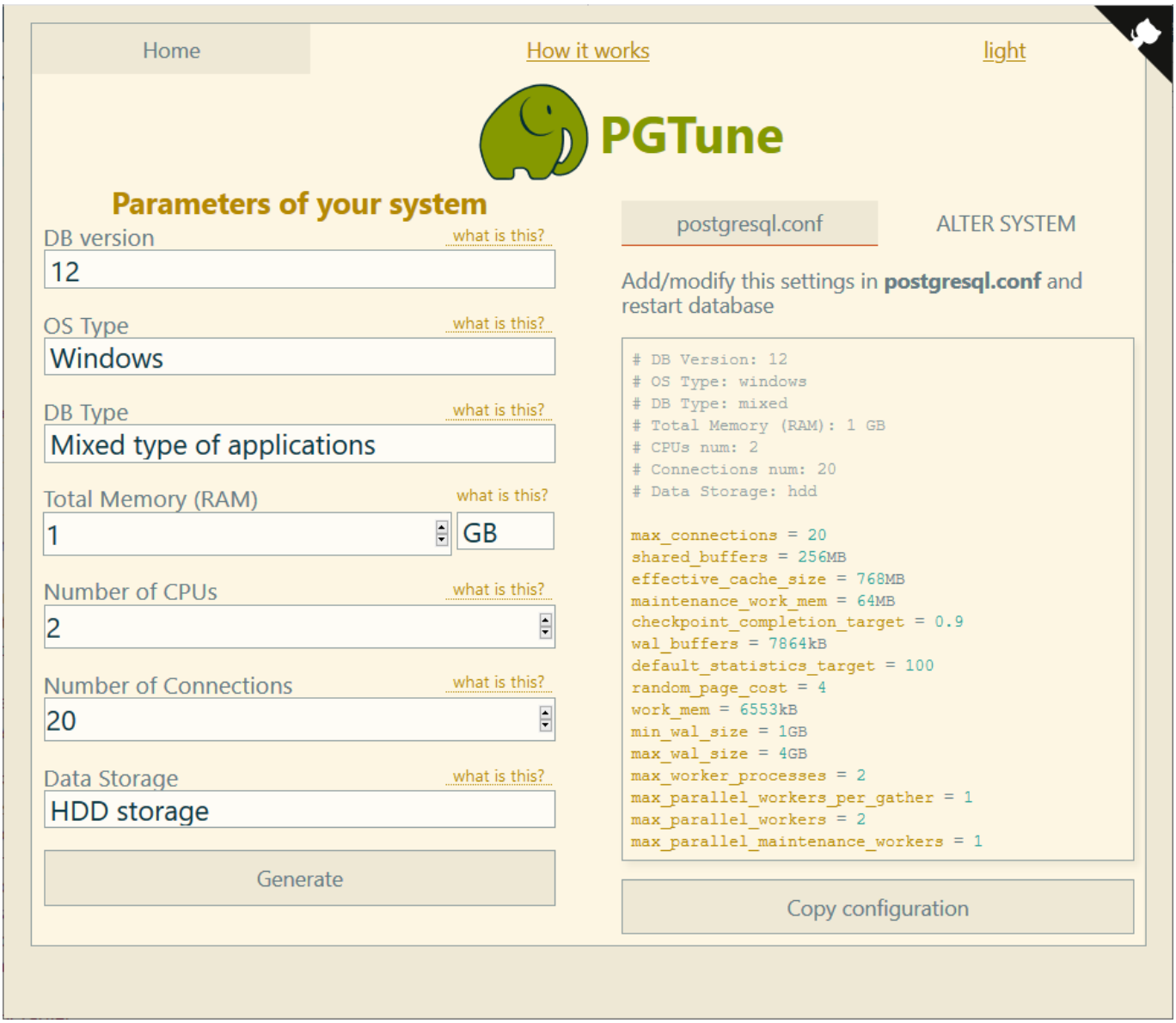
Strojenie bazy - wprowadzenie

Strojenie bazy to proces ciągły mający na celu poprawę wydajności działania bazy danych i realizowany między innymi przez:

- zmianę parametrów bazy danych
- sprzątanie `VACUUM` tabel
- aktualizację danych statystycznych bazy
- analizę planów wykonywania zapytań
- tworzenie indeksów
- partycjonowanie tabel
- klastrowanie tabel
- logowanie wolnych zapytań

Kalkulator parametrów

PostgreSQL pozwala na definiowanie wielu parametrów działania dla serwera bazy danych - szczegółowo opisane są one w dokumentacji. Najłatwiejszym sposobem na uzyskanie wydajnej startowej konfiguracji serwera jest użycie narzędzia PG_TUNE, które potrafi obliczyć je na podstawie zasobów serwera.



