Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

Ministerstwo Klimatu i Środowiska

OnGeo

Język SQL w praktyce w PostgreSQL z PostGIS

Relacyjne bazy danych

Poziom średniozaawansowany

MATERIAŁY SZKOLENIOWE

Spis treści

1. Wstęp

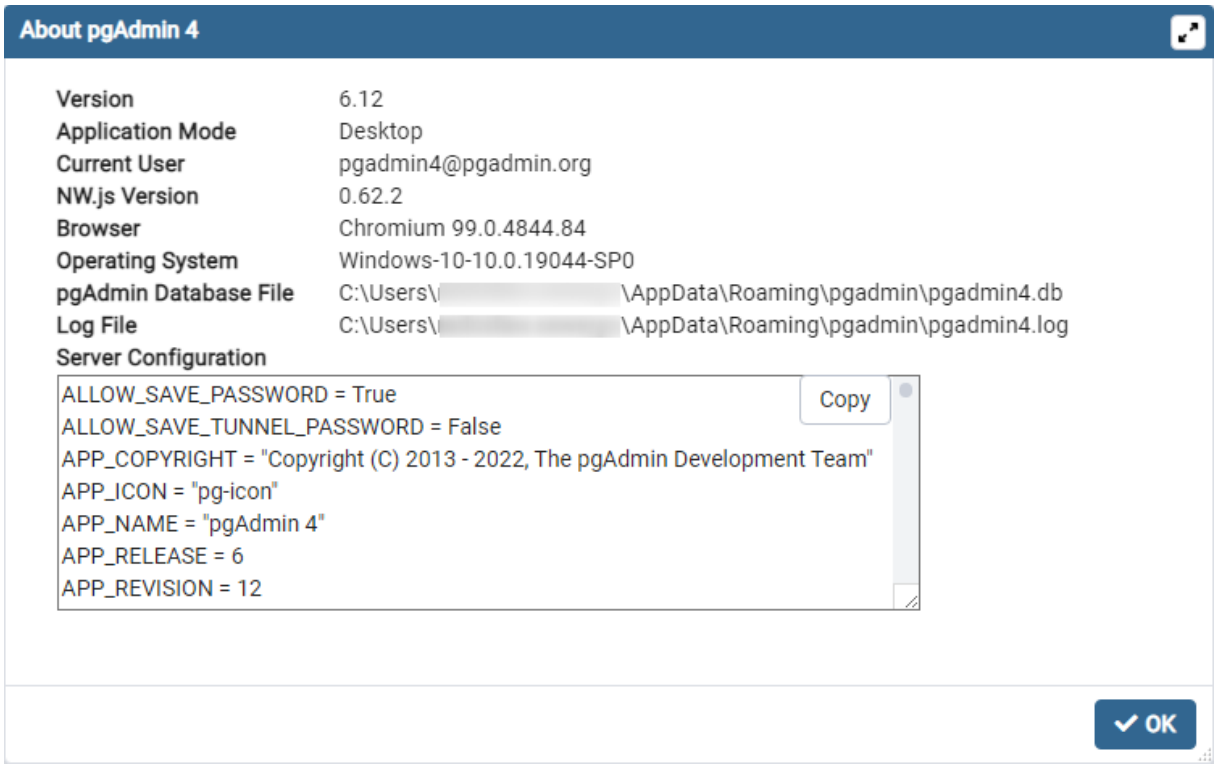
Do przygotowania skryptu wykorzystane zostały dane pochodzące z następujących zbiorów:

- Numeryczny Model Terenu,
- Dane Państwowego Rejestru Granic,
- Dane Państwowego Rejestru Nazw Geograficznych,
- Dane Państwowego Zasobu Geodezyjnego i Kartograficznego,
- Materiały szkoleniowe dostępne na stronie ekoportal.gov.pl.

Do przygotowania ćwiczeń wykorzystany został program QGIS Białowieża w wersji LTR o numerze 3.22.10.

Środowisko bazodanowe – PostgreSQL (postgresql-14.5-1-windows-x64.exe)

Środowisko bazodanowe – PostgreSQL (postgresql-14.5-1-windows-x64.exe)



2. Wprowadzenie do pracy w środowisku bazodanowym PostgreSQL

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.

pgAdmin

Oprogramowanie open source do zarządzania oraz pracy z bazą danych PostgreSQL, posiadające interfejs graficzny. Umożliwia ono administrowanie bazą, tworzenie replik baz danych, tworzenie dowolnych obiektów bazodanowych oraz zapytań do bazy danych.

3. Instalacja i konfiguracja systemu bazodanowego PostgreSQL

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

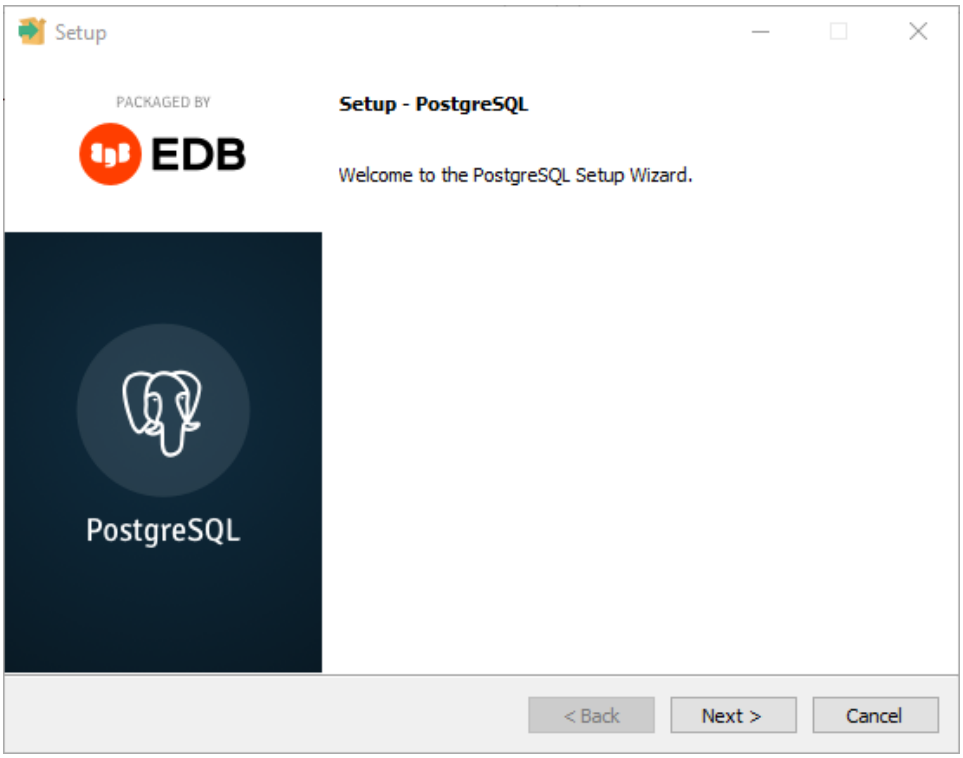
W systemach operacyjnych Windows, zgodnie z oficjalną dokumentacją bazę danych wraz z rozszerzeniami należy instalować używając instalatorów dostarczanych przez EnterpriseDB. Instalatory te zawierają:

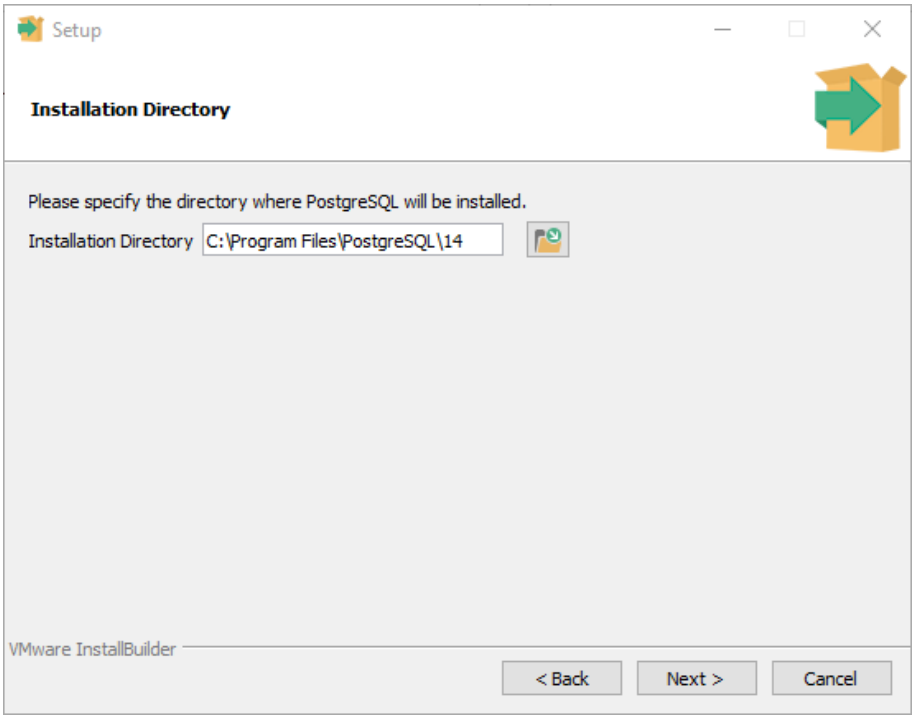
- serwer bazy danych
- pgAdmin - narzędzie do zarządzania oraz pracy na bazie danych
- StackBuilder - manager pakietów pozwalający na pobieranie oraz instalowanie dodatkowych rozszerzeń i sterowników.

Instalator może być uruchomiony w trybie interaktywnym oraz cichym.
Należy przejść na stronę https://www.enterprisedb.com/downloads/postgres-postgresql-downloads i wybrać odpowiednią wersję. Szkolenie zostało przygotowane dla wersji PostgreSQL 14. Instalacja jest możliwa wyłącznie na 64-bitowych systemach operacyjnych.

Ćwiczenie 1. Instalacja PostgreSQL i PostGIS w środowisku Windows

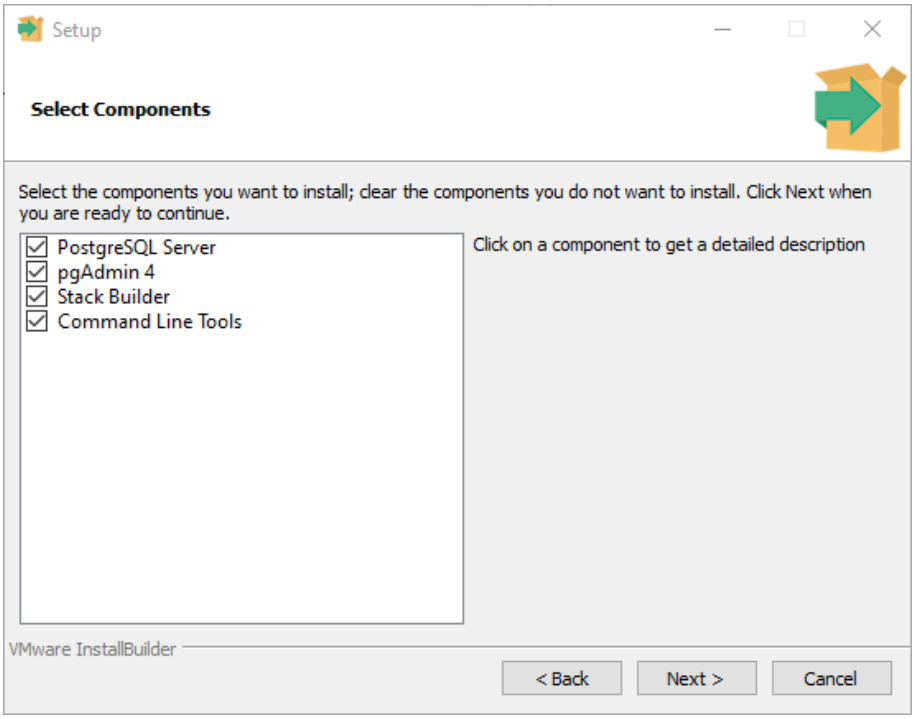
1. Proces instalacji przebiega podobnie, jak instalacja innego oprogramowania w systemie Windows. Należy pamiętać, że w jednym systemie może być zainstalowanych kilka wersji PostgreSQL. Podczas instalacji należy odpowiedzieć na pytanie odnośnie lokalizacji katalogu z systemem:



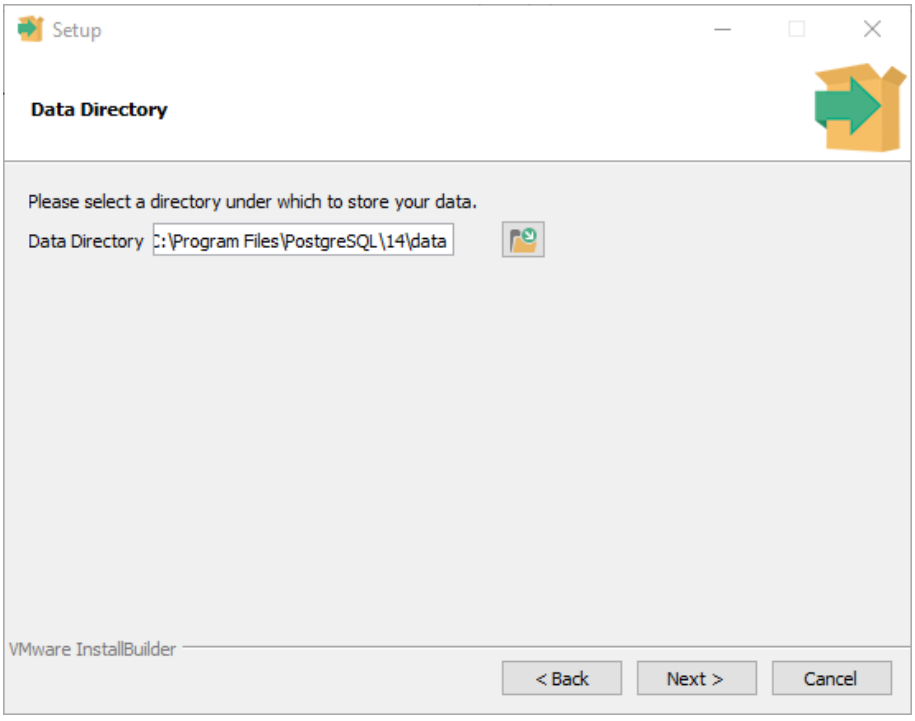


- można pozostawić domyślne.

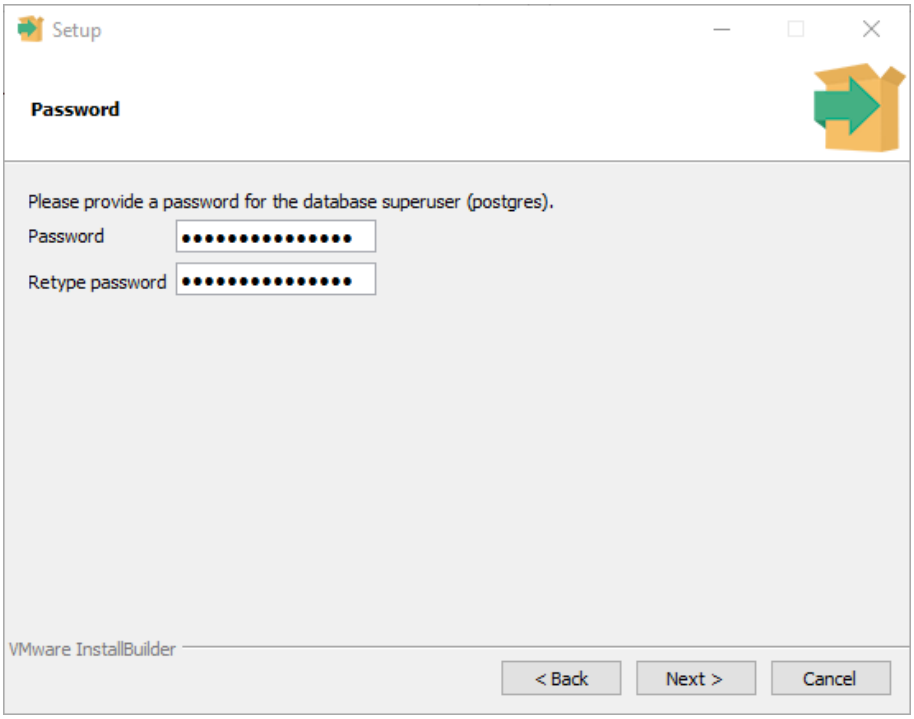
2. W wyborze komponentów należy wybrać wszystkie:



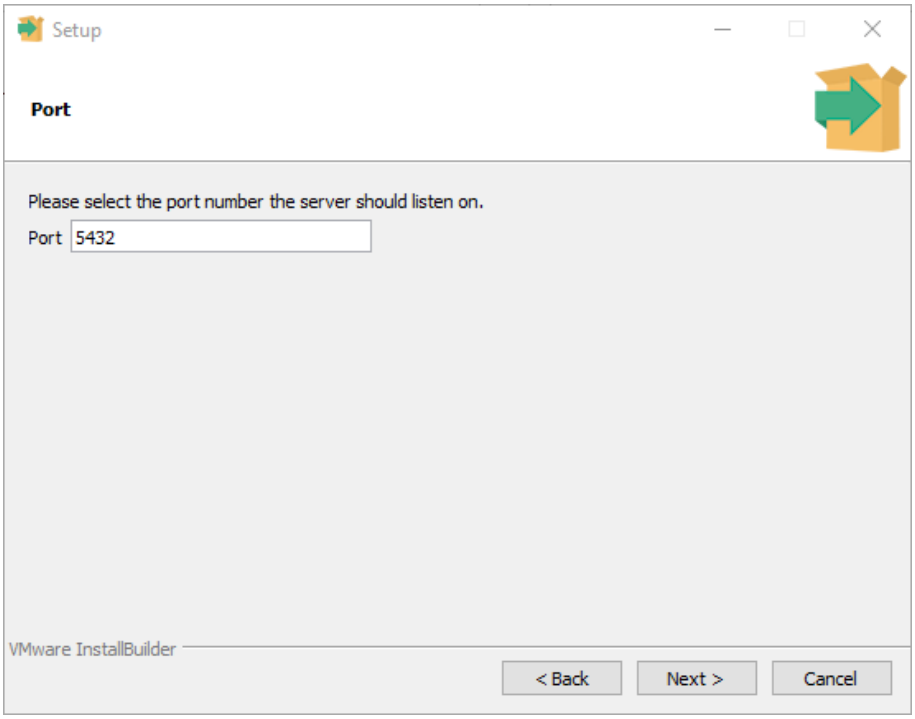
3. Katalog z danymi można pozostawić domyślny, jeśli komputer jest wyposażony w więcej niż jeden twardy dysk, należy w miarę możliwości wybrać najszybszy.



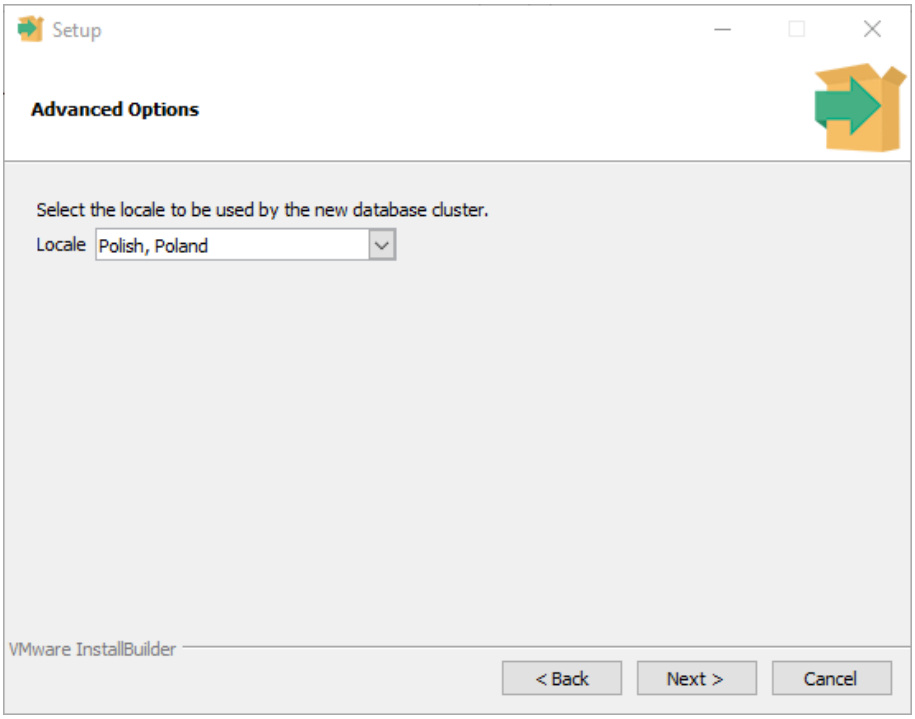
4. Ważnym elementem instalacji jest nadanie hasła superużytkownika. Hasło to należy chronić przed zagubieniem, gdyż nie istnieje prosta metoda na jego odzyskanie.



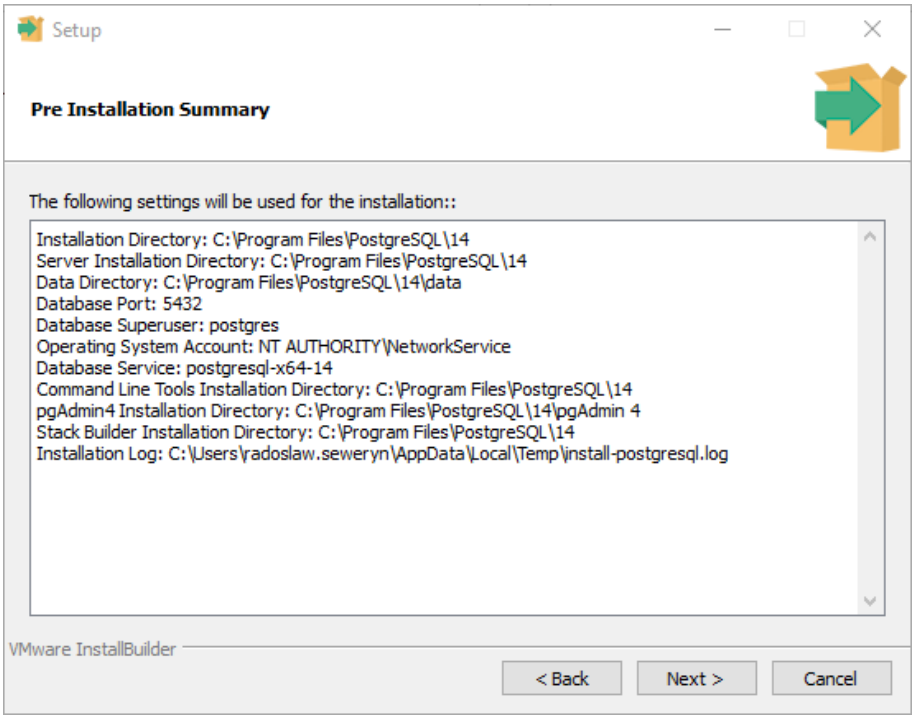
5. Numer portu można pozostawić domyślny: 5432. W przypadku instalacji kolejnych wersji PostgreSQL, instalator zaproponuje najbliższy wolny numer (np. 5433)



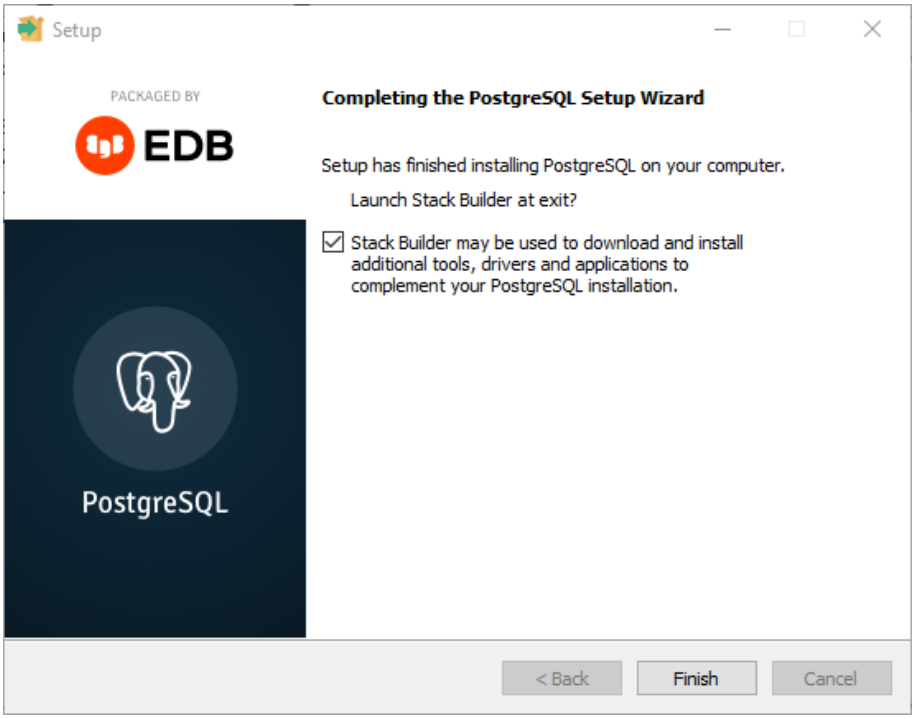
6. Wersję językową należy zmienić na polską: pl_PL.UTF8.



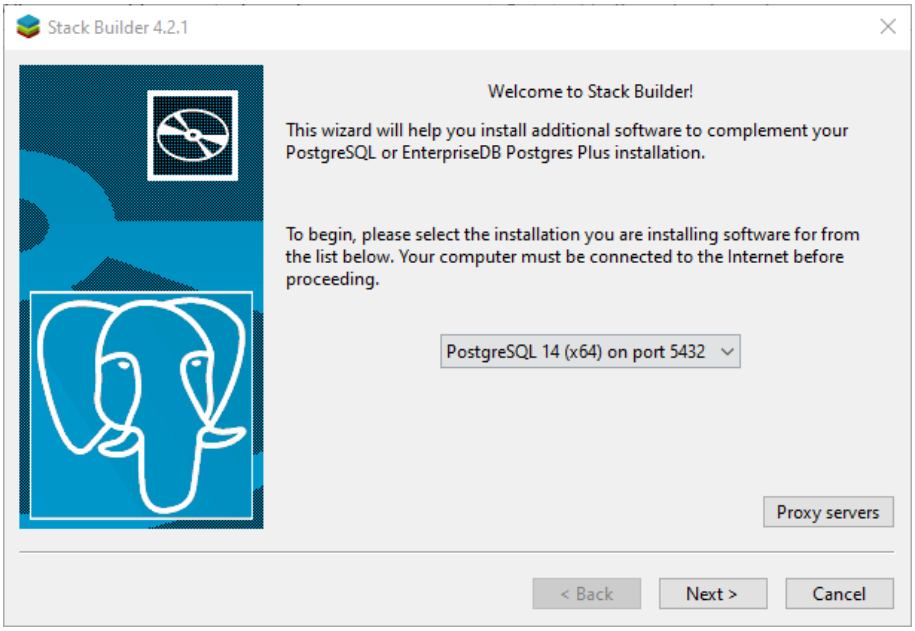
7. Następnie instalator wyświetli podsumowanie i przejdzie do właściwej instalacji systemu.