Plik konfiguracyjny vs. alter system

Istnieją dwie możliwości zmiany parametrów pracy serwera PostgreSQL:

- modyfikacja pliku konfiguracyjnego postgresql.conf
- polecenie SQL `ALTER SYSTEM`

Polecenie `ALTER SYSTEM` posiada następującą składnie:

```
ALTER SYSTEM SET configuration_parameter { TO | = } { value | 'value' |
DEFAULT }

ALTER SYSTEM RESET configuration_parameter
ALTER SYSTEM RESET ALL
```

Polecenie używane jest do zmiany parametrów serwera i jest wygodniejsze w użyciu niż tradycyjna metoda edycji pliku konfiguracyjnego. Wszystkie zmiany parametrów dokonane za pomocą tego polecenia są zapisywane do pliku postgresql.auto.conf, który przetwarzany jest zaraz po głównym pliku

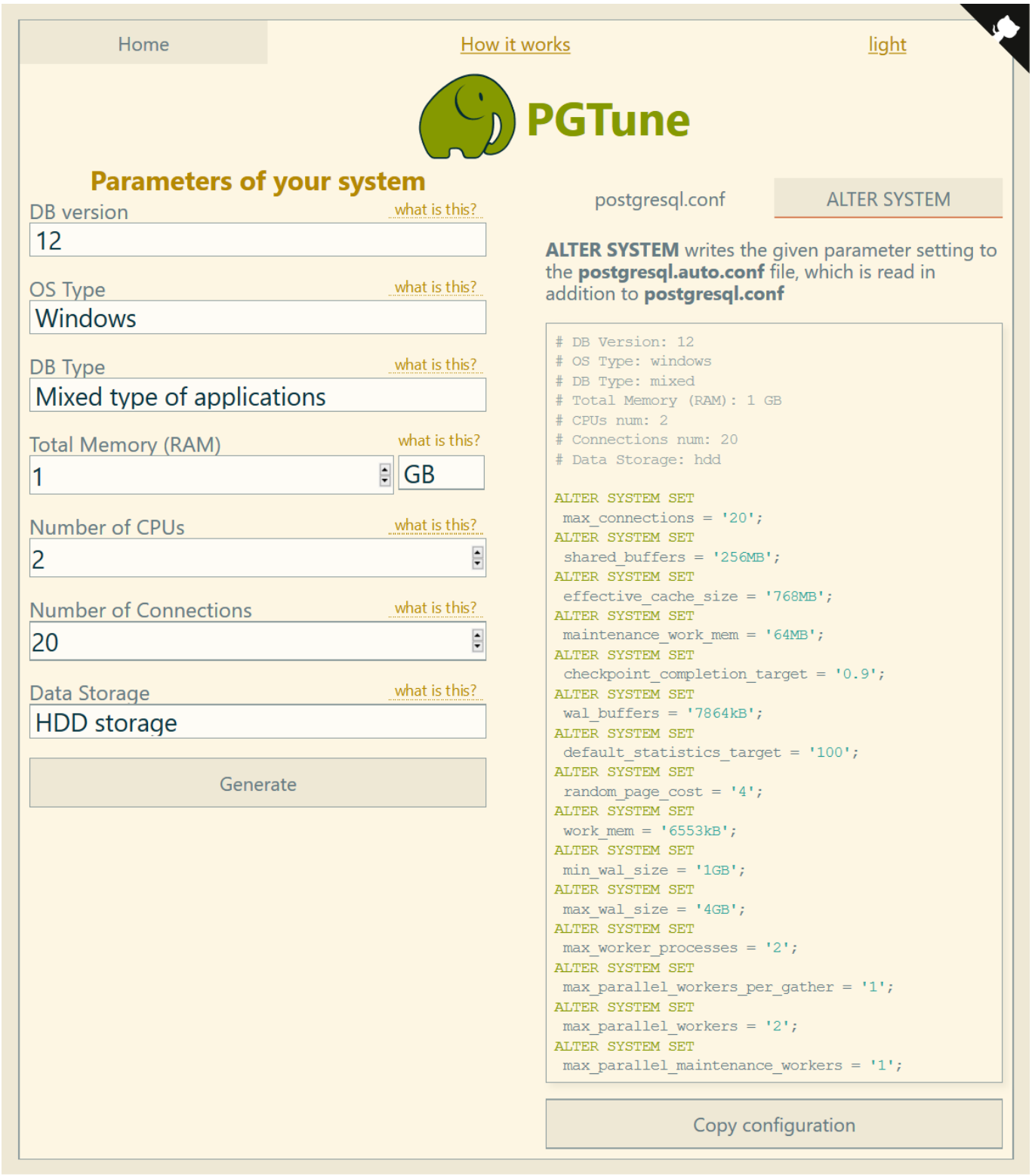
konfiguracyjnym, dzięki czemu mamy łatwy wgląd w zestaw zmian których dokonaliśmy w konfiguracji. `RESET` parametru powoduje usunięcie go z tego pliku, `RESET ALL` powoduje wyczyszczenie pliku.