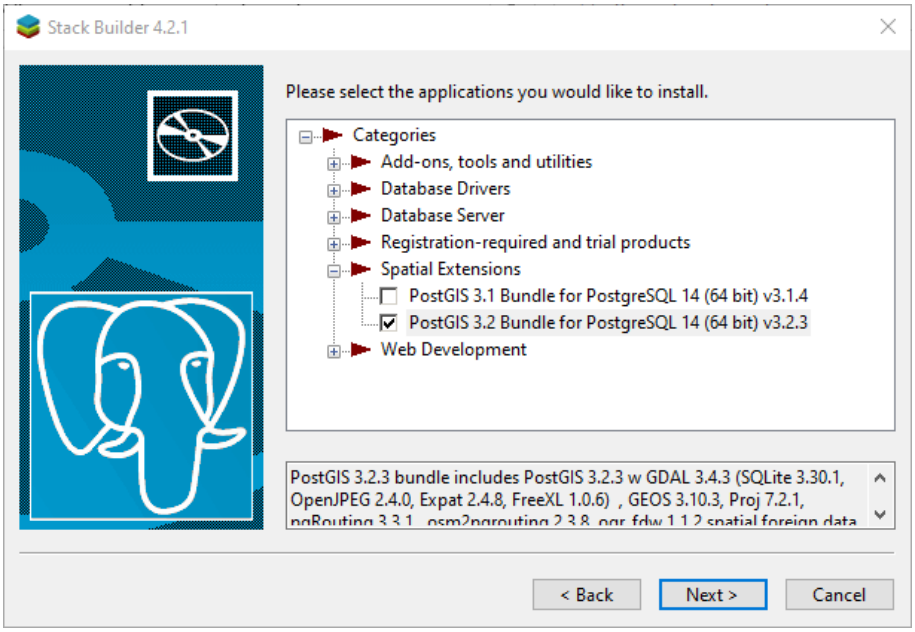
8. Po zakończeniu instalacji silnika PostgreSQL należy uruchomić instalator rozszerzeń - StackBuilder. Wystarczy zgodzić się na uruchomienie pozostawiając zaznaczone pole wyboru "Launch StackBuilder at exit?".

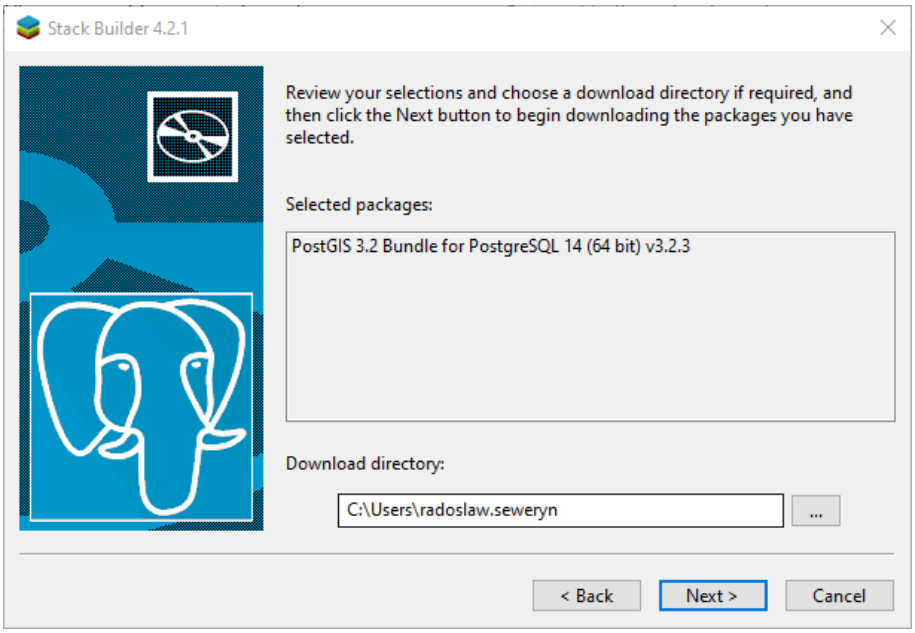


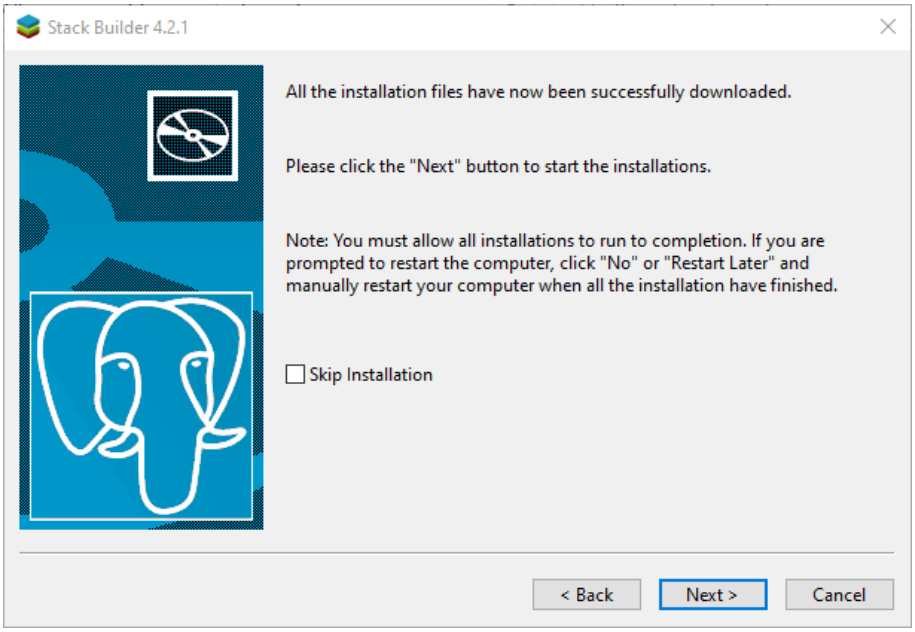
9. W instalatorze rozszerzeń należy wybrać właśnie zainstalowaną wersję PostgreSQL.



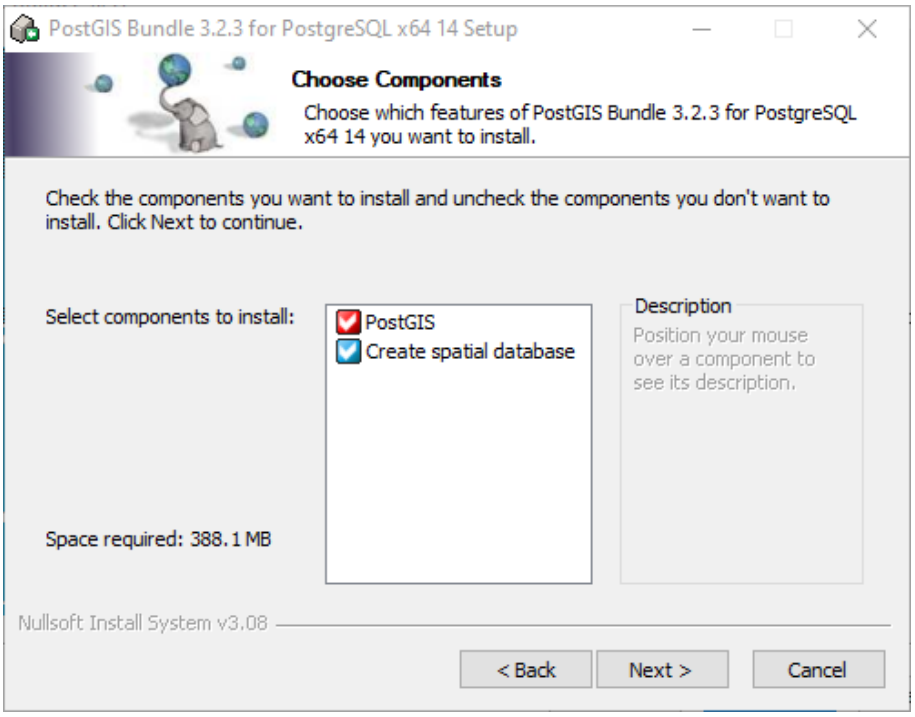
10. W sekcji Spatial Extensions należy wybrać PostGIS.

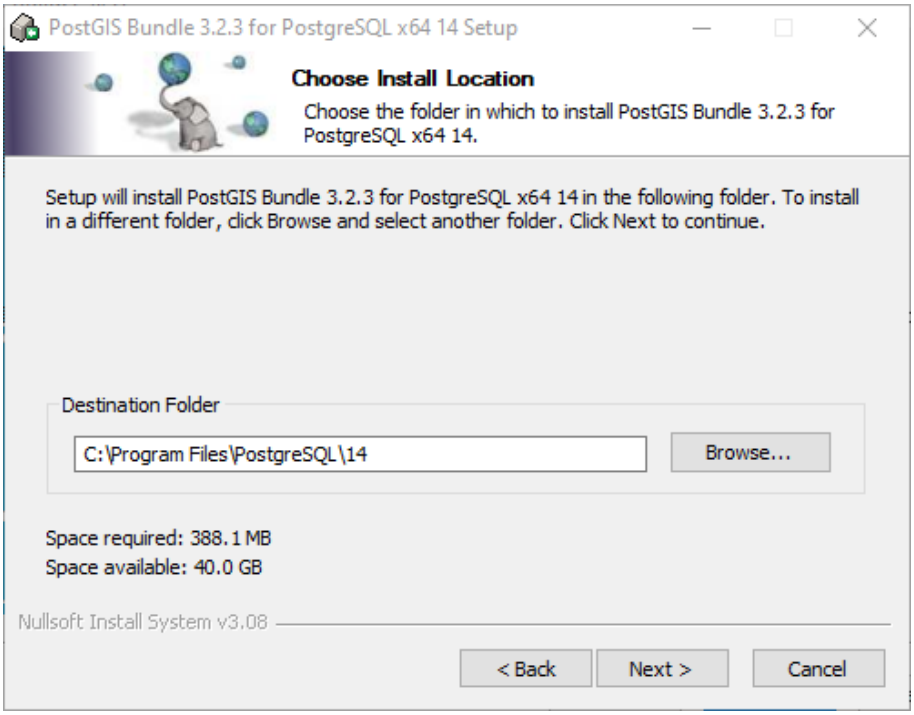

Stack Builder 4.2.1
Please select the applications you would like to install.
Categories
Add-ons, tools and utilities
Database Drivers
Database Server
Registration-required and trial products
Spatial Extensions
PostGIS 3.1 Bundle for PostgreSQL 14 (64 bit) v3.1.4
PostGIS 3.2 Bundle for PostgreSQL 14 (64 bit) v3.2.3
Web Development
PostGIS 3.2.3 bundle includes PostGIS 3.2.3 w GDAL 3.4.3 (SQLite 3.30.1, OpenJPEG 2.4.0, Expat 2.4.8, FreeXL 1.0.6) , GEOS 3.10.3, Proj 7.2.1,
< Back
Next >
Cancel


Stack Builder 4.2.1
Review your selections and choose a download directory if required, and then click the Next button to begin downloading the packages you have selected.
Selected packages:
PostGIS 3.2 Bundle for PostgreSQL 14 (64 bit) v3.2.3
Download directory:
C:\Users\radoslaw.seweryn
...
< Back
Next >
Cancel

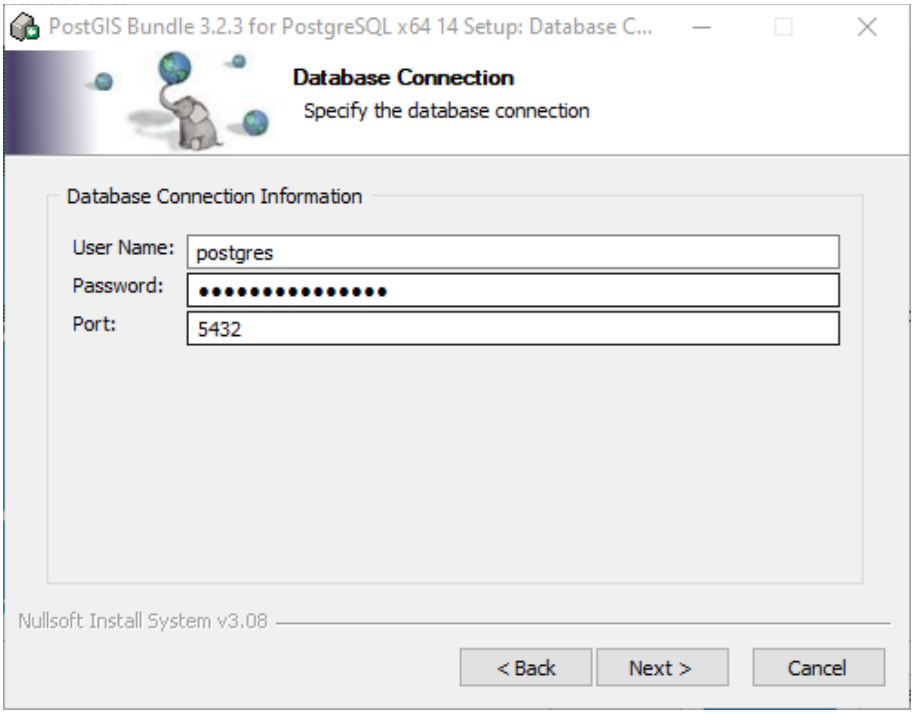


11. Instalator PostGIS daje możliwość utworzenia nowej, pustej bazy danych - skorzystajmy z tej możliwości.

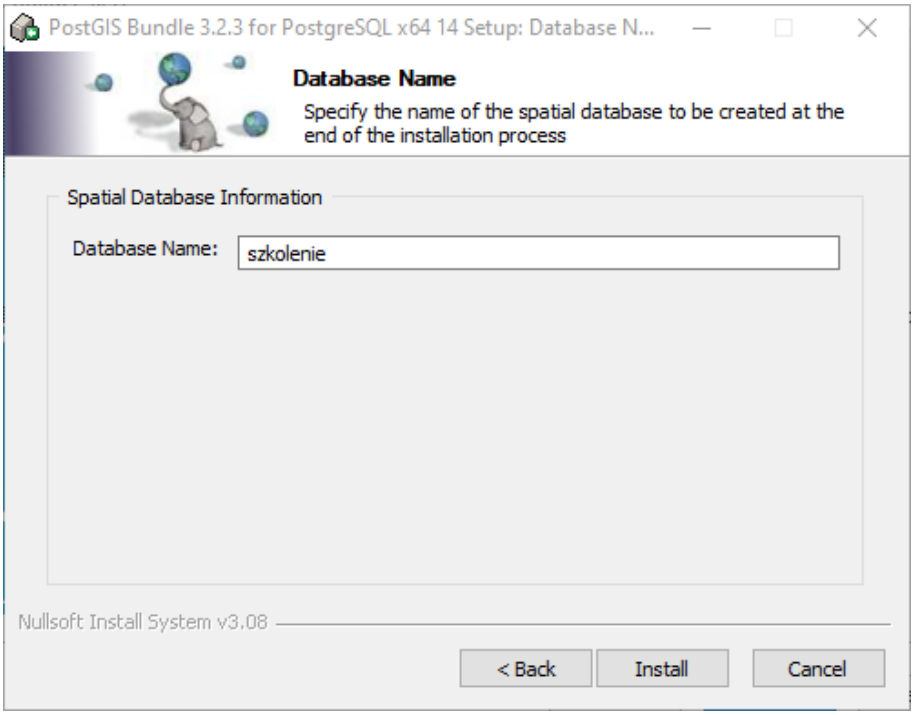




12. W trakcie instalacji PostGIS będzie konieczne podanie hasła super-użytkownika.



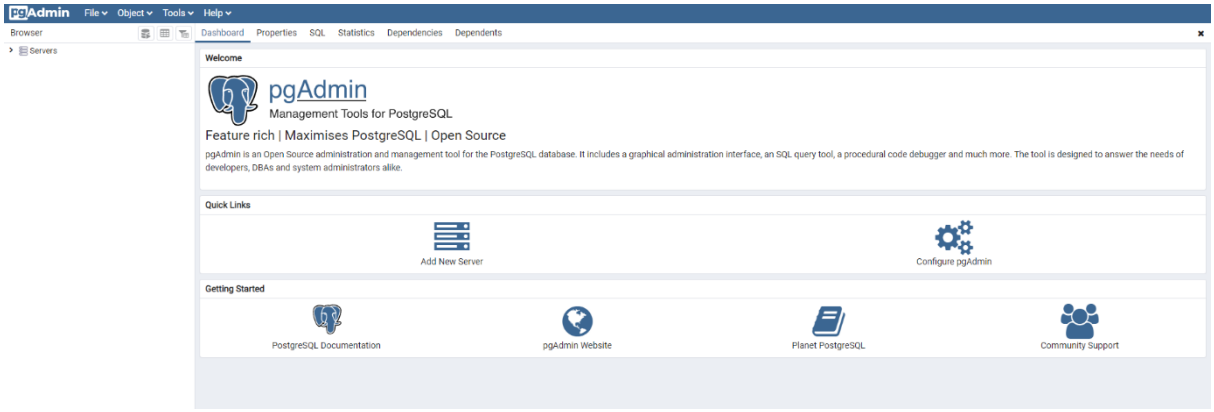
13. Dla bazy danych należy w miejsce "postgis_25_sample" wpisać nazwę "szkolenie".



14. Na następne 3 pytania o zmienne środowiskowe należy odpowiedzieć twierdząco. Po zakończeniu instalacji, system będzie gotowy do pracy i zasilenia danymi.

4. Praca z klientem bazy danych - pgAdmin

pgAdmin jest programem działającym w środowisku przeglądarki i umożliwia korzystanie z zaawansowanych funkcji systemu PostgreSQL. Uruchomienie pgAdmin odbywa się poprzez odnalezienie pozycji "**pgAdmin 4 v6**" w menu Start. Zostanie uruchomiona domyślna przeglądarka.



Po instalacji pgAdmin wraz z systemem PostgreSQL jest skonfigurowane połączenie lokalne z użyciem konta superużytkownika. Należy utworzyć nowe połączenie z wykorzystaniem użytkownika "kursant".

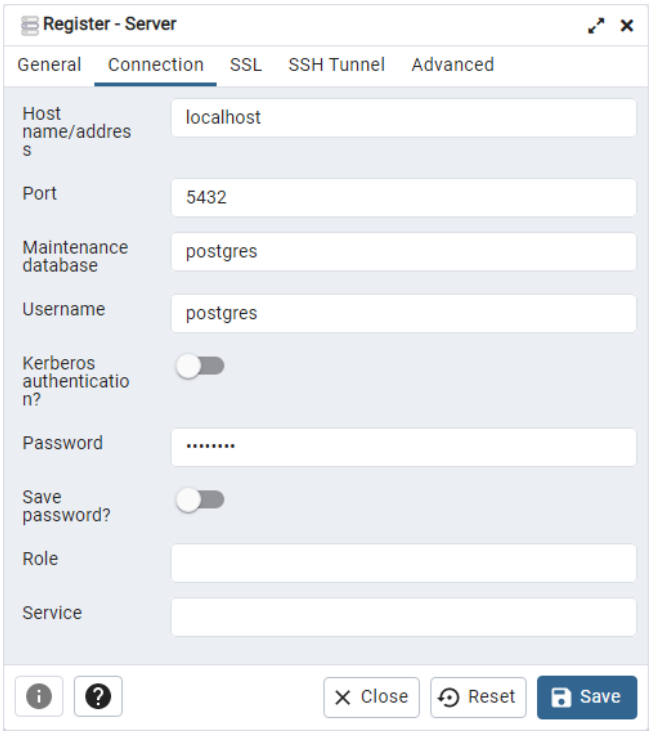
Ćwiczenie 2. Połączenie z bazą PostgreSQL w aplikacji pgAdmin

1. Klikamy prawym przyciskiem myszy na **Servers**.
2. Wybieramy z menu kontekstowego **Register→Serwer**

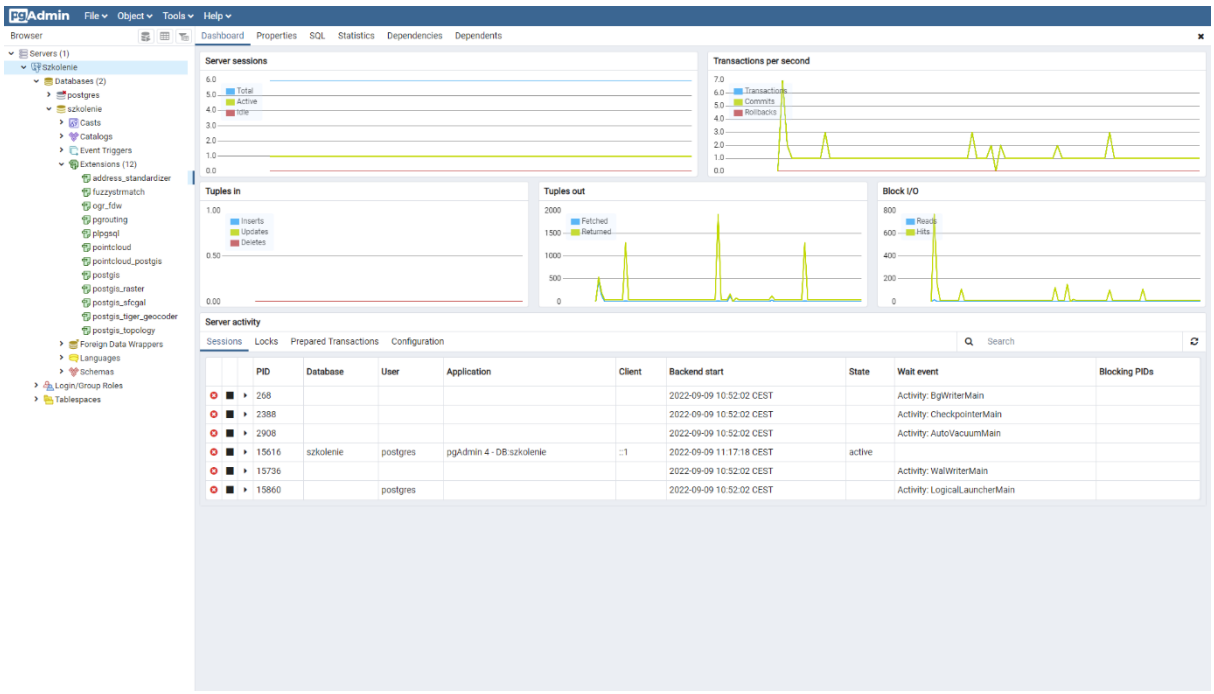
WAŻNE:

We wcześniejszych wersjach aplikacji polecenie to nazywa się **Create**. Zdarza się, że użytkownicy wykorzystują do obsługi bazy także starsze wersje aplikacji pgAdmin, z uwagi na stabilniejsze działanie w danym środowisku serwerowo-bazodanowym. Warto też nadmienić, że w niektórych wersjach aplikacji pgAdmin występowały mniej lub bardziej uciążliwe problemy w działaniu wybranych funkcji. W przypadku błędów lub niestabilnego działania zaleca się skorzystanie z najnowszej wersji aplikacji lub zainstalowanie wersji wcześniejszej.

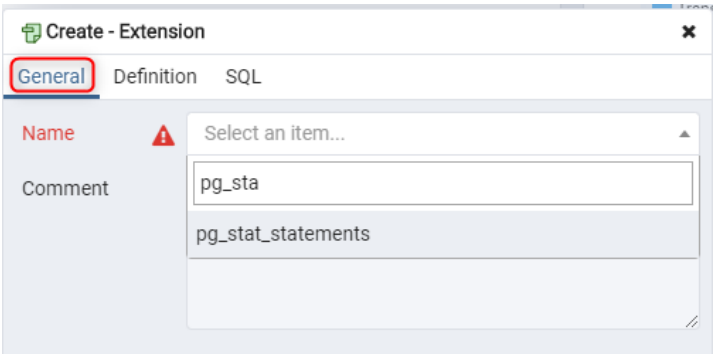
3. Wypełniamy formularz - na zakładce **General** podajemy nazwę połączenia, a na zakładce **Connection** wypełniamy pola:
   a. Host: **localhost**,
   b. Port: **5432**,
   c. Maintenance database: **szkolenie**,
   d. Username: **postgres**,
   e. Password: **hasło z instalacji**,
   f. Save Password: **zaznaczone**.



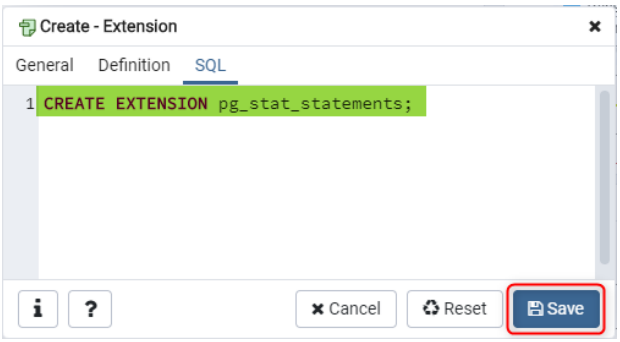
4. Po uzyskaniu połączenia, pojawi się okno aplikacji z połączoną bazą.



5. Zgodnie z wcześniejszymi informacjami 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.



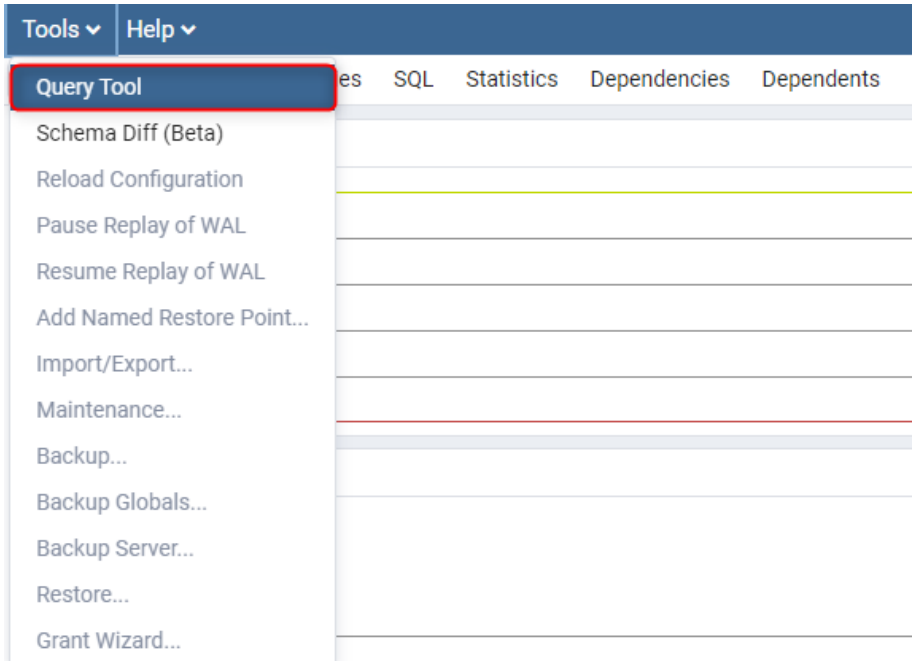
6. Zasada działania aplikacji pgAdmin jest taka, że pozwala ona za pomocą interfejsu graficznego wygenerować zapytania SQL, które później wysyłane są do bazy. Każde wygenerowane zapytanie możemy podejrzeć na zakładce *SQL*.



W naszym przypadku jest to zapytanie:

```
CREATE EXTENSION pg_stat_statements;
```

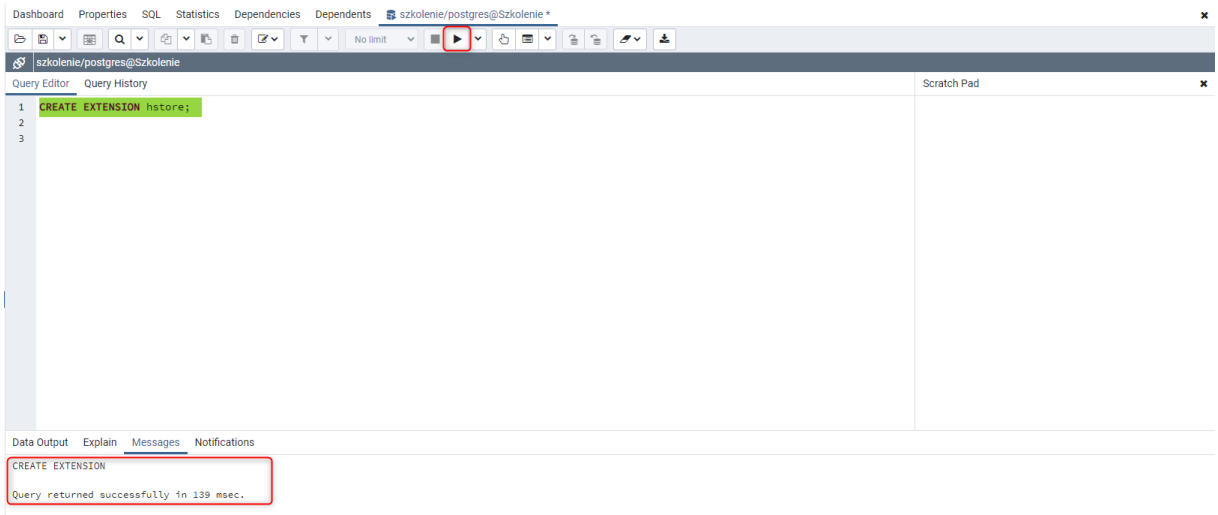
7. Aby potwierdzić tę zasadę kolejne rozszerzenia zainstalujemy już za pomocą poleceń SQL. W tym celu z menu programu wybieramy tools > query tool.



8. W oknie zapytań wpisujemy komendy instalujące kolejne rozszerzenie.

```
CREATE EXTENSION hstore;
```

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



9. 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.

5. Import zbiorów danych

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\14\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.

Ćwiczenie 3. Uzyskanie informacji o pliku shp do importu

Do importu użyjemy pliku .shp z nazwami miejscowości (*PRNG_MIEJSCOWOSCI_SHP.shp*) pochodzącej zdanych PRNG (Państwo Rejestr Nazw Geograficznych), dostępnych w folderze z danymi szkolenia.

1. 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 3.22.10/bin/ogrinfo.exe"
```

```
ogrinfo [--help-general] [-ro] [-q] [-where restricted_where|@filename]
        [-spat xmin ymin xmax ymax] [-geomfield field] [-fid fid]
        [-sql statement|@filename] [-dialect dialect] [-al] [-rl] [-so] [-
fields={YES/NO}]
        [-geom={YES/NO/SUMMARY/WKT/ISO_WKT}] [--formats] [[-oo NAME=VALUE] ...]
        [-nomd] [-listmdd] [-mdd domain|`all`]*
        [-nocount] [-noextent] [-nogeomtype] [-wkt_format WKT1|WKT2|...]
        [-fielddomain name]
        <datasource_name> [<layer> [<layer> ...]]
```

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

```
"c:/Program Files/QGIS 3.22.10/bin/ogrinfo.exe" PRNG_MIEJSCOWOSCI_SHP.shp
```

```
INFO: Open of `d:\sql_sz\dane\PRNG_MIEJSCOWOSCI_SHP.shp'
      using driver `ESRI Shapefile' successful.
1: PRNG_MIEJSCOWOSCI_SHP (Point)
```

Plik jest warstwą punktową *Point* o nazwie PRNG_MIEJSCOWOSCI_SHP.

3. Aby uzyskać więcej informacji o warstwie wpisujemy:

```
"c:/Program Files/QGIS 3.22.10/bin/ogrinfo.exe" -so PRNG_MIEJSCOWOSCI_SHP.shp
PRNG_MIEJSCOWOSCI_SHP
```

```
INFO: Open of `d:\sql_sz\dane\PRNG_MIEJSCOWOSCI_SHP.shp'
      using driver `ESRI Shapefile' successful.

Layer name: PRNG_MIEJSCOWOSCI_SHP
Metadata:
  DBF_DATE_LAST_UPDATE=2022-09-09
Geometry: Point
Feature Count: 789
Extent: (637831.560000, 552203.490000) - (697848.460000, 626664.070000)
Layer SRS WKT:
PROJCRS["ETRS_1989_Poland_CS92",
    BASEGEOGCRS["ETRF2000-PL",
        DATUM["D_ETRF2000_Poland",
            ELLIPSOID["GRS_1980",6378137,298.257222101,
                LENGTHUNIT["metre",1,
                    ID["EPSG",9001]]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["Degree",0.0174532925199433]]],
    CONVERSION["unnamed",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",19,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["Scale factor at natural origin",0.9993,
            SCALEUNIT["unity",1],
            ID["EPSG",8805]],
        PARAMETER["False easting",500000,
            LENGTHUNIT["Meter",1],
            ID["EPSG",8806]],
        PARAMETER["False northing",-5300000,
            LENGTHUNIT["Meter",1],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["Meter",1]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["Meter",1]]]
Data axis to CRS axis mapping: 1,2
idIIP: String (70.0)
idPRNG: Integer64 (11.0)
nazwaGlown: String (250.0)
elementRoz: String (250.0)
elementRod: String (50.0)
dopelniacz: String (50.0)
```

```
przymiot: String (250.0)
statusNazw: String (30.0)
kategoria: String (30.0)
rodzaj: String (50.0)
nazwaMscNd: String (250.0)
idMscNd: String (10.0)
informDod: String (250.0)
systemZewn: String (20.0)
idZewnetrz: String (50.0)
zrodloTyt: String (254.0)
zrodloData: String (254.0)
zrodloWyd: String (254.0)
rodzajRepr: String (200.0)
wspGeograf: String (200.0)
wspXY: String (200.0)
nazwaHist: String (254.0)
nazwaObocz: String (254.0)
nazwaDodat: String (254.0)
jezykDodat: String (254.0)
latynDodat: String (254.0)
endonim: String (254.0)
jezykEndon: String (254.0)
latynEndon: String (254.0)
egzonim: String (254.0)
jezykEgzon: String (254.0)
latynEgzon: String (254.0)
wojewodz: String (200.0)
powiat: String (200.0)
gmina: String (200.0)
idGminy: String (200.0)
poczWerOb: String (21.0)
wersjaOb: String (21.0)
waznaOd: String (21.0)
```

Z otrzymanej informacji zapamiętujemy liczbę obiektów, czyli **789**.

Ćwiczenie 4. Import danych wektorowych za pomocą shp2pgsql

1. Opcje programu **shp2pgsql** możemy otrzymać wpisując w wierszu poleceń:

```
"C:/Program Files/PostgreSQL/14/bin/shp2pgsql.exe"
```

```
RELEASE: 3.2.3 (3.2.3)
USAGE: shp2pgsql [<options>] <shapefile> [[<schema>.]<table>]
OPTIONS:
  -s [<from>:]<srid> Set the SRID field. Defaults to 0.
      Optionally reprojects from given SRID.
 (-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.
-Z  Prevent tables from being analyzed.
-?  Display this help screen.

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

2. Dane importujemy do bazy za pomocą polecenia:

```
"C:/Program Files/PostgreSQL/14/bin/shp2pgsql.exe" -s 2180 -c -g geom -I -S
PRNG_MIEJSCOWOSCI_SHP.shp public.miejscowosci > miejscowosci.sql
```

Parametry:

"C:/Program Files/PostgreSQL/14/bin/shp2pgsql.exe" = ścieżka do programu shp2pgsql

-s 2180 0 układ współrzędnych PL-1992

-c tworzy nową tabelę w bazie

-g way nazwa pola z geometrią

-I tworzy indeks przestrzenny

-S wymusza tworzenie geometrii prostych (simple) zamiasta złożonych (multi)

- **PRNG_MIEJSCOWOSCI_SHP.shp** nazwa importowanego pliku

- **public.miejscowosci** schemat i tabela, do której będą zaimportowane dane
- **miejscowości.sql** przekierowanie wyjścia aplikacji do pliku o tej nazwie

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

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

W lokalizacji pliku z danymi został utworzony plik *miejscowosci.sql*, zawierający skrypt sql.

4. Plik ten zostanie zaimportowany do bazy za pomocą aplikacji **psql**.

```
"C:/Program Files/PostgreSQL/14/bin/psql.exe" -f miejscowosci.sql -U postgres szkolenie
```

Parametry:
"C:/Program Files/PostgreSQL/14/bin/psql.exe" – ścieżka do aplikacji psql.exe
-f miejscowości.sql – plik SQL do uruchomienia
-U postgres – nazwa użytkownika bazy danych
szkolenia – nazwa bazy

5. Aplikacja poprosi o podanie hasła użytkownika *postgres*, po czym uruchomi skrypt. Rozpocznie się proces ładowania danych, który powinien wyświetlać następujące komunikaty:

```
SET
SET
BEGIN
CREATE TABLE
ALTER TABLE
                  addgeometrycolumn
------------------------------------------------------
 public.miejscowosci.way SRID:2180 TYPE:POINT DIMS:2
(1 row)

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

6. Poprawność importu można sprawdzić w aplikacji **pgAdmin**. W bazie danych *szkolenie*, w schemacie *public* powinna się pojawić tabela *miejscowosci*, zawierająca **789** obiektów co jest wartością zgodną z informacjami z aplikacji **ogrinfo**.

Ćwiczenie 5. Import danych wektorowych za pomocą ogr2ogr

Aplikacja **ogr2ogr** to rozbudowane narzędzie pozwalające na konwersję danych przestrzennych między kilkudziesięcioma formatami danych, jak i na wykonywanie na nich innych operacji, takich jak filtrowanie, zmiana kodowania zmiana układu współrzędnych itp. Za pomocą tej aplikacji zostanie zaimportowana do bazy warstwa gmin.

1. Za pomocą poznanej już aplikacji **ogrinfo** sprawdzimy ilość obiektów w warstwie gmin:

```
"c:/Program Files/QGIS 3.22.10/bin/ogrinfo.exe" -so A03_Granice_gmin.shp A03_Granice_gmin
```

```
INFO: Open of `A03_Granice_gmin.shp'
      using driver `ESRI Shapefile' successful.
Layer name: A03_Granice_gmin
Metadata:
  DBF_DATE_LAST_UPDATE=2022-09-06
Geometry: Polygon
Feature Count: 314
Extent: (19.259214, 51.013112) - (23.128409, 53.481806)
ERROR 1: PROJ: proj_identify: C:\Program Files\PostgreSQL\14\share\contrib\postgis-
3.2\proj\proj.db contains DATABASE.LAYOUT.VERSION.MINOR = 0 whereas a number >= 2
is expected. It comes from another PROJ installation.
Layer SRS WKT:
GEOGCRS["ETRS89",
    DATUM["European Terrestrial Reference System 1989",
        ELLIPSOID["GRS_1980",6378137,298.257222101,
            LENGTHUNIT["metre",1]],
        ID["EPSG",6258]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["Degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["longitude",east,
            ORDER[1],
            ANGLEUNIT["Degree",0.0174532925199433]],
        AXIS["latitude",north,
            ORDER[2],
            ANGLEUNIT["Degree",0.0174532925199433]]]
Data axis to CRS axis mapping: 1,2
Shape_Leng: Real (19.11)
Shape_Area: Real (19.11)
JPT_SJR_KO: String (3.0)
JPT_POWIER: Real (19.11)
JPT_KOD_JE: String (20.0)
JPT_NAZWA_: String (128.0)
JPT_ORGAN_: String (254.0)
```

```
JPT_JOR_ID: Integer (9.0)
WERSJA_OD: Date (10.0)
WERSJA_DO: Date (10.0)
WAZNY_OD: Date (10.0)
WAZNY_DO: Date (10.0)
JPT_KOD__1: String (3.0)
JPT_NAZWA1: String (3.0)
JPT_ORGAN1: String (3.0)
JPT_WAZNA_: String (3.0)
ID_BUFORA_: Real (19.11)
ID_BUFORA1: Real (19.11)
ID_TECHNIC: Integer (9.0)
IIP_PRZEST: String (20.0)
IIP_IDENTY: String (128.0)
IIP_WERSJA: String (32.0)
JPT_KJ_IIP: String (20.0)
JPT_KJ_I_1: String (128.0)
JPT_KJ_I_2: String (32.0)
JPT_OPIS: String (254.0)
JPT_SPS_KO: String (3.0)
ID_BUFOR_1: Integer (9.0)
JPT_ID: Integer (9.0)
JPT_POWI_1: Real (19.11)
JPT_KJ_I_3: String (3.0)
JPT_GEOMET: Real (19.11)
JPT_GEOM_1: Real (19.11)
Shape_Le_1: Real (19.11)
REGON: String (254.0)
```

Takich obiektów jest **314**.

2. Aby zapoznać się z parametrami aplikacji **ogr2ogr** należy uruchomić polecenie:

```
"c:/Program Files/QGIS 3.22.10/bin/ogr2ogr.exe"
```

```
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update]
               [-select field_list] [-where restricted_where|@filename]
               [-progress] [-sql <sql statement>|@filename] [-dialect dialect]
               [-preserve_fid] [-fid FID] [-limit nb_features]
               [-spat xmin ymin xmax ymax] [-spat_srs srs_def] [-geomfield field]
               [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def] [-ct string]
               [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
               dst_datasource_name src_datasource_name
               [-lco NAME=VALUE] [-nln name]
               [-nlt type|PROMOTE_TO_MULTI|CONVERT_TO_LINEAR|CONVERT_TO_CURVE]
               [-dim XY|XYZ|XYM|XYZM|layer_dim] [layer [layer ...]]

Advanced options :
               [-gt n] [-ds_transaction]
               [[-oo NAME=VALUE] ...] [[-doo NAME=VALUE] ...]
               [-clipsrc [xmin ymin xmax ymax]|WKT|datasource|spat_extent]
               [-clipsrcsql sql_statement] [-clipsrclayer layer]
               [-clipsrcwhere expression]
```

```
                [-clipdst [xmin ymin xmax ymax]|WKT|datasource]
                [-clipdstsql sql_statement] [-clipdstlayer layer]
                [-clipdstwhere expression]
                [-wrapdateline][-datelineoffset val]
                [[-simplify tolerance] | [-segmentize max_dist]]
                [-makevalid]
                [-addfields] [-unsetFid] [-emptyStrAsNull]
                [-relaxedFieldNameMatch] [-forceNullable] [-unsetDefault]
                [-fieldTypeToString All|(type1[,type2]*)] [-unsetFieldWidth]
                [-mapFieldType srctype|All=dsttype[,srctype2=dsttype2]*]
                [-fieldmap identity | index1[,index2]*]
                [-splitlistfields] [-maxsubfields val]
                [-resolveDomains]
                [-explodecollections] [-zfield field_name]
                [-gcp ungeoref_x ungeoref_y georef_x georef_y [elevation]]* [-order
n | -tps]
                [[-s_coord_epoch epoch] | [-t_coord_epoch epoch] | [-a_coord_epoch
epoch]]
                [-nomd] [-mo "META-TAG=VALUE"]* [-noNativeData]
Note: ogr2ogr --long-usage for full help.
```

3. Aby zaimportować warstwę gmin do bazy za pomocą aplikacji **ogr2ogr** wykonamy polecenie:

```
"c:/Program Files/QGIS 3.22.10/bin/ogr2ogr.exe” -f "PostgreSQL" "PG:host=localhost user=postgres dbname=szkolenie password=gis" "A03_Granice_gmin.shp" -lco GEOMETRY_NAME=geom -nln gminy -nlt POLYGON -s_srs EPSG:4258 -t_srs  EPSG:2180  -overwrite
```

Jeżeli polecenie zostanie wykonane prawidłowo na konsoli nie zostanie wyświetlony żaden komunikat.

4. Aby sprawdzić poprawność importu do bazy danych skorzystamy tym razem z aplikacji **psql** wpisując polecenie:

```
"C:/Program Files/PostgreSQL/14/bin/psql.exe" -U postgres -c "select count(*) from gminy ;" szkolenie
```

Po podaniu hasła do bazy wyświetli się liczba obiektów w utworzonej tabeli gminy:

```
count
-------
   314
(1 row)
```

Jak widać liczba obiektów zgadza się z informacją odczytaną za pomocą aplikacji **ogrinfo**.

Ćwiczenie 6. Eksport danych wektorowych za pomocą pgsql2shp

pgsql2shp jest aplikacją o działaniu odwrotnym do poznanej wcześniej aplikacji **shp2pgsql**, czyli pozwala ona na eksport danych z bazy danych do pliku w formacie *SHP*.

1. W celu odczytu dostępnych parametrów aplikacji **pgsql2shp** należy wpisać polecenie:

```
"C:/Program Files/PostgreSQL/14/bin/pgsql2shp.exe"
```

```
RELEASE: 3.2.3 (3.2.3)
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
  -q Quiet mode. No messages to stdout.
         -? Display this help screen.
```

2. W celu eksportu dowolnej warstwy wektorowej do pliku SHP należy użyć polecenia:

```
"C:/Program Files/PostgreSQL/14/bin/pgsql2shp.exe" -f gminy.shp -h localhost -p 5432 -u postgres -P
gis szkolenie public.gminy
```

Parametry:
"C:/Program Files/PostgreSQL/14/bin/pgsql2shp.exe" ścieżka do aplikacji **pgsql2shp**
-f gminy.shp nazwa wynikowego pliku SHP
-h localhost nazwa serwera bazy danych
-p 5432 port nasłuchu bazy
-u postgres nazwa użytkownika

-P gis hasło użytkownika
szkolenie nazwa bazy danych
public.gminy nazwa schematu i nazwa tabeli, która jest eksportowana

3. Po wykonanym eksporcie w oknie poleceń pojawi się komunikat

```
Initializing...
Done (postgis major version: 3).
Output shape: Polygon
Dumping: XXXX [314 rows].
```

W komunikacie podana jest informacja o typie geometrii (Polygon) oraz liczbie wyeksportowanych obiektów (314).

4. Poprawność eksportu można wykonać za pomocą aplikacji **ogrinfo**:

```
"c:/Program Files/QGIS 3.22.10/bin/ogrinfo.exe" -so gminy.shp gminy
```

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

Layer name: gminy
Geometry: Polygon
Feature Count: 314
Extent: (517613.878020, 352474.636502) - (781450.361861, 627243.164042)
Layer SRS WKT:
BOUNDCRS[
    SOURCECRS[
        PROJCRS["ETRS89 / Poland CS92",
            BASEGEOGCRS["ETRS89",
                DATUM["European Terrestrial Reference System 1989",
                    ELLIPSOID["GRS 1980",6378137,298.257222101,
                        LENGTHUNIT["metre",1]]],
                PRIMEM["Greenwich",0,
                    ANGLEUNIT["degree",0.0174532925199433]],
                ID["EPSG",4258]],
            CONVERSION["unnamed",
                METHOD["Transverse Mercator",
                    ID["EPSG",9807]],
                PARAMETER["Latitude of natural origin",0,
                    ANGLEUNIT["degree",0.0174532925199433],
                    ID["EPSG",8801]],
                PARAMETER["Longitude of natural origin",19,
                    ANGLEUNIT["degree",0.0174532925199433],
                    ID["EPSG",8802]],
                PARAMETER["Scale factor at natural origin",0.9993,
                    SCALEUNIT["unity",1],
                    ID["EPSG",8805]],
                PARAMETER["False easting",500000,
                    LENGTHUNIT["metre",1],
                    ID["EPSG",8806]],
                PARAMETER["False northing",-5300000,
                    LENGTHUNIT["metre",1],
```

```
                ID["EPSG",8807]]],
            CS[Cartesian,2],
                AXIS["(E)",east,
                    ORDER[1],
                    LENGTHUNIT["metre",1]],
                AXIS["(N)",north,
                    ORDER[2],
                    LENGTHUNIT["metre",1]],
            ID["EPSG",2180]]],
    TARGETCRS[
        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]]],
    ABRIDGEDTRANSFORMATION["Transformation from ETRS89 to WGS84",
        METHOD["Position Vector transformation (geog2D domain)",
            ID["EPSG",9606]],
        PARAMETER["X-axis translation",0,
            ID["EPSG",8605]],
        PARAMETER["Y-axis translation",0,
            ID["EPSG",8606]],
        PARAMETER["Z-axis translation",0,
            ID["EPSG",8607]],
        PARAMETER["X-axis rotation",0,
            ID["EPSG",8608]],
        PARAMETER["Y-axis rotation",0,
            ID["EPSG",8609]],
        PARAMETER["Z-axis rotation",0,
            ID["EPSG",8610]],
        PARAMETER["Scale difference",1,
            ID["EPSG",8611]]]]
Data axis to CRS axis mapping: 1,2
OGC_FID: Integer64 (11.0)
SHAPE_LENG: Real (32.10)
SHAPE_AREA: Real (32.10)
JPT_SJR_KO: String (3.0)
JPT_POWIER: Real (32.10)
JPT_KOD_JE: String (20.0)
JPT_NAZWA_: String (128.0)
JPT_ORGAN_: String (254.0)
JPT_JOR_ID: Real (32.10)
WERSJA_OD: Date (10.0)
WERSJA_DO: Date (10.0)
WAZNY_OD: Date (10.0)
WAZNY_DO: Date (10.0)
JPT_KOD__1: String (3.0)
JPT_NAZWA1: String (3.0)
JPT_ORGAN1: String (3.0)
JPT_WAZNA_: String (3.0)
```

```
ID_BUFORA_: Real (32.10)
ID_BUFORA1: Real (32.10)
ID_TECHNIC: Real (32.10)
IIP_PRZEST: String (20.0)
IIP_IDENTY: String (128.0)
IIP_WERSJA: String (32.0)
JPT_KJ_IIP: String (20.0)
JPT_KJ_I_1: String (128.0)
JPT_KJ_I_2: String (32.0)
JPT_OPIS: String (254.0)
JPT_SPS_KO: String (3.0)
ID_BUFOR_1: Real (32.10)
JPT_ID: Real (32.10)
JPT_POWI_1: Real (32.10)
JPT_KJ_I_3: String (3.0)
JPT_GEOMET: Real (32.10)
JPT_GEOM_1: Real (32.10)
SHAPE_LE_1: Real (32.10)
REGON: String (254.0)
```

Ćwiczenie 7. Eksport danych wektorowych za pomocą ogr2ogr

Poznana wcześniej aplikacja **ogr2ogr** umożliwia nie tylko import obiektów z plików w wielu formatach, ale również ich eksport, działa więc dwukierunkowo.