Wartości parametrów ustawione za pomocą `ALTER SYSTEM` stają się aktywne po kolejnym przeładowaniu konfiguracji lub po restarcie serwera.

Z poziomu SQL mamy możliwość wymuszenia przeładowania konfiguracji za pomocą funkcji `pg_reload_conf()`

Polecenie `ALTER SYSTEM` może być wywołane wyłącznie przez superużytkownika i nie może być wykonane w bloku transakcji ani w funkcji.

Wspomniany powyżej kalkulator parametrów potrafi wygenerować parametry dla serwera zarówno jako zawartość pliku konfiguracyjnego jak i w postaci poleceń `ALTER SYSTEM`



Zestaw poleceń obliczony przez narzędzie PG Tune znajduje się poniżej:

```
# DB Version: 12
# OS Type: windows
# DB Type: mixed
# Total Memory (RAM): 1 GB
# CPUs num: 2
# Connections num: 20
# Data Storage: hdd

ALTER SYSTEM SET max_connections = '20';
```

```
ALTER SYSTEM SET shared_buffers = '256MB';
ALTER SYSTEM SET effective_cache_size = '768MB';
ALTER SYSTEM SET maintenance_work_mem = '64MB';
ALTER SYSTEM SET checkpoint_completion_target = '0.9';
ALTER SYSTEM SET wal_buffers = '7864kB';
ALTER SYSTEM SET default_statistics_target = '100';
ALTER SYSTEM SET random_page_cost = '4';
ALTER SYSTEM SET work_mem = '6553kB';
ALTER SYSTEM SET min_wal_size = '1GB';
ALTER SYSTEM SET max_wal_size = '4GB';
ALTER SYSTEM SET max_worker_processes = '2';
ALTER SYSTEM SET max_parallel_workers_per_gather = '1';
ALTER SYSTEM SET max_parallel_workers = '2';
ALTER SYSTEM SET max_parallel_maintenance_workers = '1';
```

Jeśli uruchamiamy te polecenia za pomocą PGAdmin konieczne jest uruchamianie każdego z osobna. Na zakończenie wymuszamy przeładowanie konfiguracji

```
select pg_reload_conf();
```

Vacuum, Vacuum full i Autovacuum

Gdy zmieniamy jakiś wiersz, PostgreSQL tworzy nową kopię wiersza na której My jako zmieniający operujemy. Kopia tego wiersza znajduje się w ramach tabeli w której znajduje się również oryginalny wiersz. Dla zapytań odpytujących tę tabelę dostępne są stare wiersze. Po zakończeniu transakcji wszystkie zapytania wszystkich transakcji korzystają już z nowych wierszy. Taki sposób działania sprawia, że w plikach związanych z tabelami z czasem powstaje ogromna ilość nieużywanych wierszy, do których już nawet nie ma dostępu. To z kolei powoduje rozrost plików danych. Miejsce zajmowane przez takie wiersze można odzyskać za pomocą polecenia VACUUM.