1. Aby wyeksportować tabelę z bazy danych do formatu wektorowego json za pomocą aplikacji ogr2ogr należy użyć polecenia:

```
"c:/Program Files/QGIS 3.22.10/bin/ogr2ogr.exe" -f "GeoJSON" gminy.json "PG:host=localhost user=postgres dbname=szkolenie password=gis" public.gminy
```

2. Jeżeli eksport się powiódł w oknie poleceń nie pojawi się żaden komunikat, natomiast w docelowym katalogu pojawi się plik *gminy.json*
3. Ilość obiektów w pliku *json* sprawdzimy za pomocą aplikacji **ogrinfo**:

```
"c:/Program Files/QGIS 3.22.10/bin/ogrinfo.exe" -so gminy.json gminy
```

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

Layer name: gminy
Geometry: Polygon
Feature Count: 314
Extent: (517613.878020, 352474.636502) - (781450.361861, 627243.164042)
Layer SRS WKT:
```

```
PROJCRS["ETRF2000-PL / CS92",
    BASEGEOGCRS["ETRF2000-PL",
        DATUM["ETRF2000 Poland",
            ELLIPSOID["GRS 1980",6378137,298.257222101,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",9702]],
    CONVERSION["Poland CS92",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",19,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["Scale factor at natural origin",0.9993,
            SCALEUNIT["unity",1],
            ID["EPSG",8805]],
        PARAMETER["False easting",500000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8806]],
        PARAMETER["False northing",-5300000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["northing (x)",north,
            ORDER[1],
            LENGTHUNIT["metre",1]],
        AXIS["easting (y)",east,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    USAGE[
        SCOPE["Topographic mapping (medium and small scale)."],
        AREA["Poland - onshore and offshore."],
        BBOX[49,14.14,55.93,24.15]],
    ID["EPSG",2180]]
Data axis to CRS axis mapping: 2,1
shape_leng: Real (0.0)
shape_area: Real (0.0)
jpt_sjr_ko: String (0.0)
jpt_powier: Real (0.0)
jpt_kod_je: String (0.0)
jpt_nazwa_: String (0.0)
jpt_organ_: String (0.0)
jpt_jor_id: Integer (0.0)
wersja_od: Date (0.0)
wersja_do: String (0.0)
wazny_od: Date (0.0)
wazny_do: String (0.0)
jpt_kod__1: String (0.0)
jpt_nazwa1: String (0.0)
jpt_organ1: String (0.0)
jpt_wazna_: String (0.0)
id_bufora_: Real (0.0)
id_bufora1: Real (0.0)
id_technic: Integer (0.0)
iip_przest: String (0.0)
```

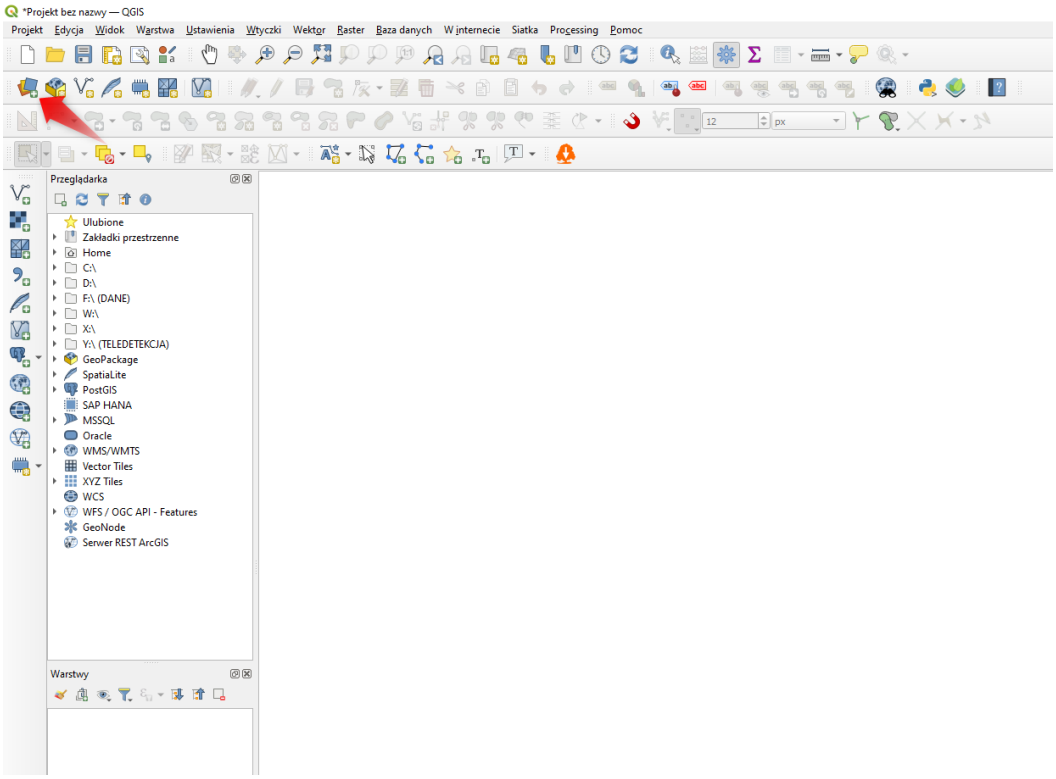
```
iip_identy: String (0.0)
iip_wersja: DateTime (0.0)
jpt_kj_iip: String (0.0)
jpt_kj_i_1: String (0.0)
jpt_kj_i_2: String (0.0)
jpt_opis: String (0.0)
jpt_sps_ko: String (0.0)
id_bufor_1: Integer (0.0)
jpt_id: Integer (0.0)
jpt_powi_1: Real (0.0)
jpt_kj_i_3: String (0.0)
jpt_geomet: Real (0.0)
jpt_geom_1: Real (0.0)
shape_le_1: Real (0.0)
regon: String (0.0)
```

Ponieważ eksportowaliśmy tą samą tabelę co poprzednio i tym razem ilość wyeksportowanych obiektów wynosi **314**.

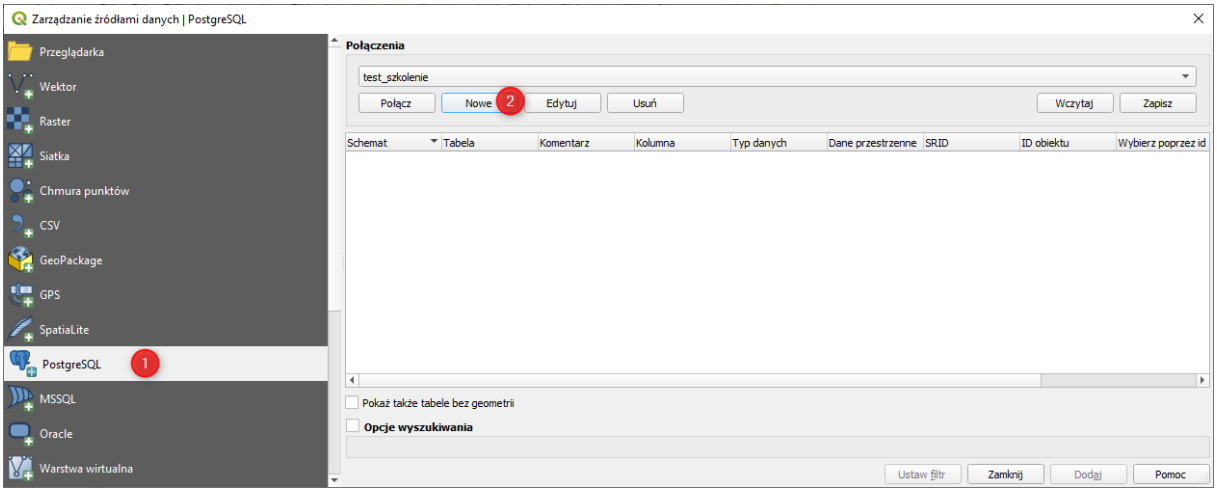
6. Obsługa bazy danych PostgreSQL za pomocą aplikacji QGIS

Ćwiczenie 8. Utworzenie połączenia z bazą danych

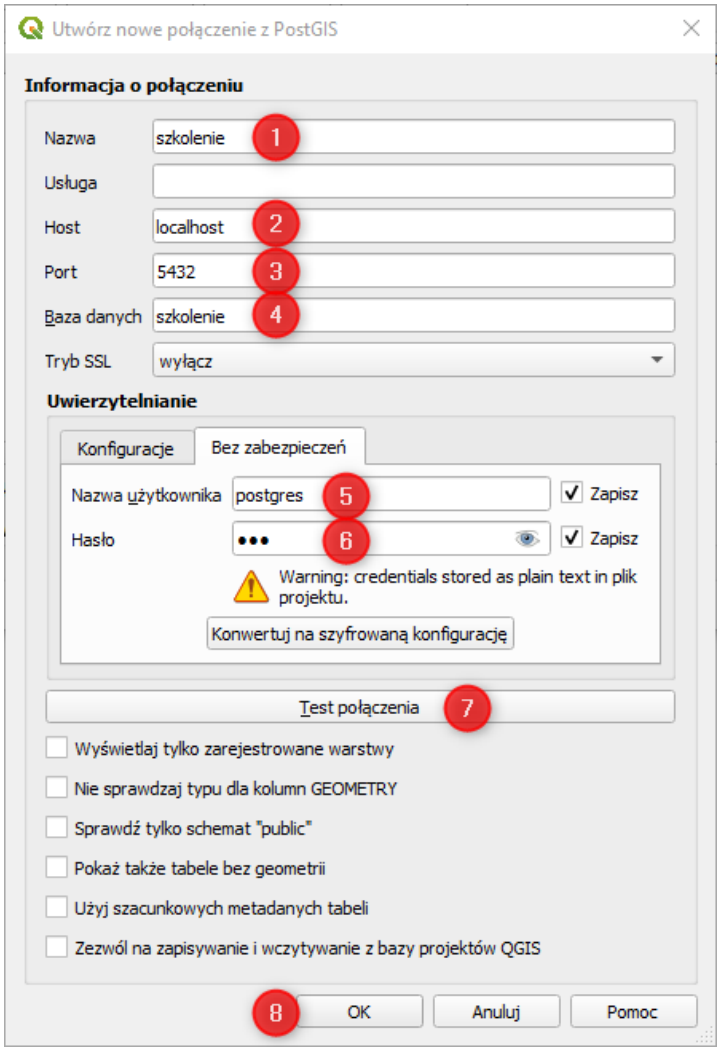
1. Jeżeli W celu utworzenia nowego połączenia z bazą danych PostgreSQL otwieramy okno **Zarządzanie źródłami danych** z menu **Warstwa/ Zarządzanie źródłami danych** lub za pomocą ikonki 



2. W oknie zarządzania źródłami danych wybieramy **PostgreSQL** i klikamy **Nowe.**



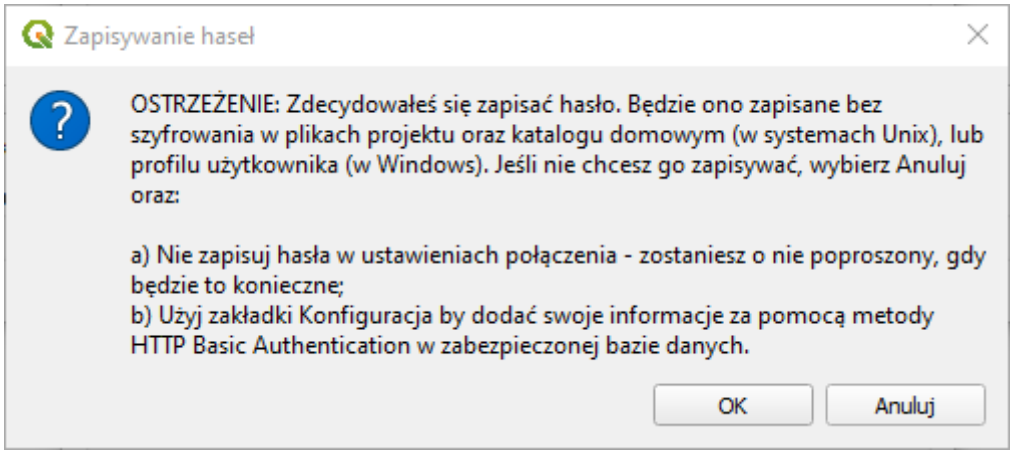
3. W oknie konfiguracji połączenia uzupełniamy kolejno dane:



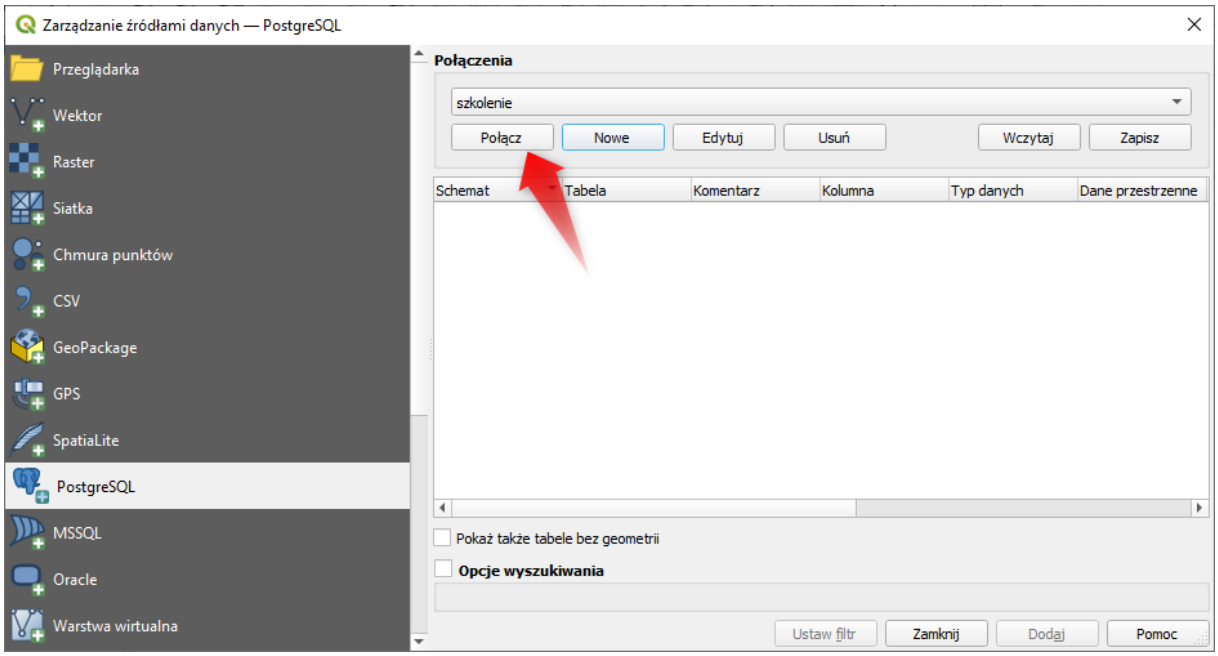
Jeżeli test połączenia przebiegł pomyślnie, zatwierdzamy definicję połączenia za pomocą przycisku **OK**.

Zapisanie nazwy użytkownika i hasła ułatwia pracę z bazą, ale należy pamiętać, że parametry te zapisywane są w plikach projektu QGIS jawnie i można je podejrzeć. Trzeba więc mieć to na uwadze decydując się na ten krok.

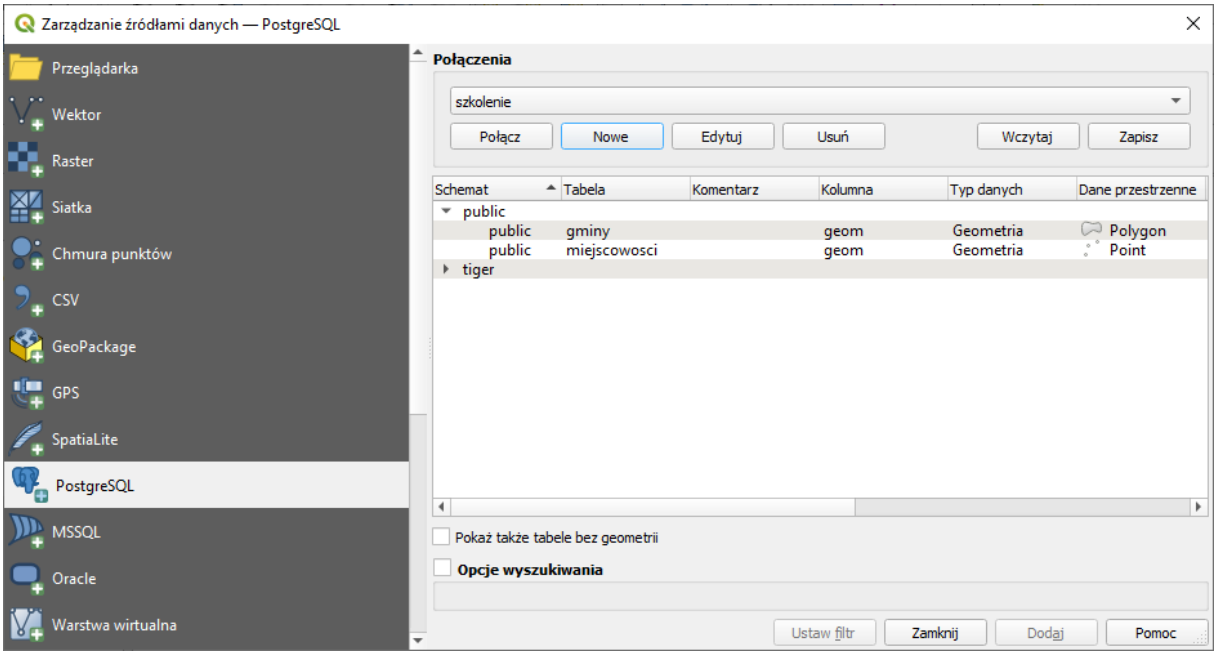
Świadczy o tym komunikat, który pojawi się w takim przypadku:



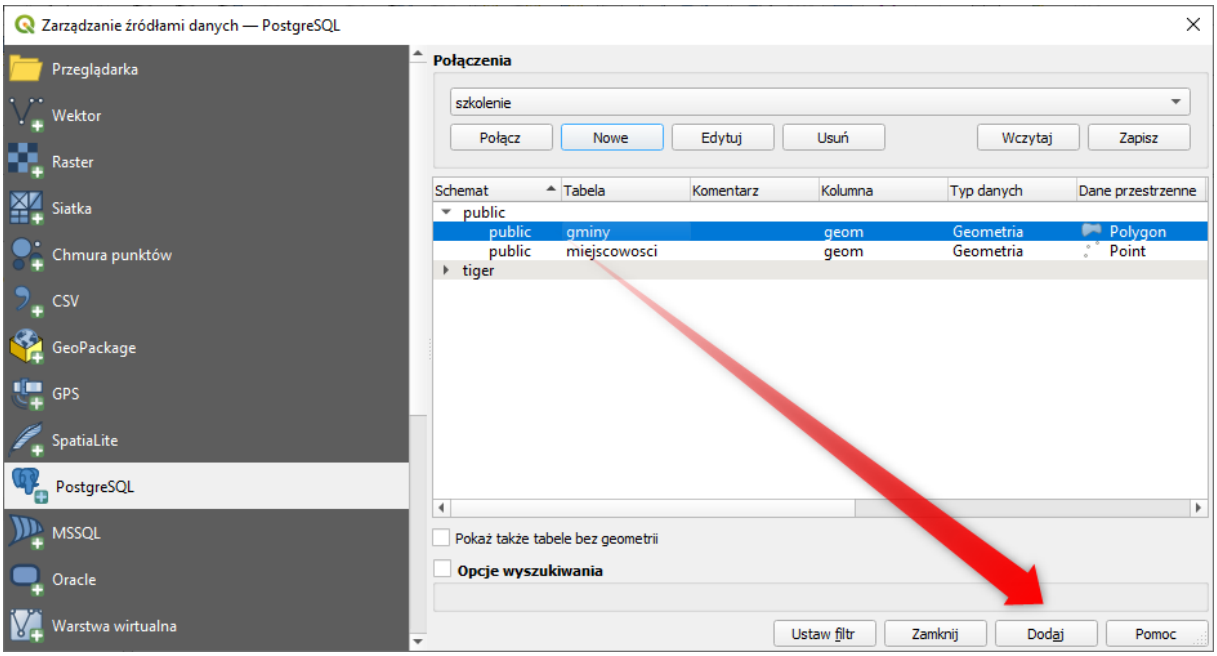
4. W oknie zarządzania źródłami danych odnajdujemy nasze połączenie i klikamy **Połącz**.

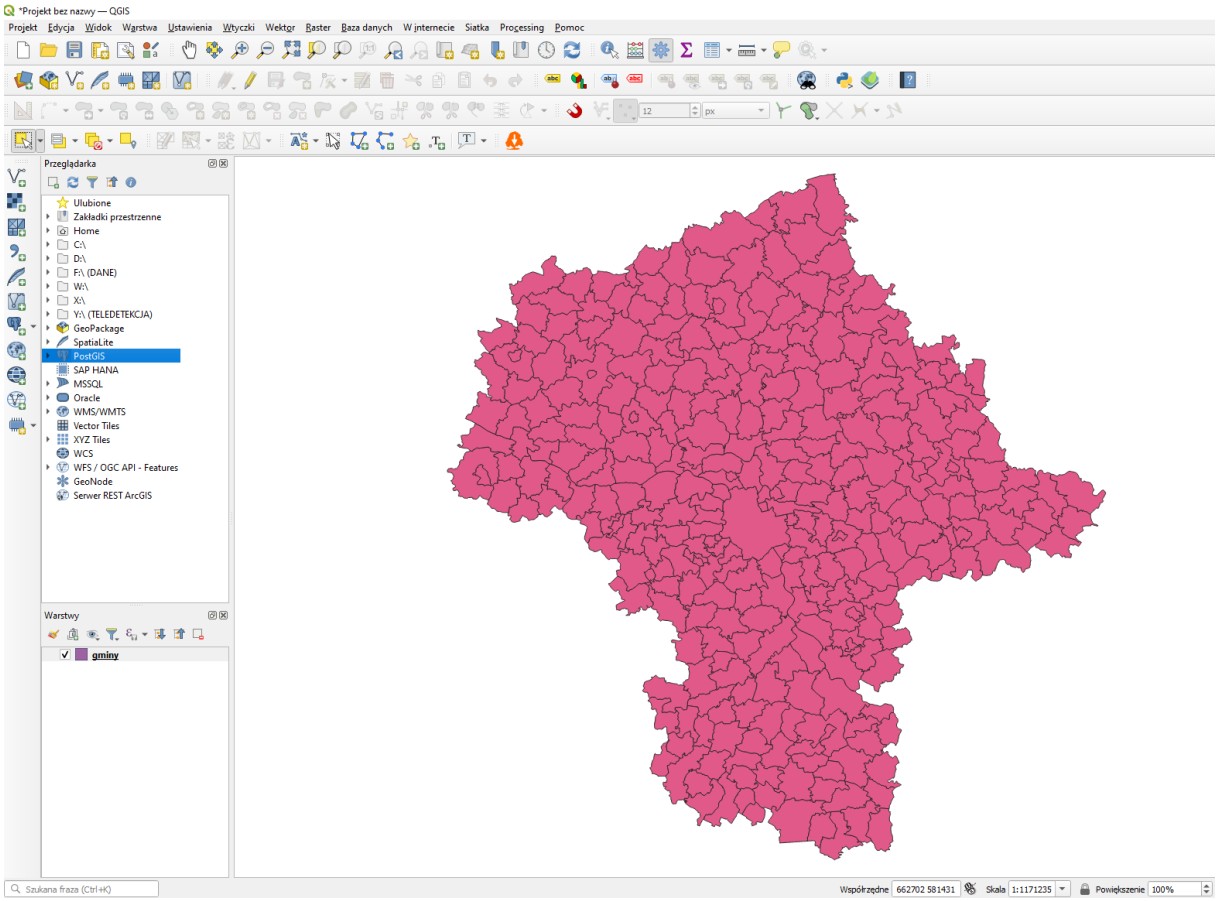


5. W oknie zawartości bazy danych pojawią się informacje o schematach i tabelach w bazie danych. Rozwińmy zawartość schematu **public**, aby zobaczyć tabele w tym schemacie.



6. Zaznaczenie jednej lub więcej tabeli w bazie i naciśnięcie przycisku Dodaj spowoduje wczytanie danych do drzewa warstw w aplikacji QGIS.

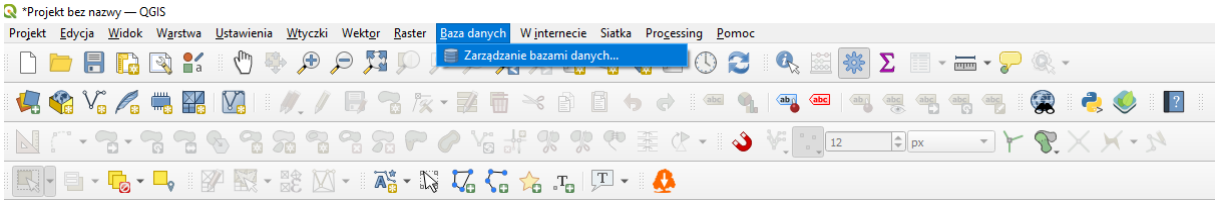




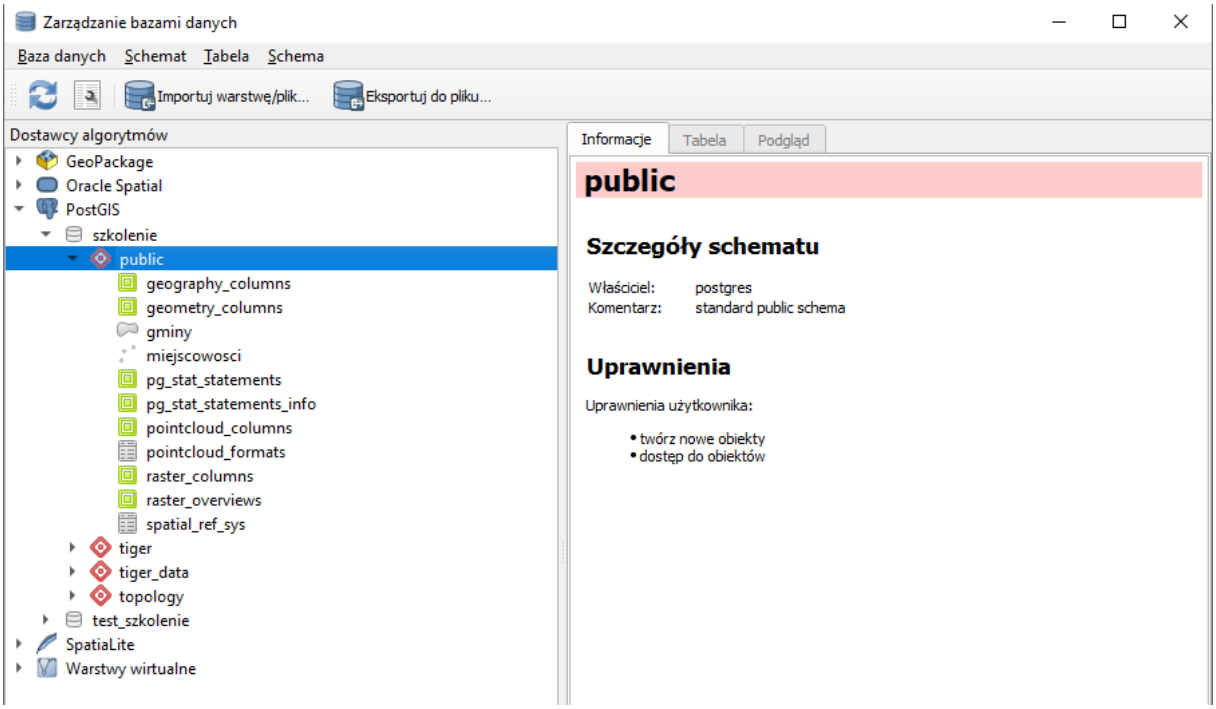
Ćwiczenie 9. Utworzenie nowego schematu i tabeli w bazie PostgreSQL

Zarządzanie bazą PostgreSQL w QGIS odbywa się za pomocą narzędzia **Zarządzanie bazami danych**.

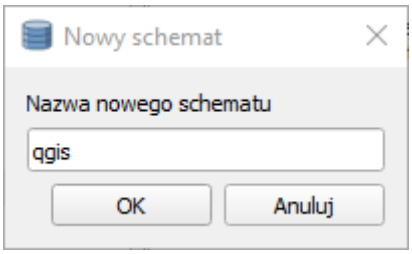
1. Uruchamiamy okno zarządzania bazami danych klikając w menu **Baza danych/ Zarządzanie bazami danych.**



2. W oknie zarządzania bazami danych rozwijamy kolejno PostGIS > szkolenie > public.

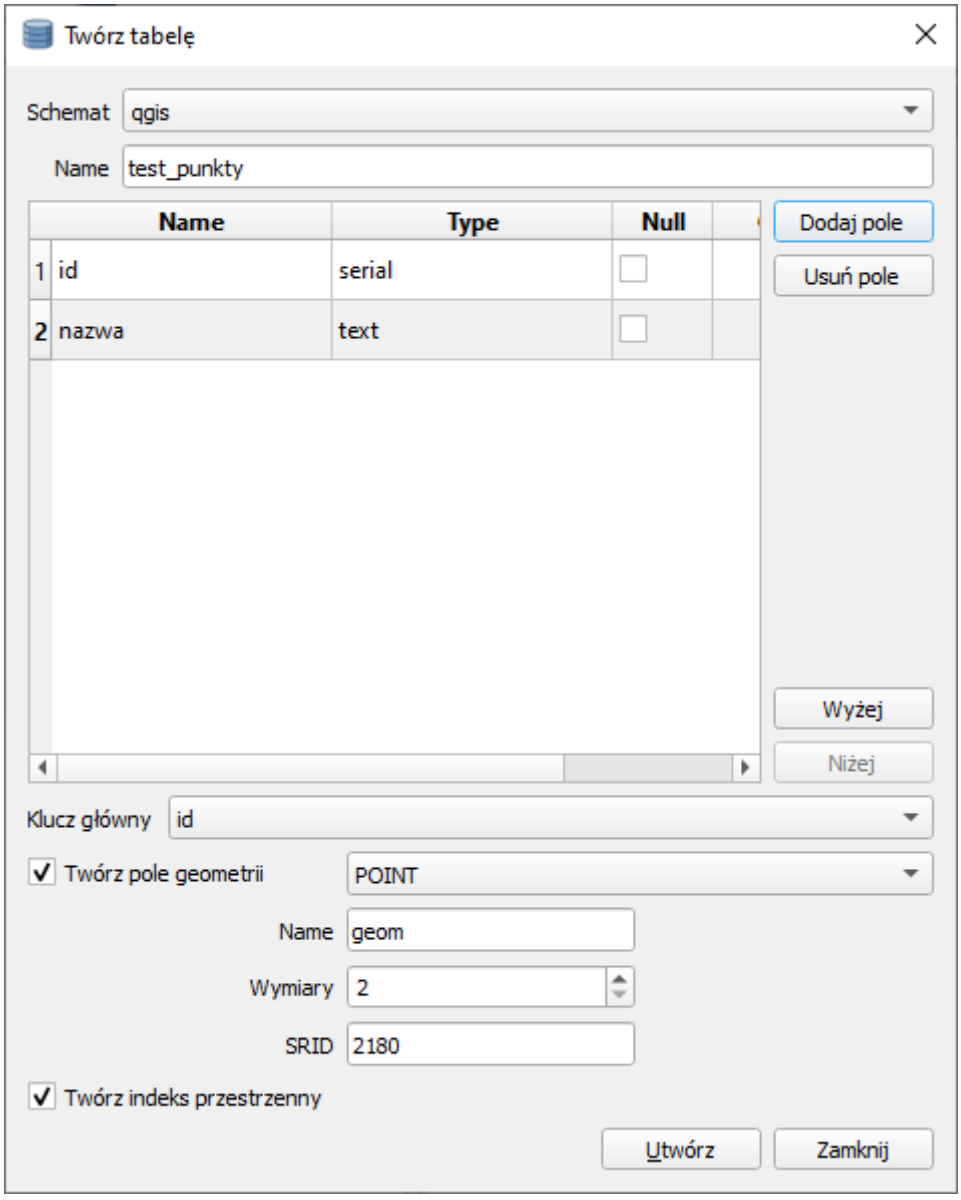


3. Aby utworzyć nowy schemat w bazie danych wybieramy w menu Schemat/Utwórz schemat oraz uzupełniamy nazwę schematu **qgis** i naciskamy **OK**:



4. W calu utworzenia tabeli zaznaczamy w drzewie Dostawcy algorytmów nowy schemat **qgis** oraz wybieramy w menu **Tabela/Utwórz tabelę**.

5. W oknie dodawania tabeli wykonujemy następujące 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 2180
- zaznaczamy **Twórz indeks przestrzenny**

Kilka słów wyjaśnienia:

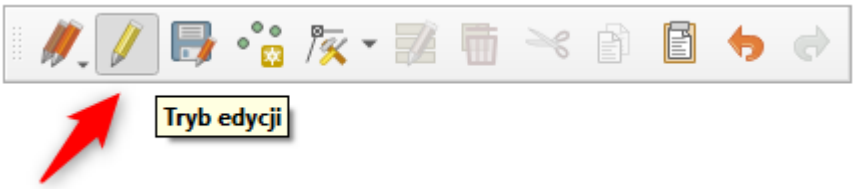
- typ danych *serial* to typ specyficzny dla PostgreSQL - jest oparty na generatorze - dzięki niemu możemy automatycznie i unikalnie numerować rekordy w tabeli
- klucz główny tabeli musi być ustawiony jawnie jeśli dane w tej tabeli mają być edytowane w QGIS
- indeks przestrzenny pozwalana na skrócenie zapytań opartych i parametry przestrzenne.

Po uzupełnieniu wszystkich danych zatwierdzamy wszystko przyciskiem **Utwórz**. Nowa tabela powinna pojawić się w drzewie obiektów, a po dwukliku powinna zostać dodana do warstw projektu.

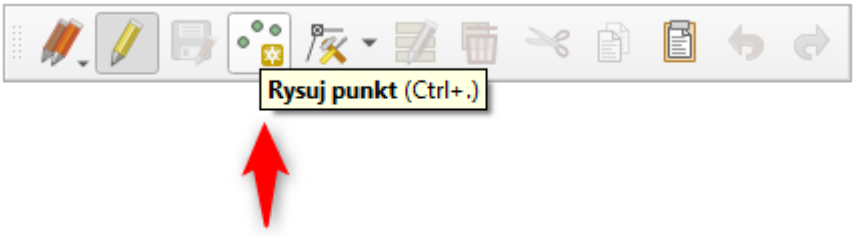
Ćwiczenie 10. Edycja danych w bazie PostgreSQL za pomocą QGIS

Edycja warstwy wektorowej pochodzącej z bazy PostgreSQL jest identyczna jak edycja każdej innej warstwy wektorowe w QGIS.

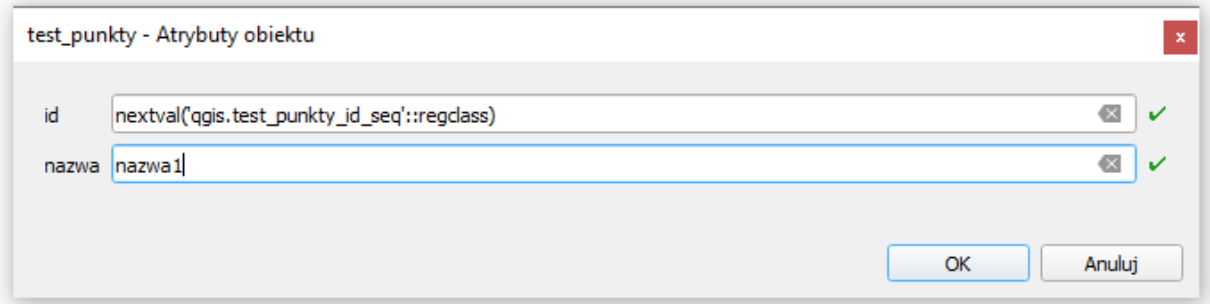
1. Zaznaczamy w drzewie warstw warstwę do edycji: **test_punkty**, a następnie w pasku edycji przełączamy warstwę w tryb edycji za pomocą przycisku **Tryb edycji**.



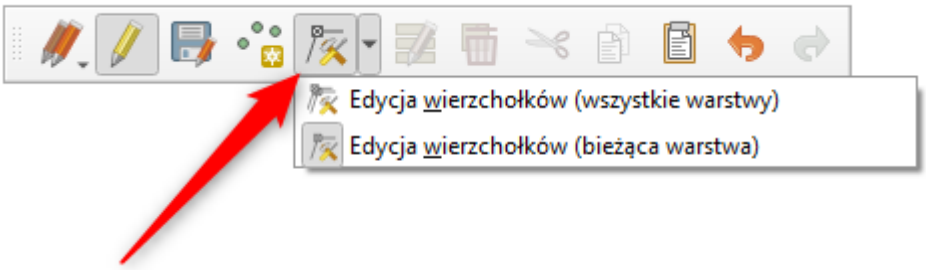
2. Wybieramy narzędzie **Rysuj punkt**.



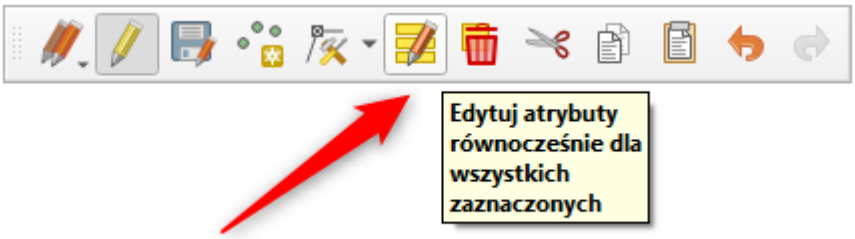
3. Po wrysowaniu punktu na mapie pojawi się okno do uzupełnienia atrybutów obiektu. Ponieważ atrybut id jest typem serial, wartość tego atrybutu obliczy się automatycznie. Uzupełniamy atrybut nazwa i naciskamy **OK**.

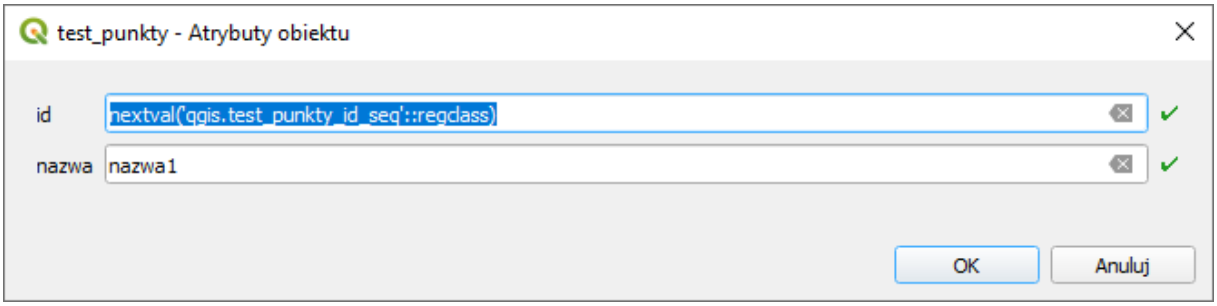


4. Narzędzie **Edycja wierzchołków** pozwala na modyfikację geometrii wierzchołków obiektu. Klikamy na dowolny wierzchołek (w przypadku punktu na ten punkt) i przesuwamy go do innej lokalizacji.

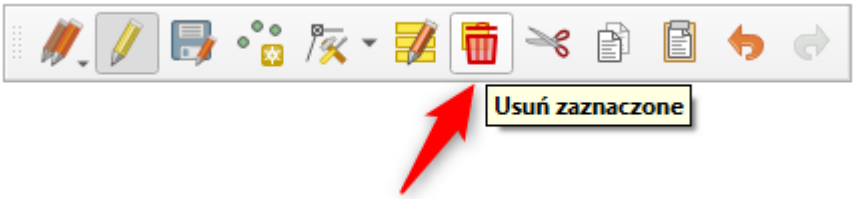


5. Jeżeli obiekt lub obiekty zostały uprzednio wybrane w narzędziach selekcji, istnieje możliwość edycji atrybutów tych obiektów za pomocą narzędzia **Edytuj atrybuty równocześnie dla wszystkich zaznaczonych**.



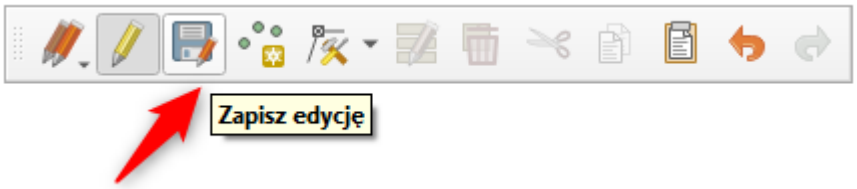


6. Usunięcie zaznaczonych obiektów odbywa się z kolei za pomocą narzędzia Usuń zaznaczone.

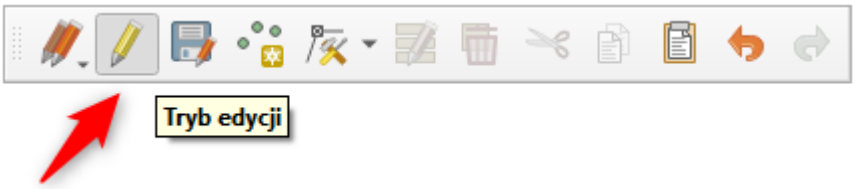


7. Kolejne narzędzia, czyli **Wytnij**, **Kopiuj**, **Wklej**, **Cofnij** oraz **Powtórz** działają analogicznie jak w innych aplikacjach.

8. Aby zapisać zmiany należy użyć przycisku **Zapisz Edycję**.



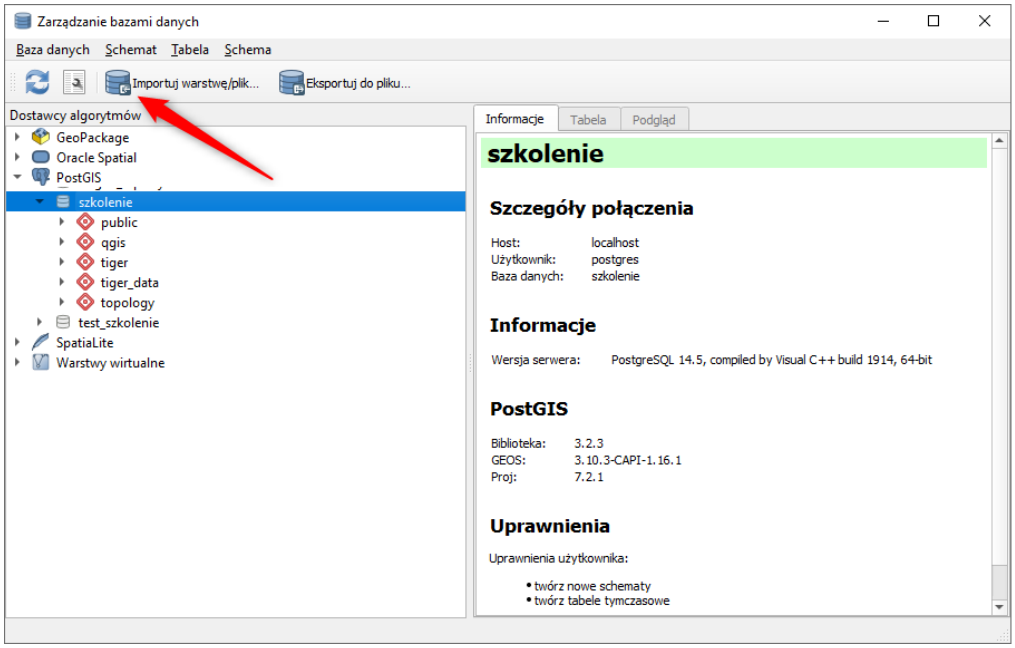
9. Wyłączenie edycji odbywa się poprzez ponowne wciśnięcie przycisku **Tryb edycji**.



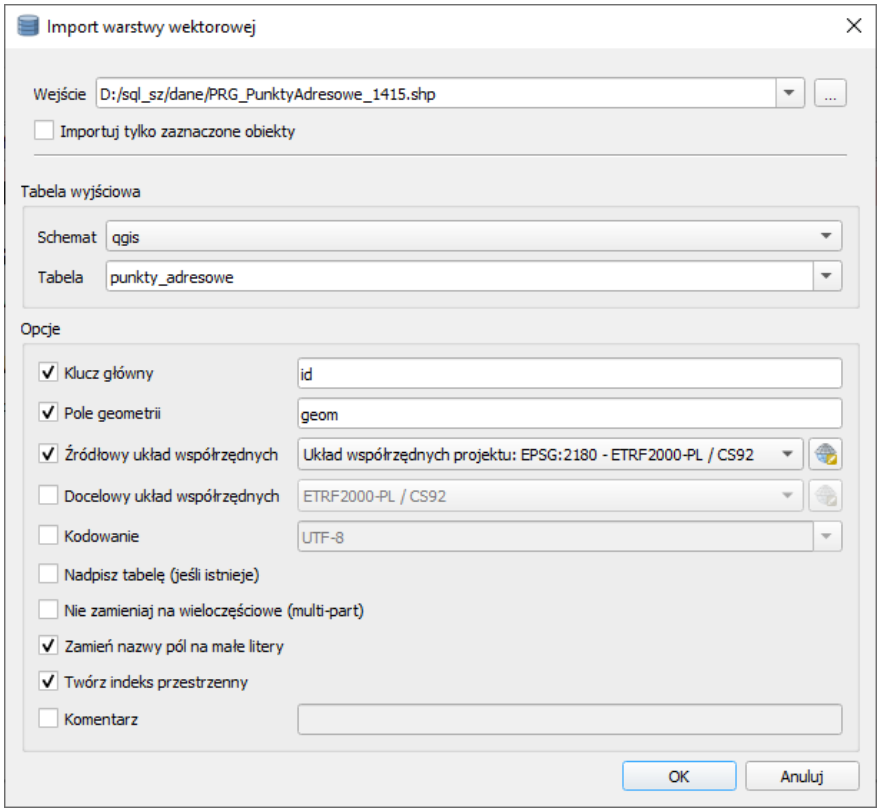
Ćwiczenie 11. Import danych wektorowych do bazy PostgreSQL za pomocą aplikacji QGIS

Import danych wektorowych do bazy wykonywany jest również z poziomu okna **Zarządzanie bazami danych**. Importowanie danych za pomocą QGIS jest prostsze niż poznane wcześniej metody oparte na aplikacjach uruchamianych z okna poleceń, ale wolniejsze.

1. Aby rozpocząć import danych wektorowych do bazy w oknie zarządzania bazami danych należy wybrać polecenie **Importuj warstwę/plik**. Okno narzędzia zostanie uruchomione, jeżeli uprzednio będzie zaznaczona docelowa baza danych, do których ma być wykonany import.

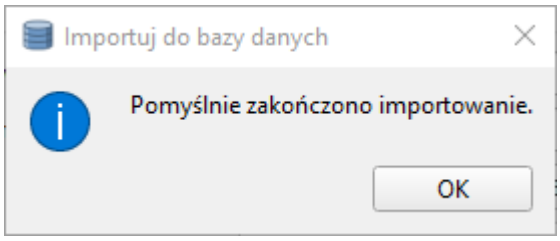


2. Jeżeli przed uruchomieniem okna narzędzia był zaznaczony konkretny schemat bazy, do którego będzie wykonywany import, będzie on teraz wybrany. Jeżeli nie, należy go teraz wskazać. Aby zaimportować warstwę punktów adresowych do schematu qgis, w polu Wejście wskaż plik PRG_PunktyAdresowe_1415.shp z katalogu z danymi do szkolenia.



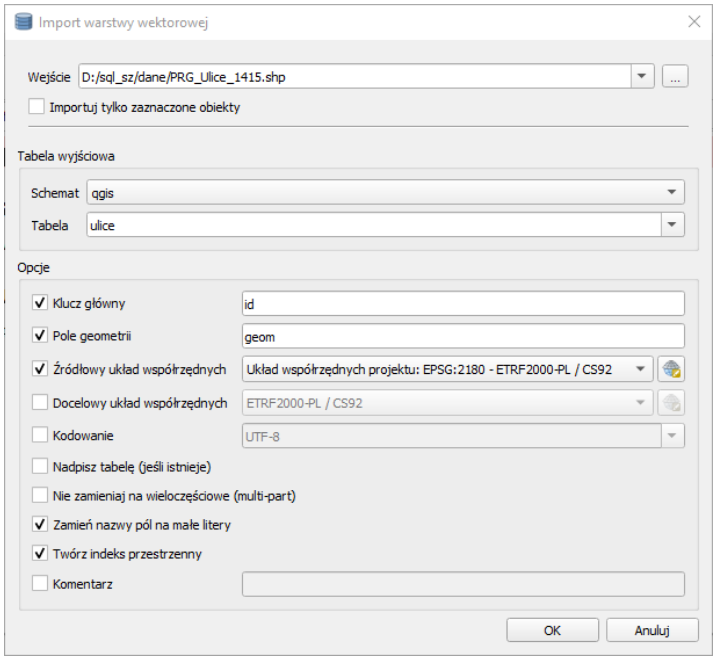
Jako klucz główny, który zostanie utworzony wskazujemy kolumnę **id**, natomiast jako nazwę pola geometrycznego **geom**. Jako układ współrzędnych dla danych wejściowych wybierzmy **EPSG:2180**. Warto na etapie importu zaznaczyć opcję **Zamień nazwy pól na małe litery**, gdyż małe litery są domyślnie obsługiwane w bazie danych, podczas gdy duże litery, podobnie jak znaki specjalne, wymagają zastosowania znaków „” przy każdej operacji na danych, w tym selektach. Z tego samego powodu jako nazwę docelową tabeli wpiszmy: **punkty_adresowe**.

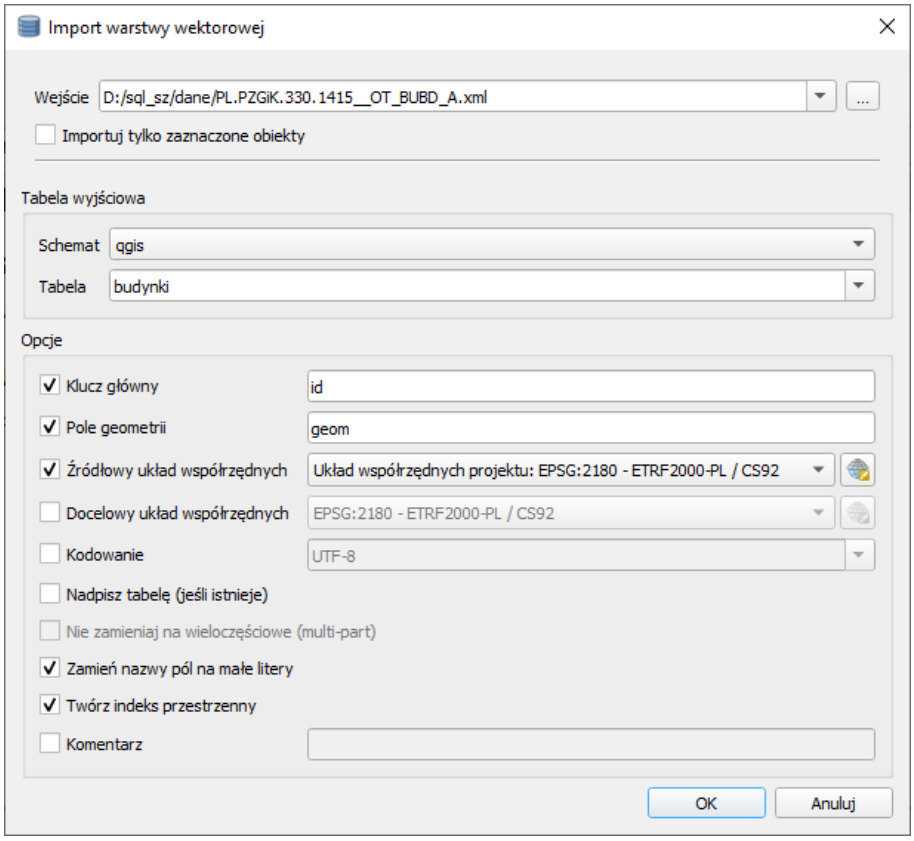
3. Jeżeli dane zostaną pomyślnie załadowane, pojawi się komunikat.

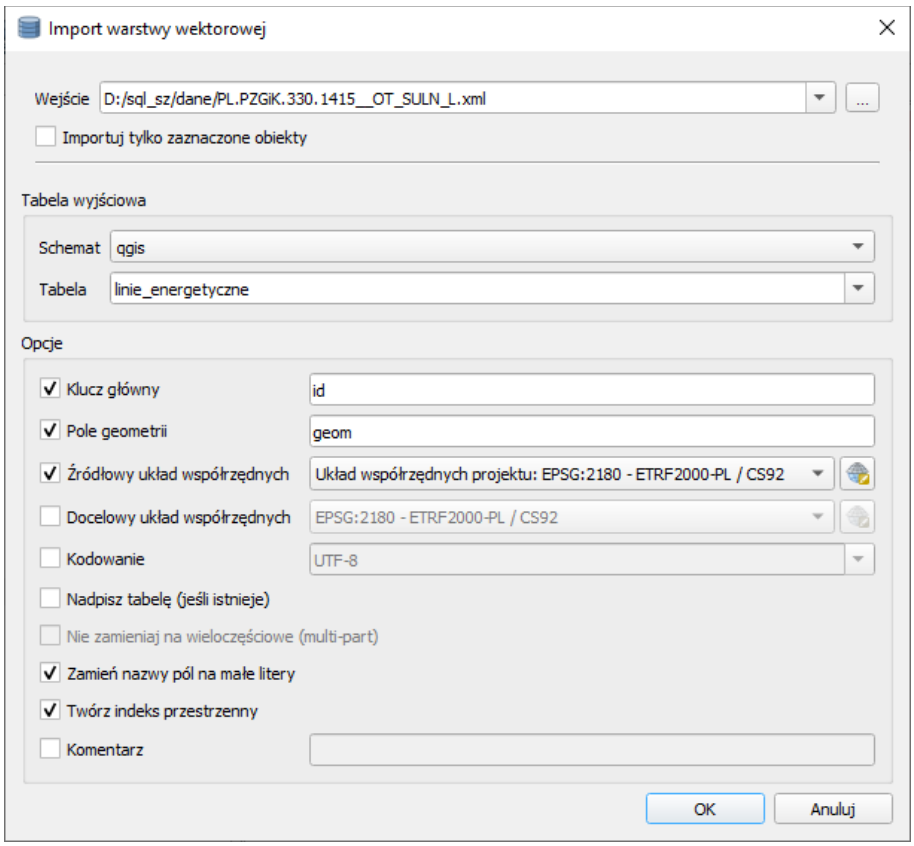


4. Po zaimportowaniu danych nowa tabela *punkty_adresowe* pojawi się w schemacie *qgis.*

5. Analogicznie do warstwy punktów adresowych, w celu przeprowadzenia kolejnych ćwiczeń, importujemy pliki *PRG_Ulice_1415.shp*, *PL.PZGiK.330.1415__OT_BUBD_A.xml* oraz *PL.PZGiK.330.1415__OT_SULN_L.xml* odpowiednio pod nazwami: *ulice, budynki, linie_energetyczne.*



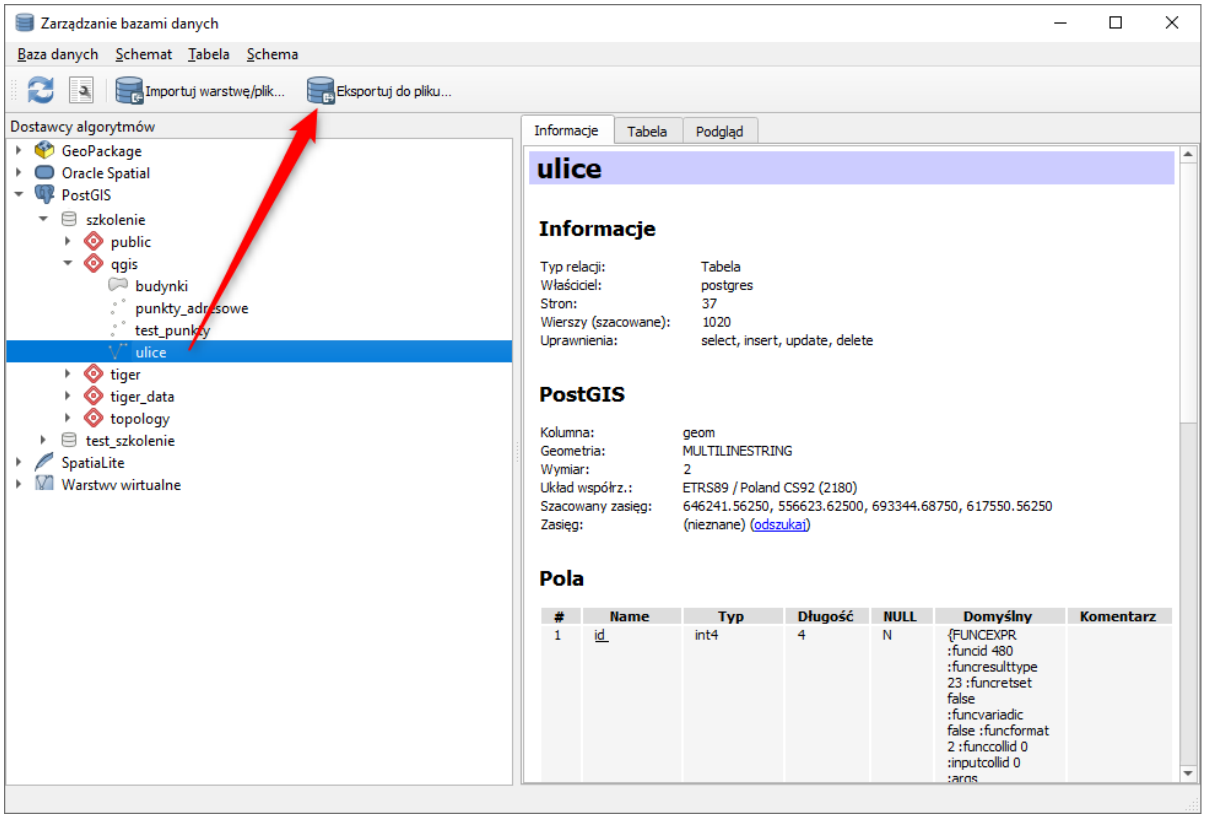

Import warstwy wektorowej
Wejście
D:/sql_sz/dane/PL.PZGiK.330.1415__OT_BUBD_A.xml
Importuj tylko zaznaczone obiekty
Tabela wyjściowa
Schemat
qgis
Tabela
budynki
Opcje
Klucz główny
id
Pole geometrii
geom
Źródłowy układ współrzędnych
Układ współrzędnych projektu: EPSG:2180 - ETRF2000-PL / CS92
Docelowy układ współrzędnych
EPSG:2180 - ETRF2000-PL / CS92
Kodowanie
UTF-8
Nadpisz tabelę (jeśli istnieje)
Nie zamieniaj na wieloczęściowe (multi-part)
Zamień nazwy pól na małe litery
Twórz indeks przestrzenny
Komentarz
OK
Anuluj


Import warstwy wektorowej
Wejście
D:/sql_sz/dane/PL.PZGiK.330.1415__OT_SULN_L.xml
Importuj tylko zaznaczone obiekty
Tabela wyjściowa
Schemat
qgis
Tabela
linie_energetyczne
Opcje
Klucz główny
id
Pole geometrii
geom
Źródłowy układ współrzędnych
Układ współrzędnych projektu: EPSG:2180 - ETRF2000-PL / CS92
Docelowy układ współrzędnych
EPSG:2180 - ETRF2000-PL / CS92
Kodowanie
UTF-8
Nadpisz tabelę (jeśli istnieje)
Nie zamieniaj na wieloczęściowe (multi-part)
Zamień nazwy pól na małe litery
Twórz indeks przestrzenny
Komentarz
OK
Anuluj

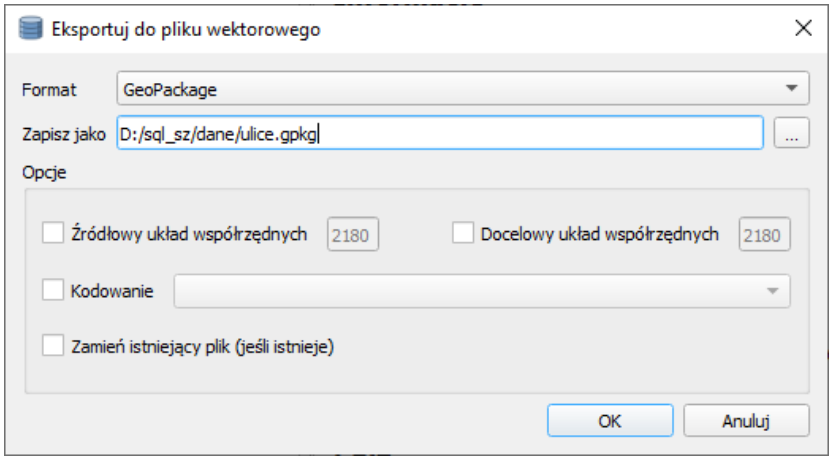
Ćwiczenie 12. Eksport danych wektorowych z bazy PostgreSQL za pomocą aplikacji QGIS

Eksport danych wektorowych z bazy wykonywany jest podobnie jak import z poziomu okna **Zarządzanie bazami danych**.

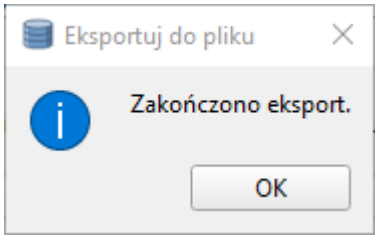
1. Aby przeprowadzić eksport danych w oknie **Zarządzanie bazami danych** wybieramy tabelę **ulice** ze schematu **qgis** oraz naciskamy przycisk Eksportuj do pliku.



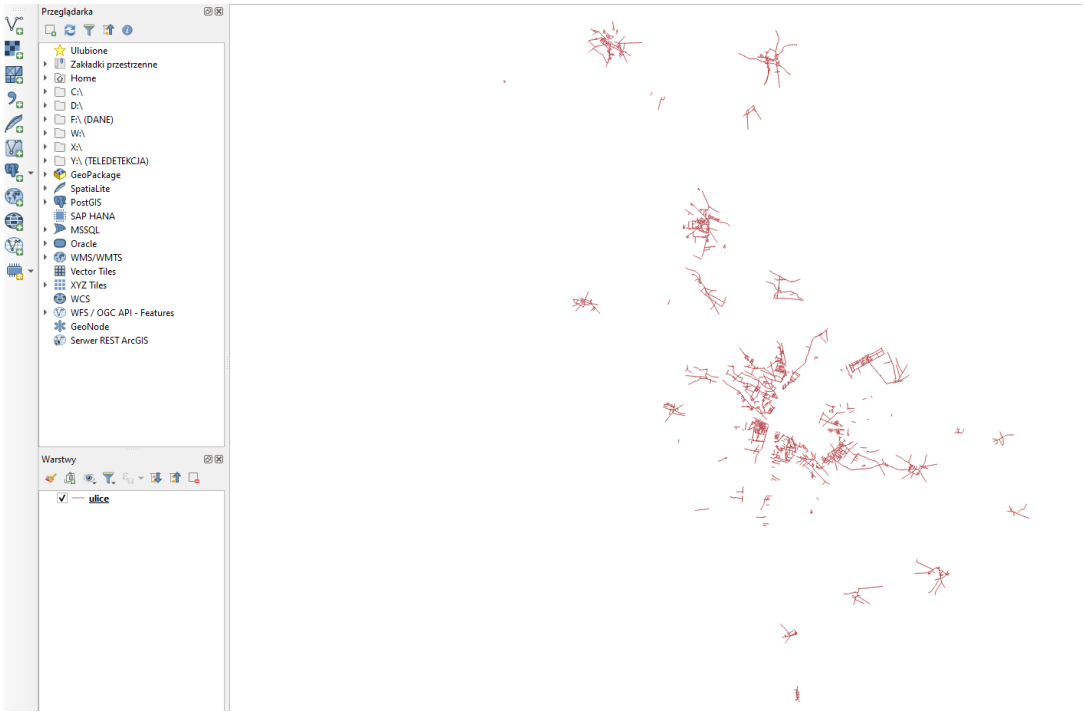
2. Okno eksportu danych umożliwia wybór wielu formatów docelowych, w tym również tabelarycznych. Jako format eksportu pozostawmy domyślny format **GeoPackage**, natomiast jako plik wynikowy **ulice.gpkg** w katalogu szkoleniowym **dane**. W trakcie eksportu możemy dokonać nadania układu współrzędnych warstwie, jeżeli nie był nadany lub jego transformacji. My nie zmieniamy nic, bo chcemy wyeksportować dane, które już są w układzie 2180.



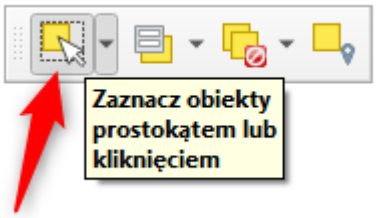
3. Po zakończeniu eksportu pojawi się komunikat:



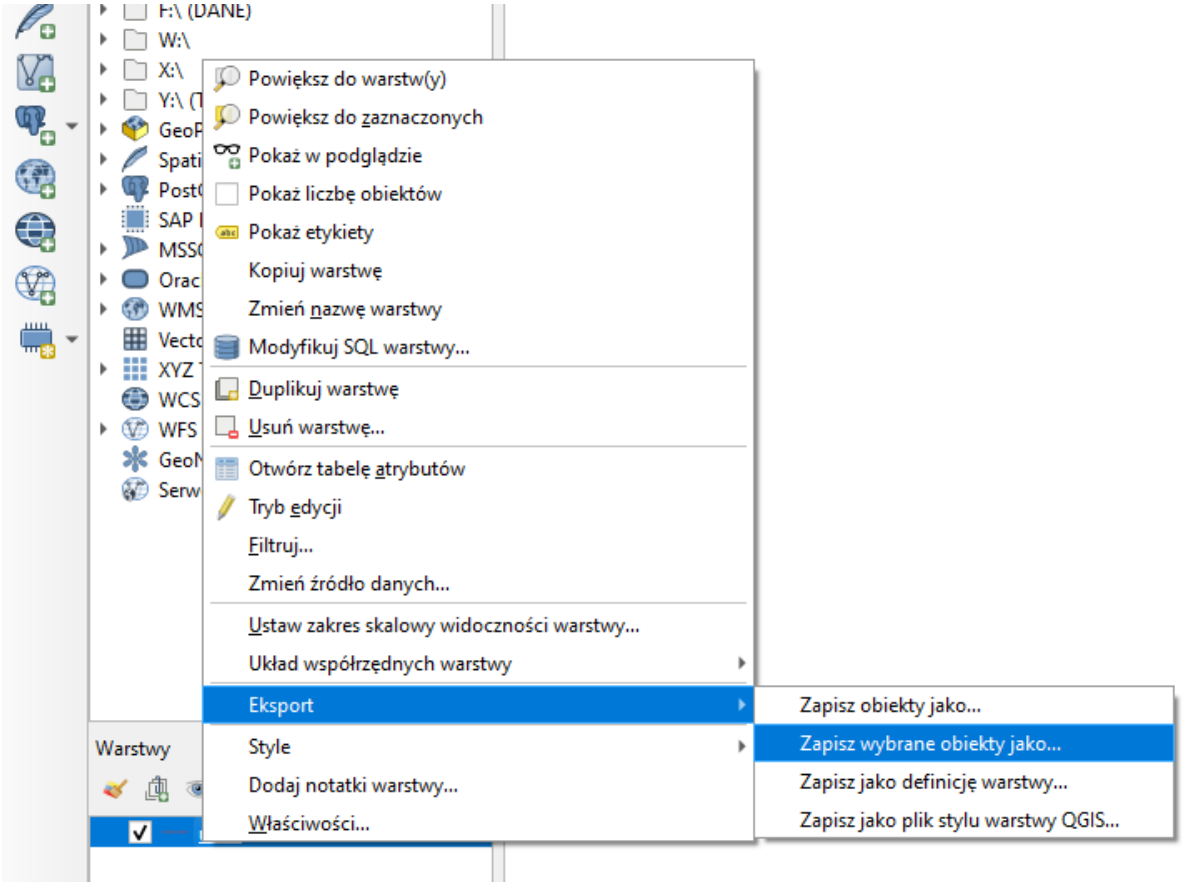
4. Narzędzie to eksportuje wszystkie dane z tabeli. Jeżeli chcemy wyeksportować tylko wybrane obiekty, musimy najpierw wczytać warstwę z bazy danych do drzewa warstw poprzez przeciągnięcie tabeli **ulice** lub dwuklik lewym przyciskiem myszy na tej tabeli.



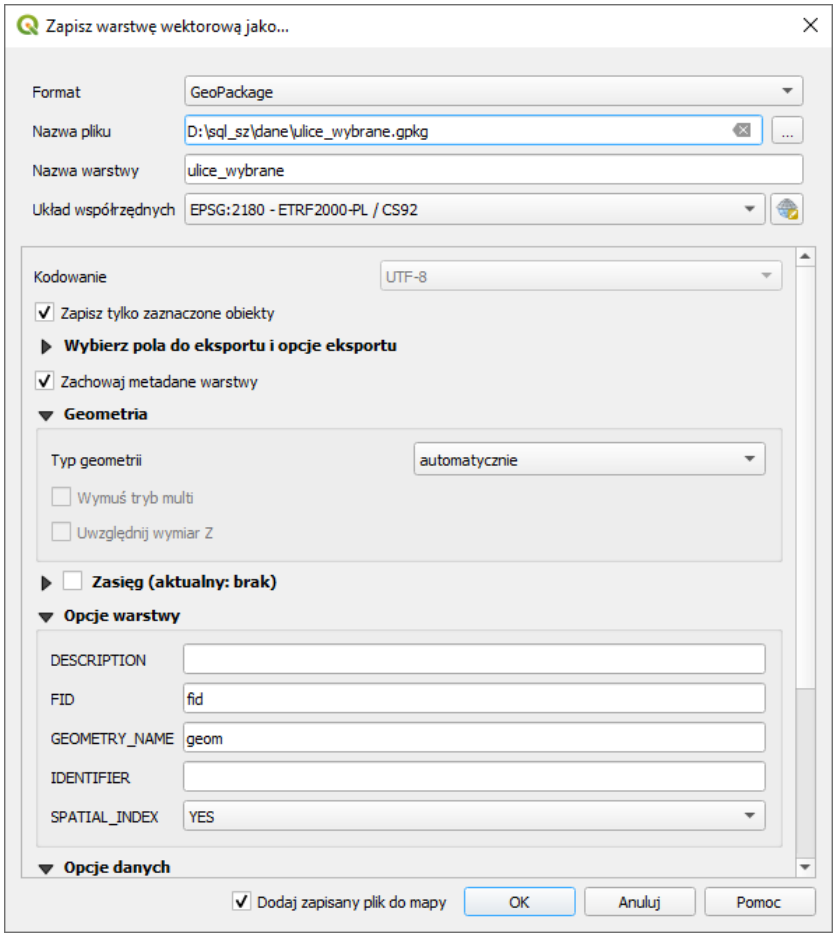
5. Dokonajmy selekcji obiektów na mapie np. za pomocą narzędzia Zaznacz obiekty **prostokątem lub kliknięciem**.



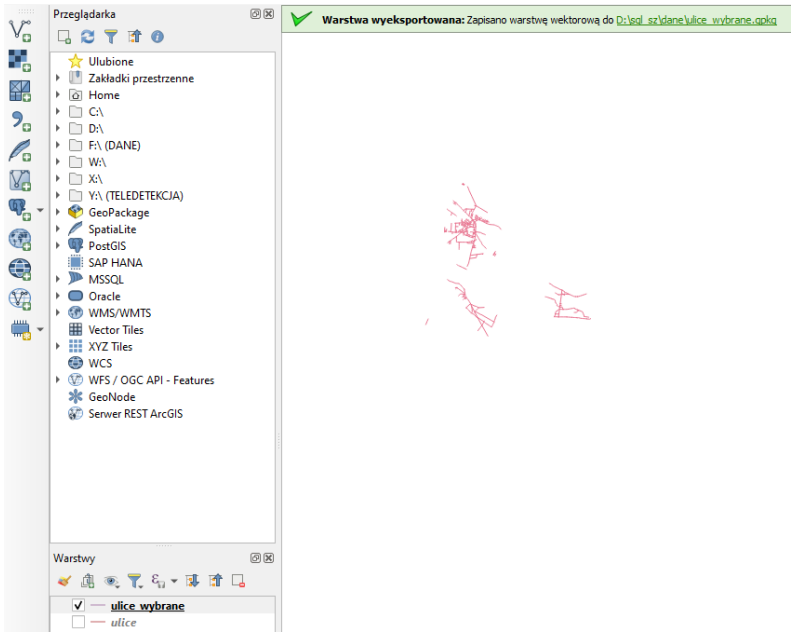
6. Następnie zaznaczamy warstwę **ulice** i pod prawy klawiszem myszy wybieramy opcję **Eksport/Zapisz wybrane obiekty jako…**



7. Podobnie jak poprzednio wybieramy jako format danych **GeoPackage**, a jako nazwę pliku **ulice_wybrane.gpkg** w katalogu **dane**.



8. Jeżeli była zaznaczona opcja **Dodaj zapisany plik do mapy**, nowa warstwa wczyta się automatycznie do aplikacji QGIS. Jak widać w pliku **ulice_wybrane** znajdują się tylko wcześniej wyselekcjonowane obiekty z bazy danych.



7. Wprowadzenie do 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-Weil 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 - Weil 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(OO)
- 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 OGCsą zawsze dwuwymiarowe i nie definiują układów odniesienia. PostGIS rozszerza te reprezentacje o kolejne wymiary oraz układy odniesienia dodając dwie nowe:

- EWKT-ExtendedWKT
- 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 0,001,01 1,01 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 00, 1 0 1,001,000)), ((1 1 0,1 1 1,1 01,1 00, 1 1 0)), ((01 0,01 1,1 1 1,1 1 0,01 0)), ((001,1 01,1 1 1,01 1,001)))
- 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 INT0 geotable ( the_geom, the_name )
VALUES ( ST_GeomFromEWKT('SRID=4326;P0INT(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.

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.

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.

ST_X, ST_Y, ST_Z

Zwracają wartość wskazanej współrzędnej dla punktu