Jeśli zechcielibyśmy zmniejszyć wielkość plików danych, musielibyśmy użyć polecenia VACUUM FULL. Działa ono w ten sposób, że poza czynnościami wykonywanymi przez zwykły vacuum, przenosi martwe wiersze na koniec tabeli a następnie zmniejsza jej wielkość odzyskując wolne miejsce z przestrzeni zwalnianej przez martwe wiersze. VACUUM FULL oczywiście nie jest rozwiązaniem idealnym i ma swoje wady. Przede wszystkim jest bardzo obciążające dla serwera i powoduje przy dużych tabelach ogromne ilości I/O. Ponadto zakłada blokadę na wyłączność. Z tych powodów operację tę powinniśmy przeprowadzać kiedy baza nie jest zbyt obciążona.

Od wersji 8.1 PostgreSQL wprowadzono nowy mechanizm – demona autovacuum. Domyślnie jest on włączony. Mechanizm ten wykonuje te same czynności co vacuum, z tą różnicą że nie musimy go wywoływać ręcznie. Uruchamia się sam co czas określony w `autovacuum_naptime` domyślnie ustawionym na minutę.Autovacuum poszukuje tabel dla których została zmieniona, dodana lub skasowana znacząca ilość wierszy i przeprowadza w nich procesy czyszczące. Operacja wykonywana jest z użyciem takiej ilości procesów jaka jest ustawiona przez parametr `autovacuum_max_workers`.Ktoś dociekliwy mógłby zapytać – ile to jest „znacząca ilość wierszy” ? A to już określamy sami poprzez parametr `autovacuum_vacuum_threshold` domyślnie ustawiony na 50.Warto wiedzieć że autovacuum odświeża

też statystyki tabel na potrzeby lepszego wybierania planów wykonania. Robi to dla tych tabel, dla których ilość zmienionych wierszy przekroczy wartość określoną w parametrze autovacuum_analyze_threshold.

Tworzenie indeksów przestrzennych

Jak robiliśmy to już wielokrotnie indeksy przestrzenne tworzymy używając polecenia

```
create index gminy_geom_idx on osm.gminy using gist(geom);
```

Należy pamiętać, że aby każdy indeks, więc również przestrzenny, działał jego argument musi być zgodny z warunkiem w zapytaniu, więc jeśli w zapytaniu używamy porównania czy przecięcia przestrzennego używając centroidu argument indeksu również powinien zawierać centroid

```
create index gminy_geom_idx on osm.gminy using gist(ST_Centroid(geom));
```

Polecenie explain i interpretacja wyników

Jeśli zapytanie wykonuje się wolno punktem wyjścia jest sprawdzenie jego planu wykonania. Możemy się w tym celu posłużyć instrukcją EXPLAIN lub EXPLAIN ANALYZE. Ponieważ szacowanie kosztów wykonania zapytania oparte jest o statystyki tabel i indeksów, przed sprawdzeniem planu należy je odświeżyć dla wszystkich tabel biorących udział w zapytaniu komendą `ANALYZE`.

Sama komenda `EXPLAIN` przestawi tylko plan wykonania zapytania.

```
explain select * from osm.adresy;
```

```
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on adresy  (cost=0.00..12107.38 rows=294938 width=141)
(1 row)
```

Jeśli skorzystamy z komendy `EXPLAIN ANALYZE`, poza wymyśleniem planu wykonania zapytania, zostanie ono jeszcze wykonane i zostanie wyświetlony czas zarówno planowania jak i wykonania zapytania.

```
explain analyze select * from osm.adresy;
```

```
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------
-------------------------------------
 Seq Scan on adresy  (cost=0.00..12107.38 rows=294938 width=141) (actual
```

```
time=0.324..318.848 rows=294938 loops=1)
 Planning Time: 0.084 ms
 Execution Time: 327.270 ms
(3 rows)
```

Analizując plany wykonania zapytania, musimy mieć na uwadze, że dane mogą być cache'owane w buforze. Najlepiej jest wykonać zapytanie kilkukrotnie i sprawdzić czy kolejne wykonania nie będą szybsze od pierwszego. Jeśli tak będzie, oznaczać to będzie że zapytanie było za pierwszym razem wykonywane przy zimnym buforze.