```
SELECT  ST_X(ST_GeomFromEWKT('P0INT(1   2   3   4)1));
```

ST_IsClosed

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

```
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)). (PostGISdoda również współrzędne ZMIN/ZMAX).

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

Edytory

Funkcje pozwalające na edycje geometrii.

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));
```

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 wjęzyku 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('0103000020E61000000100000005000000000000000000000000000000000000
000000000000000000000000000000F03F000000000000F03F000000000000F03F000000000
000F03F000000000000000000000000000000000000000000000000'::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"Szkolenie_PostGIS.md                    31.08.2020
38                                  /                           116
         }
      },
      {
         "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 obwiednią 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;
```

```
columnl     columnl     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 *geometry* 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
)
);
```

```
P0LYG0N((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 zmiennoprzecinkową 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.

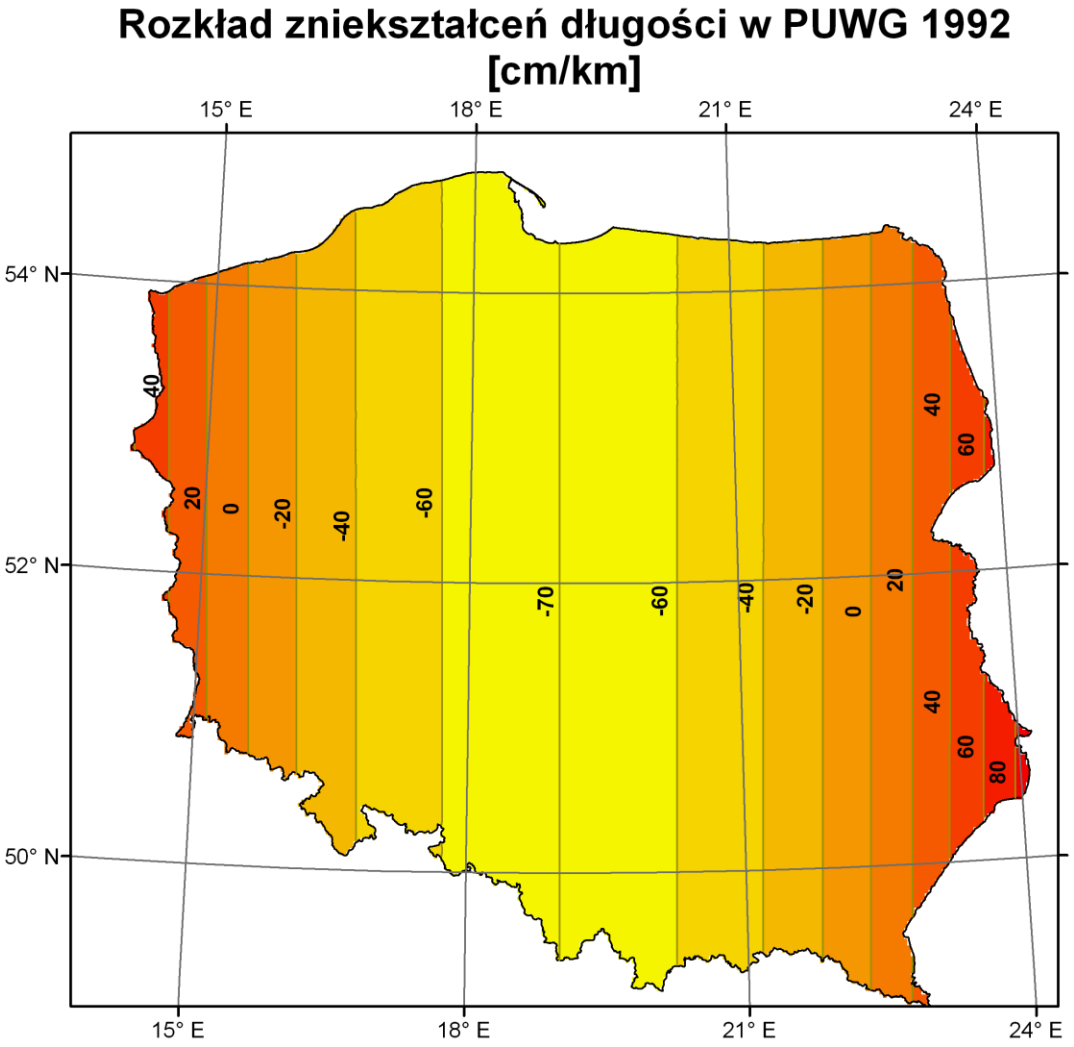
Układ współrzędnych PL-1992

KodEPSG:2180

Układ współrzędnych 1992 (Państwowy Układ Współrzędnych Geodezyjnych 1992) - układ współrzędnych płaskich prostokątnych oparty na odwzorowaniu Gaussa-Krugera 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**

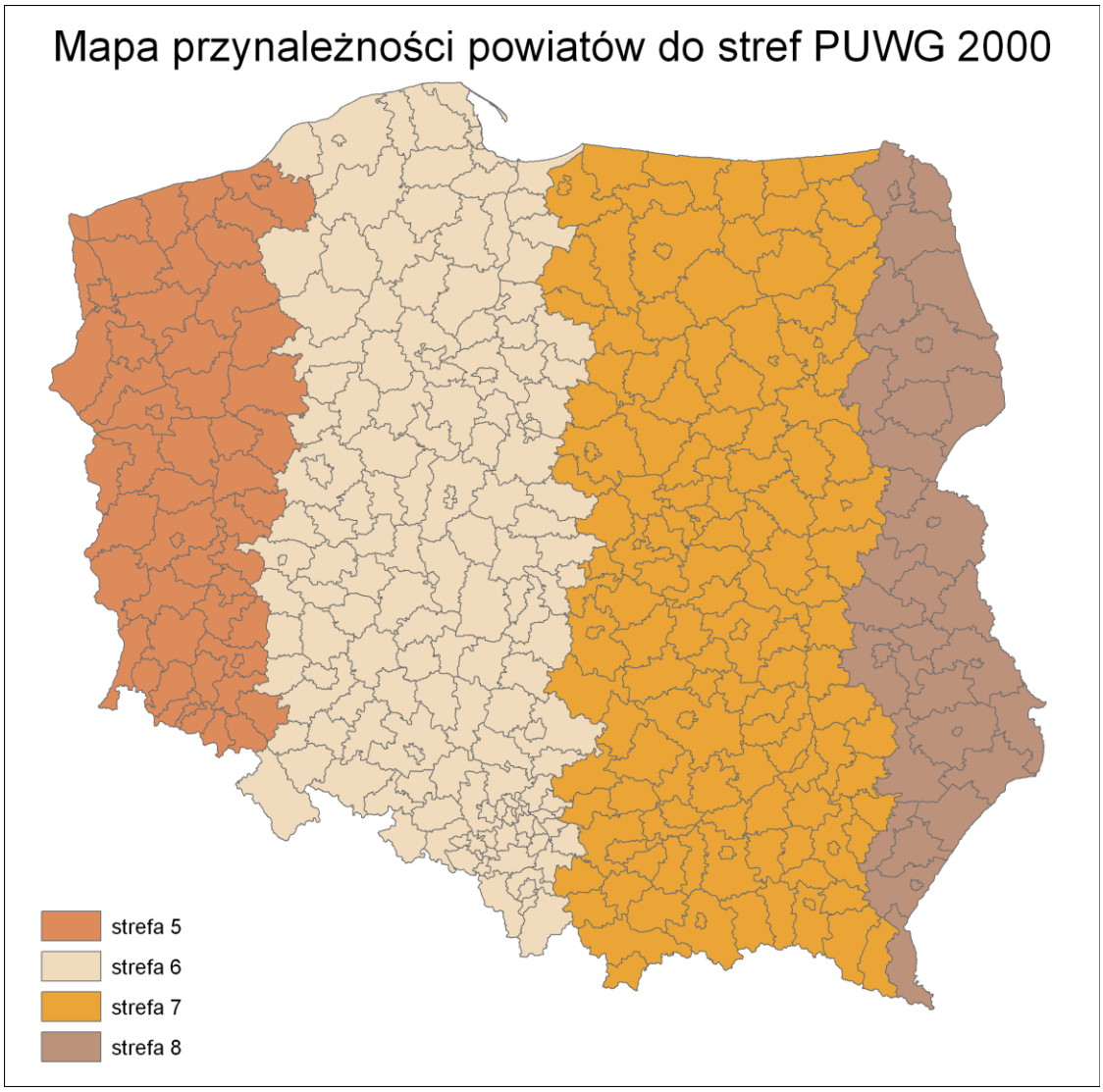
Układ współrzędnych 2000 (Państwowy Układ Współrzędnych Geodezyjnych 2000, PL-2000) - układ współrzędnych płaskich prostokątnych zwany układem „2000", powstały w wyniku zastosowania odwzorowania Gaussa-Krügera dla elipsoidy GRS 80 w czterech trzystopniowych strefach o południkach osiowych 15°E, 18°E, 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 1979 r., zatwierdzony rezolucją nr 1 na zgromadzeniu podkomisji EUREF(IAG Reference Frame Sub-Commission for Europe) we Florencji w 1990 r. 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, wczasach 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 zapisu 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.

8. Tworzenie i operacje na danych wektorowych za pomocą SQL

Ćwiczenie 13. Utworzenie schematu bazy danych oraz tabel z poziomu aplikacji pdAdmin

Tworzenie struktur oraz praca z bazą danych w języku SQL możliwe są za pomocą różnych klientów bazy danych w tym za pomocą transakcyjnego psql. W kolejnych ćwiczeniach będziemy działali za pomocą najczęściej używanego w przypadku bazy danych PostgreSQL darmowego oprogramowania pgAdmin.

1. Utworzenie nowego schematu na bazie danych **pzgik**, z danymi z **Państwowego Zasobu Geodezyjnego i Kartograficznego** oraz tabel potrzebnych w dalszych ćwiczeniach.

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

-- tworzymy schemat o nazwie pzgik
create schema pzgik;

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

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

-- tworzymy tabelę dla ulic
drop table if exists pzgik.ulice;
create table pzgik.ulice (
id serial primary key,
typ_ulicy text,
nazwa text,
geom geometry(MultiLineString, 2180)
);
-- tworzymy tabelę dla gmin
drop table if exists pzgik.gminy;
create table pzgik.gminy (
id serial primary key,
nazwa text,
```