Wiemy już w jaki sposób możemy wyświetlić plan wykonania zapytania, zajmiemy się więc teraz ich analizą, w tym celu użyjemy powyższej odpowiedzi.

- **Seq Scan on adresy** - Ta sekcja określa rodzaj skanu oraz obiekt na którym jest wykonywany. W tym przypadku jest to skan sekwencyjny na tabeli adresy.
- pierwszy nawias określa szacowane parametry
    - **cost=0.00..12107.38** - W tej części znajdziemy dwie wartości kosztu wykonania zapytania. Pierwsza to koszt początkowy określający koszt pobrania pierwszego wiersza. Zaskakująca może być wartość 0. Skan sekwencyjny zaczyna pobierać wiersze od razu, nie potrzebuje żadnych przygotować - stąd taka wartość.
    - **rows=294938** - To oszacowana liczba wierszy do wyświetlenia. Jeśli ta wartość bardzo się różni od rzeczywistej liczby wierszy w tabeli – powinien to być dla nas znak, że statystyki są nieaktualne i należy je odświeżyć.
    - **width=141** - Oszacowana liczba bajtów jaką średnio zajmuje jeden rekord. Oszacujmy więc wielkość tabeli na podstawie tych danych. Mamy (160000 wierszy * 8 bajtów każdy)/1024/1024 = 1,22 MB.
- drugi nawias oznacza faktyczne parametry
    - **actual time=0.324..318.848** - dwie wartości kosztu - tym razem rzeczywistego
    - **rows=294938** - faktyczna liczba przetworzonych wierszy
    - **loops=1** - Jeśli wartość parametru loops jest większa niż 1, oznacza to że dany węzeł był wykonywany więcej niż raz. Może tak się zdarzyć np. przy operacji łączenia tabel. Należy pamiętać, że wartość parametrów actual time i rows odnosi się do pojedynczego wykonania pętli. Jeśli ilość wykonań jest większa niż 1, należy te wartości pomnożyć przez ilość wykonań aby uzyskać faktyczny koszt i ilość wierszy przetworzonych w ramach danego węzła.

Jeśli dodamy do zapytania sortowanie to pierwszym węzłem jest sortowanie, które musi zostać wykonane zanim wiersze zaczną być przekazywane do aplikacji klienckiej. W tym przypadku koszt skanu sekwencyjnego pozostał taki sam, jednak jego rozpoczęcie wymaga wcześniejszego posortowania danych – tutaj koszt początkowy jest znacznie wyższy. Drugą wartością w podawanym koszcie jest kosztem pełnego wykonania węzła, a więc w tym przypadku odczytania całej tabeli.

```
explain analyze select * from osm.adresy order by miasto;
```

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------
 Gather Merge  (cost=39443.75..59395.44 rows=173493 width=141) (actual
time=154.756..349.213 rows=294938 loops=1)
   Workers Planned: 1
   Workers Launched: 1
   ->  Sort  (cost=38443.74..38877.47 rows=173493 width=141) (actual
time=130.956..188.463 rows=147469 loops=2)
         Sort Key: miasto
         Sort Method: external merge  Disk: 22104kB
         Worker 0:  Sort Method: external merge  Disk: 18272kB
         ->  Parallel Seq Scan on adresy  (cost=0.00..10892.93 rows=173493
width=141) (actual time=0.190..31.172 rows=147469 loops=2)
 Planning Time: 0.130 ms
 Execution Time: 357.763 ms
(10 rows)
```

Jeśli istnieje indeks który można byłoby wykorzystać w realizacji zapytania, najprawdopodobniej zostanie on wykorzystany zamiast skanu sekwencyjnego po tabeli. Weźmy na przykład zapytanie:

```
explain select * from osm.adresy where miasto = 'Maluszyn';
```

```
                                QUERY PLAN
-------------------------------------------------------------------------
-
 Gather  (cost=1000.00..12335.16 rows=85 width=141)
   Workers Planned: 1
   ->  Parallel Seq Scan on adresy  (cost=0.00..11326.66 rows=50 width=141)
         Filter: (miasto = 'Maluszyn'::text)
(4 rows)
```

Następnie utworzymy indeks który obsłuży warunek `where miasto = 'Maluszyn'`:

```
create index osm_adresy_miasto_idx on osm.adresy (miasto);
```

I ponownie przeanalizujemy wykonanie zapytania

```
explain select * from osm.adresy where miasto = 'Maluszyn';
```

```
                                QUERY PLAN
-------------------------------------------------------------------------
----------
 Bitmap Heap Scan on adresy  (cost=5.08..321.58 rows=85 width=141)
   Recheck Cond: (miasto = 'Maluszyn'::text)
```

```
    ->  Bitmap Index Scan on osm_adresy_miasto_idx  (cost=0.00..5.06 rows=85
width=0)
          Index Cond: (miasto = 'Maluszyn'::text)
(4 rows)
```

Indeks zostanie użyty również przy sortowaniu

```
explain analyze select * from osm.adresy order by miasto;
```

```
                                                                QUERY
PLAN
-------------------------------------------------------------------------------
-------------------------------------------------------------------
 Index Scan using osm_adresy_miasto_idx on adresy  (cost=0.42..44753.28
rows=294938 width=141) (actual time=0.081..105.951 rows=294938 loops=1)
 Planning Time: 1.474 ms
 Execution Time: 111.431 ms
(3 rows)
```

Zarządzanie indeksami w bazie danych

Indeks jest obiektem bazodanowym niezależnym logicznie i fizycznie od tabeli. Pozwala uzyskać szybszy dostęp do danych. Indeksy zakłada się na kolumnę w tabeli lub kilka kolumn naraz. Oczywiście bez nich wszystko będzie działać, jednak indeksy pozwolą nam szybciej dostać się do danych.

Indeksy przechowują wartości kolumn na które są nakładane oraz ROWID wiersza, dlatego w szczególnych przypadkach pobranie danych może odbyć się bez skanowania samej tabeli a jedynie indeksu.

Korzyść wydajnościowa ze stosowania indeksów jest największa w przypadku dużych tabel (zawierających najwięcej rekordów) oraz zapytań, które wykonywane są najczęściej. W PostgreSQL zaleca się indeksować następujące kolumny:

- kolumny najczęściej padające po słowie `WHERE`,
- kolumny dwóch tabel, które często łączymy `JOIN .. ON`,
- kolumny, według których sortujemy dane w raportach (kolumny padające po słowie `ORDER BY` i `GROUP BY`),
- kolumny które często zliczamy (`SUM()`, `AVG()`, `MIN()`, `MAX()`, `COUNT()`)
- klucze obce i kolumny, których będziemy używać tak jak kluczy obcych,
- klucze unikalne `UNIQUE_KEY` (typu NIP, PESEL itd…),

Można się kierować prostą zasadą, polegającą na tym, że nie tworzymy indeksu jeżeli nie jesteśmy przekonani, że faktycznie będziemy z niego korzystać.

Problemy wynikające z użycia indeksów

- **Konieczność aktualizacji** - Z indeksami wcale nie jest tak różowo jak mogłoby się wydawać. Z jednej strony mogą nam pomóc w przyspieszeniu odczytu danych, ale trzeba wziąć też pod uwagę że takie indeksy trzeba będzie aktualizować przy wykonywaniu operacji `UPDATE`, `DELETE` i `INSERT`. To powoduje wydłużenie tych operacji. Nie możemy więc zakładać więcej indeksów niż jest niezbędne i nie powinniśmy ich stosować tam gdzie korzyść z ich zastosowania jest znikoma.
- **Zajęte miejsce** - Indeksy muszą być przechowywane na dysku podobnie jak tabele. Jeśli więc np. utworzysz na jakiejś tabeli indeksy na każdej kolumnie, musisz liczyć się z tym, że ilość zajmowanego miejsca na potrzeby danej tabeli oraz jej indeksów przynajmniej się podwoi.
- **Blokady podczas tworzenia i odbudowywania** - Podczas budowania lub odbudowywania indeksu nakładana jest blokada na wszystkie wiersze których dotyczy. Wydawać by się mogło że to nic poważnego, tymczasem zakładanie indeksu na tabeli liczącej kilkaset tysięcy rekordów może trwać bardzo długo... a zwłaszcza jeśli mówimy o indeksach przestrzennych. Wszystkie transakcje które będą w tym czasie próbowały założyć swoją blokadę będą czekały (nie zostanie zgłoszony błąd) na zakończenie budowania indeksu. Takie transakcje mogły wcześniej zablokować inne zasoby. Aby ten problem „obejść" możesz zastosować współbieżne tworzenie indeksu.

Ogólne informacje o utworzonych indeksach możemy uzyskać uruchamiając polecenie:

```
SELECT
    pg_class.relname,
    pg_size_pretty(pg_class.reltuples::bigint) AS rows_in_bytes,
    pg_class.reltuples AS num_rows,
    count(indexname) AS number_of_indexes,
    CASE WHEN x.is_unique = 1 THEN 'Y'
       ELSE 'N'
    END AS UNIQUE,
    SUM(case WHEN number_of_columns = 1 THEN 1
              ELSE 0
            END) AS single_column,
    SUM(case WHEN number_of_columns IS NULL THEN 0
             WHEN number_of_columns = 1 THEN 0
             ELSE 1
           END) AS multi_column
FROM pg_namespace
LEFT OUTER JOIN pg_class ON pg_namespace.oid = pg_class.relnamespace
LEFT OUTER JOIN
       (SELECT indrelid,
           max(CAST(indisunique AS integer)) AS is_unique
       FROM pg_index
       GROUP BY indrelid) x
       ON pg_class.oid = x.indrelid
LEFT OUTER JOIN
    ( SELECT c.relname AS ctablename, ipg.relname AS indexname, x.indnatts
AS number_of_columns FROM pg_index x
           JOIN pg_class c ON c.oid = x.indrelid
           JOIN pg_class ipg ON ipg.oid = x.indexrelid  )
    AS foo
    ON pg_class.relname = foo.ctablename
WHERE
     pg_namespace.nspname='public'
```

```
AND  pg_class.relkind = 'r'
GROUP BY pg_class.relname, pg_class.reltuples, x.is_unique
ORDER BY 2;
```

Poniższe polecenie wyświetli wielkości i statystyki użycia poszczególnych indeksów:

```
SELECT
    t.schemaname,
    t.tablename,
    indexname,
    c.reltuples AS num_rows,
    pg_size_pretty(pg_relation_size(quote_ident(t.schemaname)::text || '.'
|| quote_ident(t.tablename)::text)) AS table_size,
    pg_size_pretty(pg_relation_size(quote_ident(t.schemaname)::text || '.'
|| quote_ident(indexrelname)::text)) AS index_size,
    CASE WHEN indisunique THEN 'Y'
        ELSE 'N'
    END AS UNIQUE,
    number_of_scans,
    tuples_read,
    tuples_fetched
FROM pg_tables t
LEFT OUTER JOIN pg_class c ON t.tablename = c.relname
LEFT OUTER JOIN (
    SELECT
        c.relname AS ctablename,
        ipg.relname AS indexname,
        x.indnatts AS number_of_columns,
        idx_scan AS number_of_scans,
        idx_tup_read AS tuples_read,
        idx_tup_fetch AS tuples_fetched,
        indexrelname,
        indisunique,
        schemaname
    FROM pg_index x
    JOIN pg_class c ON c.oid = x.indrelid
    JOIN pg_class ipg ON ipg.oid = x.indexrelid
    JOIN pg_stat_all_indexes psai ON x.indexrelid = psai.indexrelid
) AS foo ON t.tablename = foo.ctablename AND t.schemaname = foo.schemaname
WHERE t.schemaname NOT IN ('pg_catalog', 'information_schema')
ORDER BY 1,2;
```

Zdublowane indeksy możemy odnaleźć uruchamiając polecenie:

```
SELECT pg_size_pretty(sum(pg_relation_size(idx))::bigint) as size,
       (array_agg(idx))[1] as idx1, (array_agg(idx))[2] as idx2,
       (array_agg(idx))[3] as idx3, (array_agg(idx))[4] as idx4
FROM (
    SELECT indexrelid::regclass as idx, (indrelid::text ||E'\n'||
```