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

2. Załadowanie danych do tabeli *pzgik.budynki* za pomocą konstrukcji **insert … select**.

```
insert into pzgik.budynki (budynek, liczbakondygnacji, geom)
select
funogolnabudynku,
liczbakondygnacji,
geom
from
qgis.budynki;
```

Kolumna klucza głównego (id) została celowo pominięta, ponieważ zdefiniowaliśmy ją jako *serial* więc sama wypełni się unikalnymi identyfikatorami.
Ilość wstawionych obiektów: **70928.**

3. Załadowanie danych do tabeli *pzgik.adresy* za pomocą konstrukcji **insert … select.**

```
insert into pzgik.adresy (kod_poczt, miejscowosc, ulica, numer, geom)
select
pna,
simc_nazwa,
ulic_nazwa,
numer,
geom
from
qgis.punkty_adresowe;
```

Ilość wstawionych obiektów: **28027.**

4. Załadowanie danych do tabeli *pzgik.ulice* za pomocą konstrukcji **insert … select**.

```
insert into pzgik.ulice (typ_ulicy, nazwa, geom)
select
cecha,
ulic_nazwa,
geom
from
qgis.ulice;
```

Ilość wstawionych obiektów: **1020.**

5. Załadowanie danych do tabeli *pzgik.gminy* za pomocą konstrukcji **insert … select**.

```
insert into pzgik.gminy (nazwa, teryt, geom)
select
jpt_nazwa_,
jpt_kod_je,
geom
```

```
from
public.gminy;
```

Ilość wstawionych obiektów: **314.**

Ćwiczenie 14. Obliczenie ilości budynków w 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 do 10 gmin z największą ilością budynków

1. W tym celu uruchamiamy polecenie.

```
select nazwa, count(*) as ilosc
from pzgik.gminy g
join pzgik.budynki b on st_intersects(g.geom, b.geom)
group by nazwa
order by 2 desc nulls last
limit 10;
```

Zapytanie zwróciło wynik, ale czas zapytania był długi: 5 min 54 sec.

Data Output | Explain | Mess

	nazwa text	ilosc bigint
1	Kadzidło	8378
2	Goworowo	8328
3	Myszyniec	7901
4	Olszewo-Borki	7163
5	Rzekuń	7145
6	Łyse	7065
7	Lelis	6686
8	Baranowo	5941
9	Czerwin	5229
10	Troszyn	4746

2. Aby zwiększyć szybkość wykonywania analizy dodamy indeksy przestrzenne dla obu tabel z geometrią.

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

3. Ponownie uruchamiamy polecenie z punktu 1.

```
select nazwa, count(*) as ilosc
from pzgik.gminy g
join pzgik.budynki b on st_intersects(g.geom, b.geom)
group by nazwa
order by 2 desc nulls last
limit 10;
```

Zapytanie zwróciło ten wynik, ale w czasie 901 msec, a więc nieporównywalnie krótszym.

Ćwiczenie 15. Przypisanie gminy do punktów adresowych

W celu przypisania nazwy gminy do tabeli punktów adresowych należy dodać do tej tabeli dodatkowy atrybut, który będzie przechowywał tą informację.

1. Utworzenie atrybutu **gmina** w tabeli *pzgik.adresy*

```
alter table pzgik.adresy add column gmina text;
```

2. W celu przyspieszenia analiz przestrzennych dodajmy indeks przestrzenny do tabeli *pzgik.adresy*.

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

3. W calu zakodowanie nazwy gminy dla punktów adresowych wywołujemy polecenie.

```
update
pzgik.adresy
set
gmina = g.nazwa
from
pzgik.gminy g
where
st_intersects(adresy.geom, g.geom);
```

Polecenie powinno zwrócić informacje:

```
UPDATE 28027

Query returned successfully in 1 secs 261 msec.
```

4. Sprawdźmy wypełnienie atrybutu gmina w tabeli adresy dla 10 rekordów.

```
select * from pzgik.adresy limit 10;
```

Zapytanie zwraca dane z tabeli adresy z zakodowaną informacją o nazwie gminy:

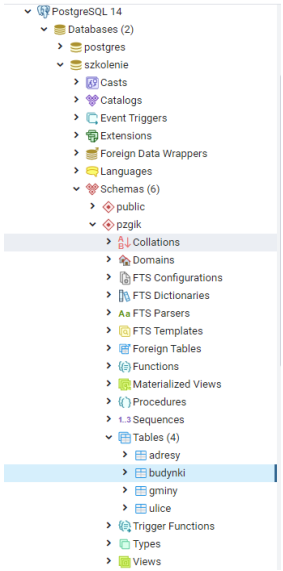
Data Output | Explain | Messages | Notifications

	id [PK] integer	kod_poczt text	miejscowosc text	ulica text	numer text	geom geometry	gmina text
1	2958	07-405	Repki	[null]	6	0101000020840800...	Troszyn
2	14963	07-415	Nakły	[null]	39	0101000020840800...	Olszewo-Borki
3	18227	07-402	Gnaty	[null]	16	0101000020840800...	Lelis
4	25368	07-402	Gnaty	[null]	23B	0101000020840800...	Lelis
5	25429	07-402	Gnaty	[null]	9E	0101000020840800...	Lelis
6	27491	07-407	Gostery	[null]	17A	0101000020840800...	Czerwin
7	27598	07-407	Seroczyn	[null]	41	0101000020840800...	Czerwin
8	16114	07-407	Wojsze	[null]	5	0101000020840800...	Czerwin
9	16130	07-407	Grodzisk-Wieś	[null]	4	0101000020840800...	Czerwin
10	16133	07-407	Wojsze	[null]	2	0101000020840800...	Czerwin

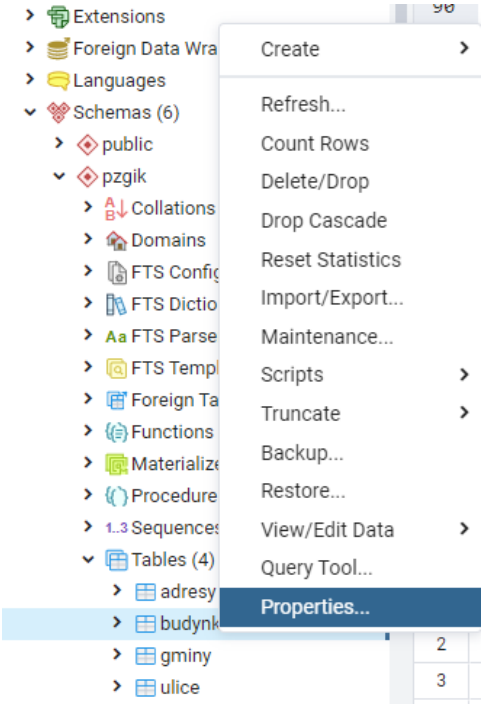
Ćwiczenie 16. Zakodowanie adresu do budynku

W celu przypisania adresu do budynków, konieczne jest dodanie atrybutu **adres** do tabeli *pzgik.budynki*. Atrybut można dodać jak w powyższym ćwiczeniu za pomocą polecenie SQL, lub za pomocą dedykowanego narzędzie w **pgAdmin**. Teraz skorzystamy z tego drugiego sposobu.

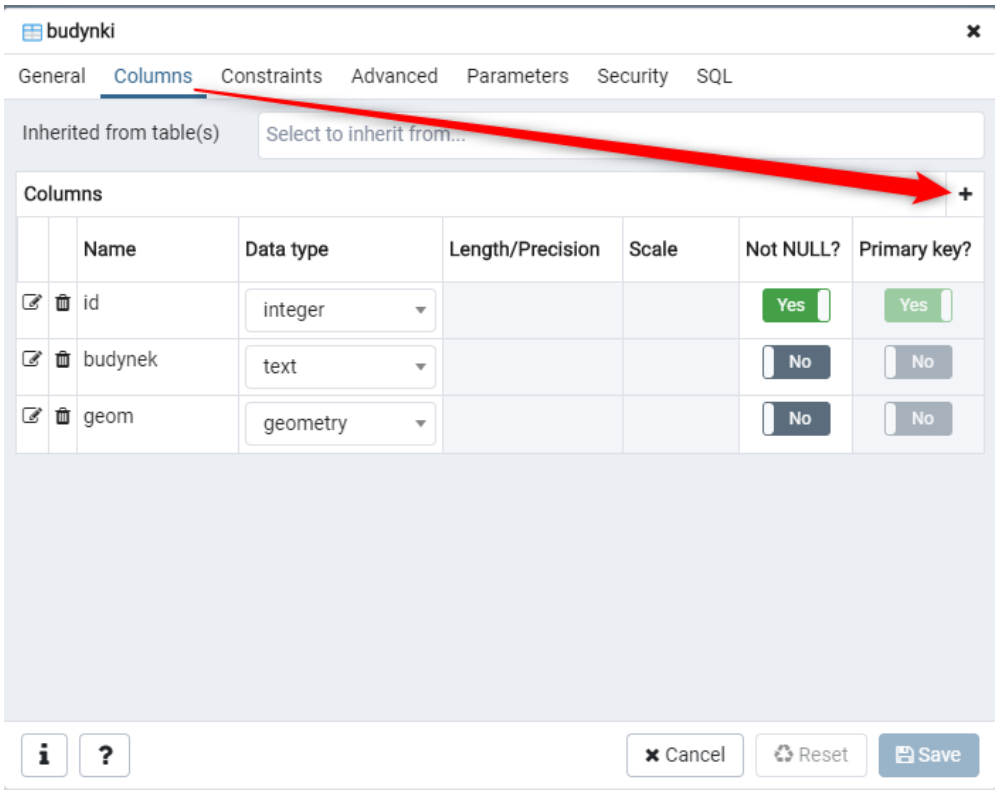
1. W drzewie tabel bazy danych PostgreSQL zaznacz tabelę *budynki* w schemacie *pzgik*.



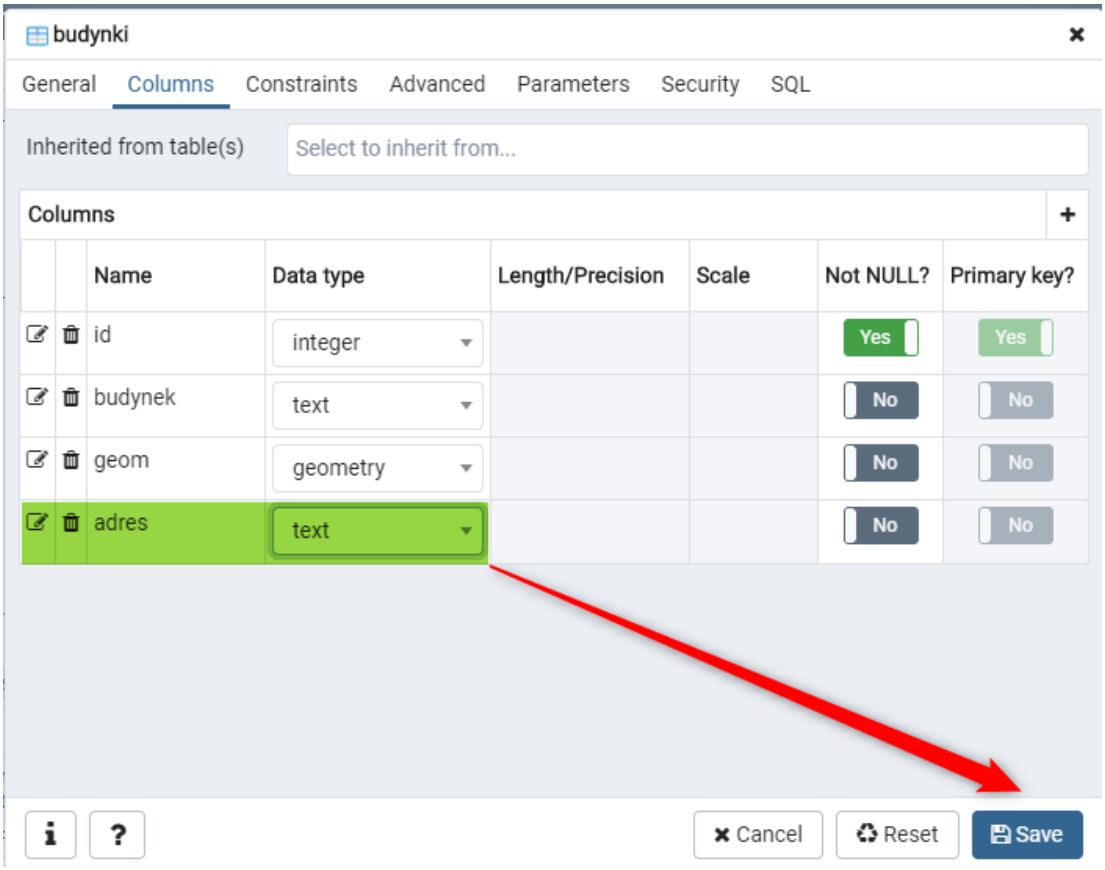
2. Kliknij prawym klawiszem myszy i wybierz opcję **Properties**.



3. W oknie **budynki** przejdź do zakładki **Columns** i wciśnij przycisk **+**



4. W nowym wierszu na dole listy atrybutów wpisz nazwę atrybutu: **adres** oraz jego typ: **text**. Zatwierdź przyciskiem **Save**.



Ponieważ zarówno tabela *pzgik.budynki* jak i *pzgik.adresy* mają utworzone indeksy przestrzenne, możemy od razu wykonać kodowanie z użyciem funkcji przestrzennej **st_intersects**. W celu zapisania w polu adres kompletnego adresu, składającego się z nazwy ulicy, numeru adresowego, kodu pocztowego i nazwy miejscowości, które to atrybuty są przechowywane w tabeli pzgik.adresy jako osobne atrybuty, podczas kodowania dokonamy ich złączenia w jeden tekst np. **Spacerowa 5, 07-407 Czerwin**.
W przypadku, gdy w danej miejscowości nie będzie ulicy, adres powinien mieć postać: **Zaorze 57, 07-407 Zaorze**

```
update pzgik.budynki
set adres =
case
when ulica is not null then ulica||' '||numer||', '||kod_poczt||'
'||miejscowosc
else miejscowosc||' '||numer||', '||kod_poczt||' '||miejscowosc
end
from pzgik.adresy a
where st_intersects(budynki.geom, a.geom);
```

5. Aby sprawdzić ile budynków pozyskało informację o adresie wykonajmy zapytanie.

```
select count(*) from pzgik.budynki where adres is not null;
```

Takich budynków jest **23864**.

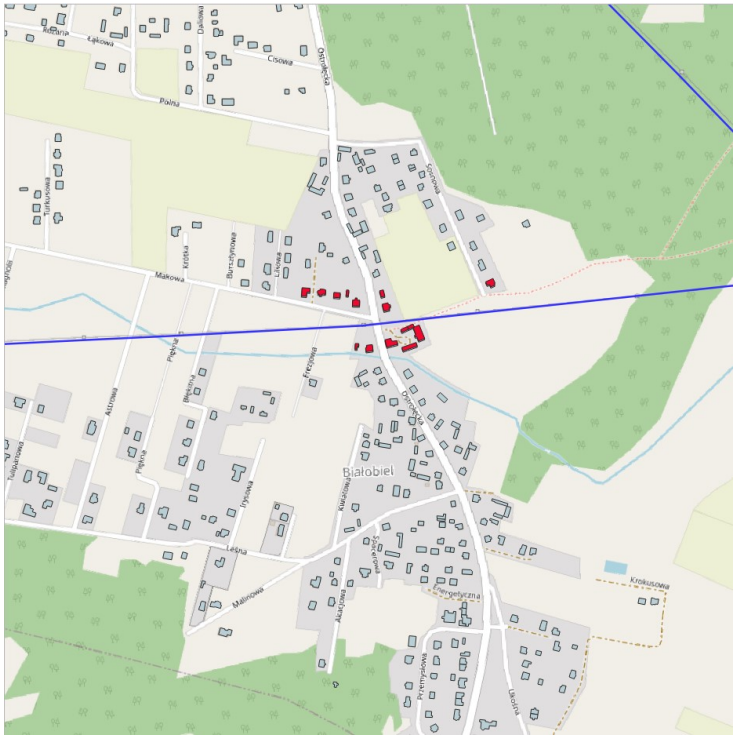
Ćwiczenie 17. Utworzenie warstwy budynków w pobliżu sieci energetycznych wysokiego napięcia

Korzystając z zaimportowanej uprzednio w QGIS tabeli *qgis.linie_energetyczne*, pochodzącej z bazy **BDOT10k**, utworzymy warstwę budynków, które znajdują się w odległości mniejszej niż 50 m od linii wysokiego napięcia. W bazie BDOT10k klasyfikacja napięcia w napowietrznych liniach energetycznych zapisana jest w atrybucie *rodzaj*. Jako linie z wysokim napięciem przyjmiemy linie, dla których atrybut *rodzaj* przyjmuje wartości '**WN'** (wysokie napięcie) oraz '**NN'** (najwyższe napięcie). W celu zapisu wyniku analizy do nowej tabeli, użyjemy polecenie *CREATE TABLE* w połączniu z poleceniem *SELECT*.

1. Aby utworzyć nową warstwę budynków, spełniającą powyższe kryterium wykonujemy polecenie.

```
create table pzgik.budynki_linenerg as
select
a.*
from pzgik.budynki a, qgis.linie_energetyczne b
where
b.rodzaj in ('WN', 'NN') and
st_intersects(a.geom, st_buffer(b.geom,50));
```

2. Wynik analizy możemy zobaczyć w aplikacji QGIS wczytując powstałą tabelę *pzgik.budynki_linenerg*.



9. Operacje na rastrach w bazie PostgreSQL

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;
```

W naszym przypadku podczas instalacji PostgreSQL rozszerzenie to zostało automatycznie dodane. W takim przypadku po wykonaniu powyższego polecenie pojawi się komunikat:

```
ERROR:  extension "postgis_raster" already exists
SQL state: 42710
```

*Ćwiczenie 18. Import rastra do bazy z pomocą aplikacji **raster2pgsql***

Import danych rastrowych do bazy za pomocą aplikacji **raster2pgsql** uruchamia się analogicznie jak miało to miejsce w przypadku aplikacji dla danych wektorowych, czyli z poziomu wiersza poleceń. Aplikacja instalowana jest razem z bazą danych PostgreSQL.

1. Aby zobaczyć wszystkie dostępne parametry uruchamiamy aplikację wpisując:

```
"c:/Program Files/PostgreSQL/14/bin/raster2pgsql.exe"
```

```
RELEASE: 3.2.3 GDAL_VERSION=34 (3.2.3)
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 import obiema metodami.

2. Do bazy danych będziemy importowali raster numerycznego modelu terenu o wielkości piksela 10mx10m, pochodzący z PZGiK i docięty dla zmniejszenia jego wielkości do granicy powiatu ostrołęckiego. Aby zaimportować raster uruchamiamy komendę:

```
"c:/Program Files/PostgreSQL/14/bin/raster2pgsql.exe" -c -s 2180 -t 100x100 -f rast -I nmt10_lzw.tif nmt_10 > nmt_10.sql
```

W trakcie importu pliku do rastra będzie wyświetlany komunikat:

```
Processing 1/1: nmt10_lzw.tif
```

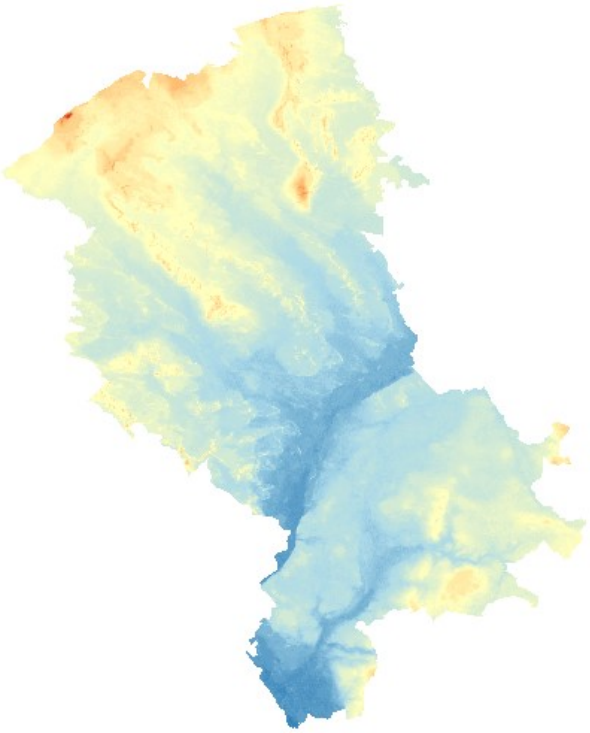
3. Wynikowy skrypt SQL (nmt_10.sql) zaimportujemy do bazy za pomocą poznanej wcześniej aplikacji **psql**

```
"c:/Program Files/PostgreSQL/14/bin/psql.exe" -U postgres -f nmt_10.sql szkolenie
```

Jeżeli import zakończy się powodzeniem zostaną wyświetlone komentarze:

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

4. Zaimportowany raster **public.nmt_10** można teraz przeglądać w aplikacji QGIS.



5. Import rastra w całości wykonujemy w ten sam sposób, tylko pomija się parametr **-t** oznaczający kafelkowanie docelowego rastra. Import wykonujemy więc za pomocą polecenia:

```
"c:/Program Files/PostgreSQL/14/bin/raster2pgsql.exe" -c -s 2180 -f rast -I nmt10_lzw.tif
nmt_10_calosc > nmt_10_calosc.sql
```

6. Podobnie jak poprzednio zapisujemy raster w bazie danych za pomocą **psql**

```
"c:/Program Files/PostgreSQL/14/bin/psql.exe" -U postgres -f nmt_10_calosc.sql szkolenie
```

Tym razem komunikat będzie krótszy:

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

7. Zaimportowany raster **public.nmt_10_calosc** można również przeglądać w aplikacji QGIS.

Ćwiczenie 19. Przypisanie wysokości dla każdego budynku

Funkcje *postgis_raster* oraz *postgis* umożliwiają wykonywanie operacji na danych przestrzennych zarówno rastrowych i wektorowych oddzielnie lub łącznie, czyli można łączyć jednocześnie funkcje z obu rozszerzeń. To pozwala np. na wykonywanie złączeń przestrzennych między danymi wektorowymi i przestrzennymi. I taka też funkcjonalność zostanie wykorzystana w tym ćwiczeniu.

1. W celu pozyskania wysokości dla budynku najlepiej jest wykonać tą operację dla konkretnego obiektu punktowego reprezentującego budynek. Może to być centroid budynku, jednak środek ciężkości budynku może w przypadku niektórych kształtów budynków znajdować się poza samym obiektem. Dlatego w ćwiczeniu tym będziemy wyznaczać tzw. labelpoint, czyli punkt znajdujący się możliwie blisko środka ciężkości budynku, ale zawsze w ramach jego obrysu. Aby zakodować dodatkową geometrię dla rekordów budynków musimy dodać do tabeli budynków dodatkowe pole geometryczne, na co baza PostgreSQL jak najbardziej pozwala. W tym celu wykonujemy polecenie:

```
alter table pzgik.budynki add column labelpoint geometry(Point,2180);
```

2. Obliczenie geometrii labelpoint z geometrii budynku dokonuje się za pomocą funkcji **st_pointonsurface**

```
update pzgik.budynki set labelpoint = st_pointonsurface(geom);
```

3. Utwórzmy indeks przestrzenny dla dodatkowej geometrii tabeli budynków, aby przyspieszyć analizy przestrzenne z użyciem rastrów.

```
create index budynki_cent_idx on pzgik.budynki using gist(labelpoint);
```

4. W celu trwałego dopisania wysokości do budynków utwórzmy pole do jej przechowywania.

```
alter table pzgik.budynki add column wysokosc numeric;
```

5. Wreszcie możemy przystąpić do obliczenia wysokości każdego budynku z użyciem numerycznego modelu terenu.

```
update pzgik.budynki set wysokosc = ST_Value(r.rast, labelpoint,
true)
from public.nmt_10 r
where ST_Intersects(labelpoint, rast);
```

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

```
select count(*) from pzgik.budynki where wysokosc is null;
```

Ćwiczenie 20. Wygenerowanie rastra z cieniowaniem terenu

Zapisanie rastra w całości wykorzystywane jest w tych sytuacjach, w których chcemy utworzyć produkty pochodne, czyli nowe rastry będące wynikiem przetworzenia pierwotnego rastra z pomocą funkcji **postgis_raster**.