```
    indclass::text ||E'\n'|| indkey::text ||E'\n'||
                                            coalesce(indexprs::text,'')||E'\n'
    || coalesce(indpred::text,'')) as key
        FROM pg_index) sub
    GROUP BY key HAVING count(*)>1
    ORDER BY sum(pg_relation_size(idx)) DESC;
```

Rozszerzenie pg_stat_statements

Pg_stat_statements to rozszerzenie bazy danych pozwalające na analizowanie i rejestrowanie wszystkich operacji wykonywanych na bazie danych, przez co jest niezastąpione w procesie strojenia bazy.

Rozszerzenie wymaga załadowania do pamięci dodatkowych bibliotek, dlatego żeby je uruchomić musimy najpierw edytować konfigurację serwera

```
alter system set shared_preload_libraries = 'pg_stat_statements';
```

Zmiana konfiguracji wymaga restartu serwera bazy, więc uruchamiamy wiersz polecenia jako administrator i zatrzymujemy serwer:

```
net stop postgresql-x64-12
```

Konsola wyświetli

```
Usługa postgresql-x64-12 - PostgreSQL Server 12 jest właśnie zatrzymywana.
Usługa postgresql-x64-12 - PostgreSQL Server 12 została zatrzymana
pomyślnie.
```

Po czym uruchamiamy ponownie:

```
net start postgresql-x64-12
```

Konsola wyświetli:

```
Usługa postgresql-x64-12 - PostgreSQL Server 12 jest właśnie uruchamiana.
Pomyślnie uruchomiono usługę postgresql-x64-12 - PostgreSQL Server 12.
```

Aby włączyć rozszerzenie dla bazy danych musimy wykonać komendę SQL:

```
create extension pg_stat_statements;
```

Gdy rozszerzenie jest uruchomione śledzi ono wszystkie zapytania wykonywane na bazie danych. Zebrane dane udostępnia za pomocą widoku `pg_stat_statements`, który zawiera następujące dane:

- **userid** - OID użytkownika który wykonał zapytanie
- **dbid** - OID bazy danych na której wykonano zapytanie
- **queryid** - unikalny identyfikator zapytania
- **query** - treść zapytania
- **calls** - ilość wykonań
- **total_time** - łączny czas wykonywania zapytania w milisekundach
- **min_time** - minimalny czas wykonywania zapytania w milisekundach
- **max_time** - maksymalny czas wykonywania zapytania w milisekundach
- **mean_time** - średni czas wykonywania zapytania w milisekundach
- **stddev_time** - dewiacja czasów wykonywania zapytania
- **rows** - łączna liczba zwróconych rekordów
- **shared_blks_hit** - łączna liczba bloków cache użytych przez zapytanie
- **shared_blks_read** - łączna liczba bloków cache odczytanych przez zapytanie
- **shared_blks_dirtied** - łączna liczba bloków cache oznaczonych do odświeżenia przez zapytanie
- **shared_blks_written** - łączna liczba bloków cache zapisanych przez zapytanie
- **local_blks_hit** - łączna liczba bloków lokalnych użytych przez zapytanie
- **local_blks_read** - łączna liczba bloków lokalnych odczytanych przez zapytanie
- **local_blks_dirtied** - łączna liczba bloków lokalnych oznaczonych do odświeżenia przez zapytanie
- **local_blks_written** - łączna liczba bloków lokalnych zapisanych przez zapytanie
- **temp_blks_read** - łączna liczba bloków tymczasowych odczytanych przez zapytanie
- **temp_blks_written** - łączna liczba bloków tymczasowych zapisanych przez zapytanie
- **blk_read_time** - łączny czas odczytu bloków w milisekundach (jeśli włączona jest opcja `track_io_timing`, jeśli nie to 0)
- **blk_write_time** - łączny czas zapisu bloków w milisekundach (jeśli włączona jest opcja `track_io_timing`, jeśli nie to 0)

Z powodów zachowania bezpieczeństwa tylko superużytkownicy widzą identyfikatory oraz treści zapytań w statystykach.

Statystyki można wyczyścić za pomocą funkcji `pg_stat_statements_reset`.