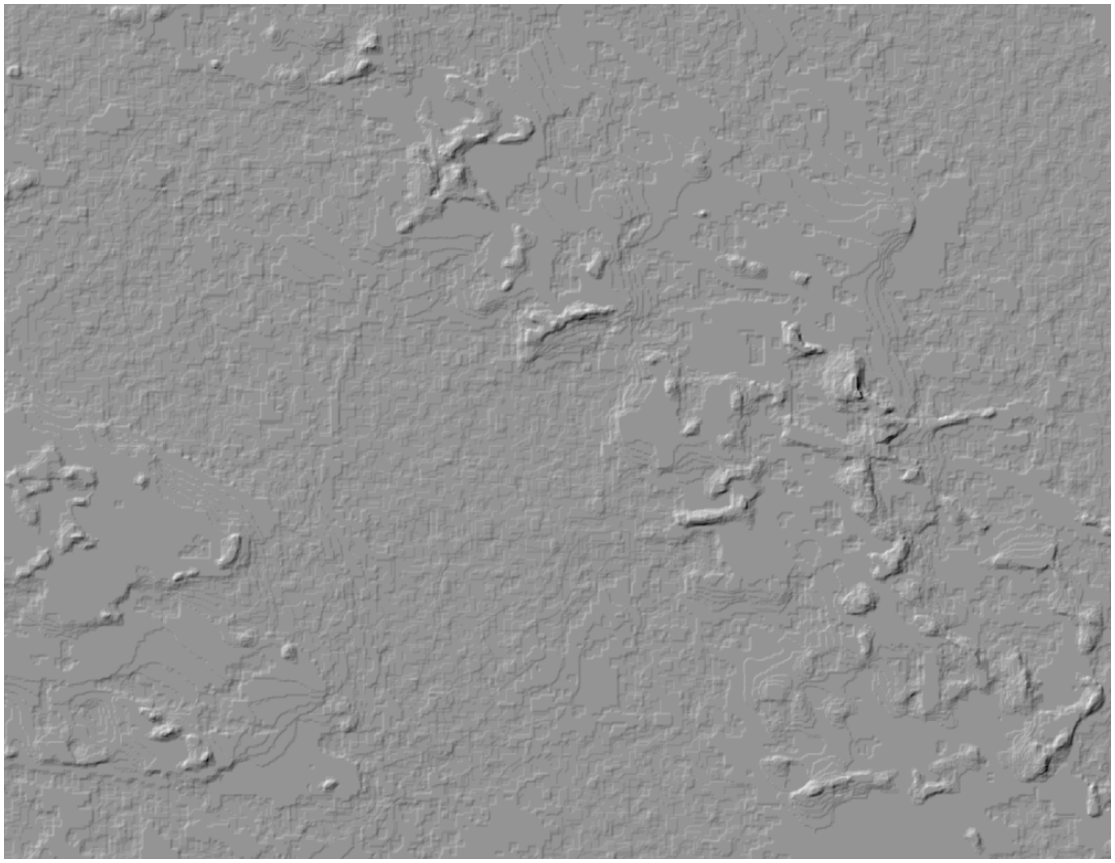
1. W postgis_raster dostępna jest funkcja do generowania rastra z cieniowaniem terenu **st_hillshade**. Aby wygenerować taki nowy raster i zapisać go w bazie danych wykonujemy polecenie:

```
drop table if exists public.nmt_10_hillshade;
create table public.nmt_10_hillshade as
select rid, ST_HillShade(rast, 1, '32BF', 315, 45, 255, 1, false) as rast
from public.nmt_10_calosc;
```

Ponieważ generowany jest nowy raster procedura jest dość czasochłonna.

```
Query returned successfully in 14 min 53 secs.
```

2. Wczytajmy utworzony raster do aplikacji QGIS.



Ćwiczenie 21. Wygenerowania mapy spadków terenu i przypisanie spadków do budynków

1. W **postgis_raster** dostępna jest funkcja do generowania spadków terenu **st_slope**. Aby wygenerować taki raster i zapisać go w bazie danych wykonujemy polecenie:

```
drop table if exists public.nmt_10_slope;
create table public.nmt_10_slope as
select rid, ST_Slope(rast, 1, '32BF', 'DEGREES', 111120, false) as rast
from public.nmt_10_calosc;
```

Również i tym razem procedura tworzenia nowego rastra może być dość czasochłonna.

```
Query returned successfully in 12 min 8 secs.
```

2. Aby użyć nowego rastra do analiz przestrzennych warto utworzyć z niego wersję pokafelkowaną. Tym razem również utworzymy kafle o wielkości 100mx100m.

```
drop table if exists public.nmt_10_slope_tiles;
create table public.nmt_10_slope_tiles as select ST_Tile(rast, 1, 100, 100,
false, null) as rast
from public.nmt_10_slope;
```

Tym razem operacja nie trwała długo:

```
Query returned successfully in 4 secs 959 msec.
```

3. Utwórzmy dodatkowe pole w tabeli *pzgik.budynki* aby wstawić spadki pobrane z rastra.

```
alter table pzgik.budynki add column spadki numeric;
```

4. Uzupełnienie tabeli o wartość spadków terenu, pod którym leży budynek wykonamy za pomocą polecenia:

```
update pzgik.budynki set spadki = ST_Value(r.rast, labelpoint, true)
from public.nmt_10_slope_tiles r
where ST_Intersects(labelpoint, rast);
```

5. Wczytajmy uzupełnioną warstwę budynków i wykonajmy klasyfikację po wartości atrybutu spadki.

10. Wskaźniki statystyczne, widoki bazodanowe i zestawienia

W poprzednich ćwiczeniach wykonywaliśmy proste i złożone zapytania do bazy danych korzystając z polecenia *SELECT*. W niektórych przypadkach utrwalaliśmy wynik takiego zapytania w postaci nowej tabeli danych z użyciem polecenia *CREATE TABLE AS*. W tym bloku ćwiczeń w celu wykonywania zestawień i raportów statystycznych z danych skorzystamy dodatkowo z dwóch innych istotnych elementów baz danych, a mianowicie widoków i widoków zmaterializowanych.

Pod pojęciem **widoku bazodanowego** (perspektywy) rozumie się wirtualną tabelę, tworzoną dynamicznie na podstawie zapytania SQL. Widok, podobnie jak tabela, posiada kolumny o określonym typie i wiersze, ale w odróżnieniu od zwykłej tabeli, wartości wyświetlane w poszczególnych wierszach i kolumnach są pozyskiwane w wyniku wykonania zapytania SQL, stanowiącego definicje widoku. Wartości w widoku nie są więc zapisane na stałe jak w tabeli, ale każde odpytanie widoku powoduje ponowne wykonanie zapytania SQL i wyświetlenie wyniku. Dlatego też wartości pozyskiwane z widoków są zawsze aktualne. Można sobie po prostu wyobrazić widok bazodanowy jako utrwalone zapytanie SQL, występujące jako element strukturalny bazy danych.

Widoki są powszechnie stosowane do zbierania informacji z bazy danych, zarówno z pojedynczych tabel, jak i zestawów tabel i zwyczajowo prezentuje się w nich konkretne informacje wybierając z tabel tylko te kolumny, z których informacje są nam w widoku potrzebne. Widoki mają więc zastosowanie przede wszystkich do robienia zestawień i raportów z danych, przy czym sposób prezentacji danych, czy nazwy atrybutów, można dowolnie zmieniać w porównaniu do nazw i wartości z tabel, z których widok jest wykonywany. Widoki mają również ogromne znaczenie dla bezpieczeństwa udostępniania danych, gdyż pozwalają na odcięcie dostępu do tabel, w których przechowywane są dane. Z uwagi na fakt, że nie ma możliwości edycji danych poprzez widok bazodanowy, zabezpieczamy się w ten sposób przed ryzykiem usunięcia bądź edycji danych oryginalnych. W widoku sami wybieramy zakres prezentowanych danych z tabel, dlatego możemy ukryć przed nieuprawnionymi użytkownikami informacje o charakterze poufnym, które w oryginalnych tabelach mogą być zawarte.

Widoki bazodanowe mogą, podobnie jak tabele, zawierać kolumny geometryczne, przez co można je również wykorzystywać w zapytaniach przestrzennych lub wyświetlać w aplikacjach typu QGIS. Jeżeli w widoku zostanie użyte pole geometryczne z tabeli, w której dla tego pola utworzony jest indeks przestrzenny, indeks ten zadziała również w widoku. Jeżeli jednak wartości w polu geometrycznym w widoku są efektem przekształceń geometrii za pomocą dostępnych w postgis funkcji, taka operacja wykonywana jest za każdym razem w chwili wywołania widoku, nie ma więc możliwości utworzenia dla widoku dedykowanego indeksu przestrzennego. Dlatego jeżeli pozyskanie danych w widoku jest bardzo czasochłonne, czy to

z uwagi na zastosowanie przekształceń geometrii, czy tez innych powodów np. bardzo dużej ilości danych, zamiast standardowego widoku można użyć tzw. widoku zmaterializowanego. **Widok zmaterializowany** tym różni się od zwykłego widoku, że nie jest on już wirtualny, ale jest fizycznie zapisywany na dysku. Odpytanie widoku nie wiąże się z każdorazowym wykonaniem zapytania SQL, ale z odczytem gotowych (przeliczonych) danych. Widok zmaterializowany działa więc znacznie szybciej. Nie prezentuje on jednak aktualnej informacji z bazy danych, ale informacje z ostatniej aktualizacji tego widoku. Ponieważ wyświetlane w widoku zmaterializowanym informacje są już wcześniej pozyskane i zapisane na dysku, dla widoku dostępnych jest wiele narzędzi typowych dla tabel, jak np. indeksy przestrzenne.

Ćwiczenie 22. Sporządzenie wykazu najwyżej położonych budynków w poszczególnych gminach

Pozyskanie informacji o maksymalnej wartości atrybutu liczbowego wykonuje się za pomocą funkcji **MAX**. Funkcję **MAX** podobnie jak inne funkcje statystyczne, można zastosować samodzielnie dla całego zbioru danych i wskaże ona wówczas maksymalną wartość atrybutu w tym zbiorze, pod warunkiem, że odpytujemy bazę tylko o ten jeden atrybut, z którego wartość ma być pozyskana, lub może zostać użyta z wykorzystaniem funkcji grupującej *GROUP BY* i wtedy wskaże ona maksymalną wartość dla każdej grupy danych.

1. W celu pozyskania informacji o maksymalnej wysokości budynku w całym zbiorze danych skorzystamy z tabeli *pzgik.budynki* i wyliczonej wcześniej wartości atrybutu **wysokosc**.

```
select max(wysokosc) wysokosc from pzgik.budynki;
```

Data Output | Explain

	wysokosc numeric
1	144.5

2. Jeżeli chcemy do uzyskanego wyniku dodać jednostkę wysokości [m] musimy zmienić typ wyświetlanego atrybutu na tekstowy i dodać przyrostek [m]

```
select max(wysokosc)::text||' m' wysokosc from pzgik.budynki;
```

Data Output | Explain | M

	wysokosc text
1	144.5 m

3. Aby pozyskać informacje o najwyżej położonym budynku w każdej gminie musimy przygotować sobie odpowiednio dane. W pierwszej kolejności utworzymy tabelę tylko z tymi gminami, dla których w bazie znajdują się budynki. Budynki pochodzą z dwóch powiatów o kodzie teryt 1415 i 1461. Dla nowo utworzonej tabeli utworzymy też indeks przestrzenny.

```
create table pzgik.gminy_wybrane as
select * from pzgik.gminy
where left(teryt,4) in ('1415','1461');

create index gminy_wybrane_idx on pzgik.gminy_wybrane using gist(geom);
```

4. W tym momencie możemy już wyszukać najwyżej położony budynek w każdej gminie z użyciem zapytania przestrzennego ***ST_Intersects*** i posortować wynik od najwyższej wysokości do najniższej.

```
select a.nazwa as "nazwa gminy", max(wysokosc)::text||' m' as "wysokość"
from
pzgik.gminy_wybrane a
join pzgik.budynki b on ST_Intersects(a.geom, b.labelpoint)
group by 1
order by 2 desc;
```

Data Output | Explain | Messages

	nazwa gminy text	wysokość text
1	Łyse	144.5 m
2	Myszyniec	135 m
3	Czarnia	130 m
4	Czerwin	125 m
5	Baranowo	124.5 m
6	Goworowo	123.5 m
7	Kadzidło	122.5 m
8	Troszyn	122 m
9	Olszewo-Borki	115.5 m
10	Lelis	110 m
11	Rzekuń	108 m

Ćwiczenie 23. Sporządzenie listy budynków mieszkalnych położonych w pasie 100 m od linii wysokiego napięcia wraz z odległościami od tych linii oraz zestawienia najmniejszej i średniej odległości tych budynków w rozbiciu na poszczególne gminy.

W ćwiczeniu tym skorzystamy z funkcji **MIN**, która wyszukuje **najniższą** wartość atrybutu w zbiorze danych oraz funkcji **AVG**, która wylicza wartość **średnią**. Zasady korzystania z tych funkcji są identyczne jak z funkcji **MAX**.
Aby poprawnie wykonać ćwiczenie konieczna jest informacja, że zgodnie ze słownikiem funkcji ogólnej budynków z bazy danych BDOT10k, budynki pełniące funkcje mieszkaniowe mają kody:

- 1110 – budynki mieszkalne jednorodzinne
- 1121 – budynki o dwóch mieszkaniach
- 1122 – budynki o trzech i więcej mieszkaniach
- 1130 – budynki zbiorowego zamieszkania.

Pominiemy tu budynki hoteli (1211) i pozostałe budynki zakwaterowania turystycznego (1212). W zakresie linii energetycznych przyjmiemy jak poprzednio linie wysokiego napięcia (WN) oraz najwyższego napięcia (NN). Jako funkcji badającej odległość użyjemy funkcji **postgis** *ST_Distance*. Tym razem funkcję statystyczną MIN użyjemy do wykazu budynków sporządzonych w postaci widoku.

1. W zapytaniu tym skorzystamy z podzapytania, w którym policzymy odległości budynków mieszkalnych posiadających adres od linii wysokiego napięcia w pasie 100m od tych linii. Chcemy znać dokładną funkcję takiego budynku, jego adres oraz odległość od linii wysokiego napięcia.

```
select
a.budynek, a.adres, a.labelpoint, ST_Distance (a.geom, b.geom) odleglosc
from pzgik.budynki a, qgis.linie_energetyczne b
where
b.rodzaj in ('WN', 'NN') and
a.budynek in ('1110','1121','1122','1130') and
a.adres is not null and
st_intersects(a.geom, st_buffer(b.geom,100));
```

	budynek text	adres text	labelpoint geometry	odleglosc double precision
1	1110	Lipniki 136, 07-436 Lipniki	0101000020840800000D86262D49...	80.52756625993169
2	1110	Lipniki 137, 07-436 Lipniki	0101000020840800000077BF272D7...	41.4304942709142
3	1110	Lipniki 135, 07-436 Lipniki	0101000020840800000723834BCA...	51.74361883453036
4	1110	Władysława Reymonta 2A, 07-430 Myszyniec	0101000020840800000056D584273...	88.55662607783637
5	1110	Wykrot 131, 07-430 Wykrot	0101000020840800000030BADFD1F...	77.66550472474356
6	1110	Wykrot 141, 07-430 Wykrot	0101000020840800000F9799797...	70.24924617296574
7	1110	Ostrołęcka 18, 07-437 Łyse	0101000020840800000042C306C2B...	8.522312646766569

Takich budynków mamy w bazie **454**. Pole geometryczne *labelpoint* posłuży nam w kolejnych krokach w tym ćwiczeniu.

2. Mając już zapytanie odnajdujące takie budynki pozostaje nam dodać informację o gminie, w której te budynki się znajdują.

```
select
a.nazwa,
b.*
from
pzgik.gminy_wybrane a
join (select
a.budynek, a.adres, a.labelpoint, ST_Distance (a.geom, b.geom) odleglosc
from pzgik.budynki a, qgis.linie_energetyczne b
where
b.rodzaj in ('WN', 'NN') and
a.budynek in ('1110','1121','1122','1130') and
a.adres is not null and
st_intersects(a.geom, st_buffer(b.geom,100))) b
on ST_Intersects(a.geom, b.labelpoint);
```

nazwa text	budynek text	adres text	labelpoint geometry	odleglosc double precision
Łyse	1110	Lipniki 136, 07-436 Lipniki	010100002084080...	80.52756625993169
Łyse	1110	Lipniki 137, 07-436 Lipniki	010100002084080...	41.4304942709142
Łyse	1110	Lipniki 135, 07-436 Lipniki	010100002084080...	51.74361883453036
Myszyniec	1110	Władysława Reymonta 2A...	010100002084080...	88.55662607783637
Myszyniec	1110	Wykrot 131, 07-430 Wykrot	010100002084080...	77.66550472474356
Myszyniec	1110	Wykrot 141, 07-430 Wykrot	010100002084080...	70.24924617296574
Łyse	1110	Ostrołęcka 18, 07-437 Łyse	010100002084080...	8.522312646766569

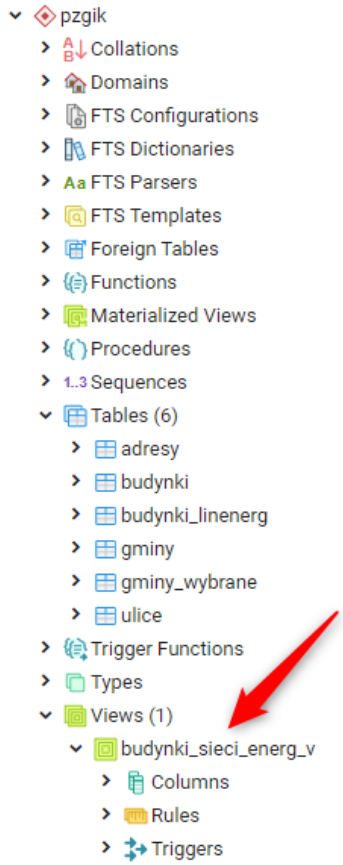
3. Aby takie zestawienie móc utrwalić w strukturach logicznych bazy, utworzymy widok z powyższego zapytania. W widoku chcemy jednak :
   - ponazywać pola korzystając w polskich znaków,
   - zamienić wartości kodowe funkcji budynków na ich opisy,
   - zamiast pola punktowego pola geometrycznego *labelpoint* podać współrzędne w układzie WGS84
   - podać odległość do dwóch miejsc po przecinku z dopiskiem jednostki.

   Powyższe polecenie będzie wyglądać następująco:

```
create or replace view pzgik.budynki_sieci_energ_v as
select
a.nazwa as "Nazwa gminy",
case
```

```
when b.budynek = '1110' then 'budynki mieszkalne jednorodzinne'
when b.budynek = '1121' then 'budynki o dwóch mieszkaniach'
when b.budynek = '1122' then 'budynki o trzech i więcej mieszkaniach'
when b.budynek = '1130' then 'budynki zbiorowego zamieszkania'
else null
end as "Funkcja budynku",
b.adres as "Adres",
round(b.odleglosc::numeric,2)::text||' m' as "Odległość",
st_aslatlontext(st_transform(b.labelpoint, 4326), 'D°M''S" C'::text) AS
"Lokalizacja"
from
pzgik.gminy_wybrane a
join (select
a.budynek, a.adres, a.labelpoint, ST_Distance (a.geom, b.geom) odleglosc
from pzgik.budynki a, qgis.linie_energetyczne b
where
b.rodzaj in ('WN', 'NN') and
a.budynek in ('1110','1121','1122','1130') and
a.adres is not null and
st_intersects(a.geom, st_buffer(b.geom,100))) b
on ST_Intersects(a.geom, b.labelpoint);
```

4. W bazie w schemacie *pzgik* w zakładce *View* pojawił się utworzony widok. Widok dodał się bardzo szybko, ale to dlatego, że w bazie dodała się tylko jego definicja, natomiast nie zostało wykonane zapytanie stanowiące jego treść.



5. Aby zobaczyć zawartość widoku wystarczy wykonać zwykły *SELECT* na bazie danych.

```
select * from pzgik.budynki_sieci_energ_v;
```

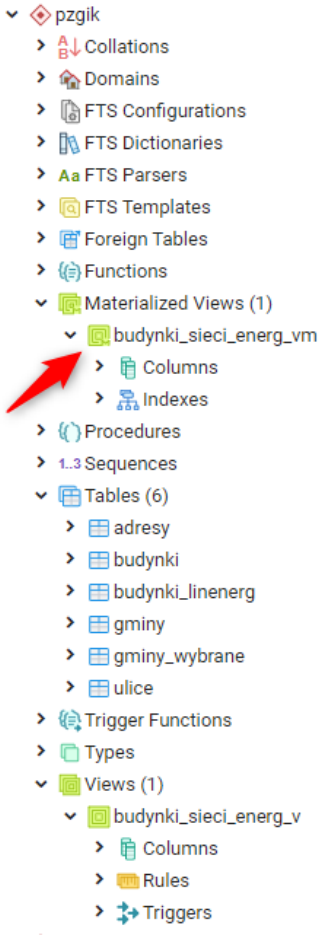
	Nazwa gminy text	Funkcja budynku text	Adres text	Odległość text	Lokalizacja text
1	Łyse	budynki mieszkalne jednorodzinne	Lipniki 136, 07-436 Lipniki	80.53 m	53°21'29" N 21°30'55" E
2	Łyse	budynki mieszkalne jednorodzinne	Lipniki 137, 07-436 Lipniki	41.43 m	53°21'33" N 21°31'8" E
3	Łyse	budynki mieszkalne jednorodzinne	Lipniki 135, 07-436 Lipniki	51.74 m	53°21'25" N 21°30'49" E
4	Myszyniec	budynki mieszkalne jednorodzinne	Władysława Reymonta 2A, 07-430 Myszyniec	88.56 m	53°22'55" N 21°23'13" E
5	Myszyniec	budynki mieszkalne jednorodzinne	Wykrot 131, 07-430 Wykrot	77.67 m	53°21'1" N 21°25'17" E
6	Myszyniec	budynki mieszkalne jednorodzinne	Wykrot 141, 07-430 Wykrot	70.25 m	53°20'48" N 21°25'33" E

Odpytanie powyższego widoku powoduje wykonanie całości zapytania SQL z definicji widoku. Trwa to około **4 sec**.
Jeżeli warstwa budynków lub sieci elektroenergetycznych zostanie zaktualizowana, każdorazowe wyświetlenie wartości z widoku będzie uwzględniało tą aktualizację.

6. Aby przyspieszyć działanie widoku możemy utworzyć widok zmaterializowany.

```
CREATE MATERIALIZED VIEW pzgik.budynki_sieci_energ_vm
AS
select * from pzgik.budynki_sieci_energ_v
WITH DATA;
```

7. W bazie w schemacie *pzgik* w zakładce *Materializad_Views* pojawił się utworzony widok zmaterializowany. Widok dodał się bardzo szybko, ale to dlatego, że w bazie dodała się tylko jego definicja, natomiast nie zostało wykonane zapytanie stanowiące jego treść.



Widok tworzył się dłużej, podobnie jak odpytanie widoku zwykłego, gdyż oprócz utworzenia widoku zmaterializowanego, zostało wykonane zapytanie stanowiące jego treść, a uzyskane dane zapisane.

8. Aby odpytać zawartość widoku zmaterializowanego wykonujemy polecenie *SELECT*.

```
select * from pzgik.budynki_sieci_energ_vm
```

9. Dane zostały wyświetlone w czasie **129 msec**, gdyż nie zostało wykonane pełne zapytanie SQL, tylko odczytane już zapisane dane w widoku zmaterializowanym. Dane w tym widoku pozostaną niezmienione nawet w sytuacji edycji danych na bazie, chyba że widok ten zostanie ręcznie zaktualizowany poleceniem.

```
REFRESH MATERIALIZED VIEW pzgik.budynki_sieci_energ_vm
WITH DATA;
```

10. Aby uzyskać ostatecznie wykaz najmniejszej i średniej odległości budynków mieszkalnych względem sieci elektroenergetycznych w pasie 100m od tych linii dla każdej gminy, skorzystamy z utworzonego widoku zmaterializowanego. Dodatkowo do wyniku dodamy liczbę budynków w pasie 100m od linii dla każdej gminy. Wynik posortujmy po nazwie gminy. Aby wykonać to polecenie musimy dane tekstowe z odległością budynków od linii zamienić na postać liczbową, pozbywając się wcześniej oznaczenia jednostki. Służy do tego celu funkcja *REPLACE*.

```
select
"Nazwa gminy",
min(replace("Odległość",' m','')::numeric)::text||' m' "Min. odległość",
round(avg(replace("Odległość",' m','')::numeric),2)::text "Średna
odległość",
count(*) "Ilość budynków"
from pzgik.budynki_sieci_energ_vm
group by 1
order by 1;
```

	Nazwa gminy text	Min. odległość text	Średna odległość text	Ilość budynków bigint
1	Baranowo	88.35 m	88.35	1
2	Czarnia	97.38 m	97.38	1
3	Czerwin	35.87 m	66.23	9
4	Goworowo	18.74 m	61.16	39
5	Kadzidło	10.63 m	58.17	31
6	Lelis	13.18 m	63.33	62
7	Łyse	8.52 m	60.45	9
8	Myszyniec	6.60 m	60.89	20
9	Olszewo-Borki	11.31 m	58.68	72
10	Rzekuń	6.44 m	59.68	205
11	Troszyn	34.63 m	72.19	5

Ćwiczenie 24. Obliczenie łącznej długości napowietrznych sieci elektroenergetycznych niskiego napięcia dla każdej gminy

Funkcją sumującą wartości wybranych atrybutów jest w PostgreSQL funkcja ***SUM***.
Linie napowietrzne niskiego napięcia oznaczone są w bazie BDOT10k jako **'n/n'**.
W ćwiczeniu tym skorzystamy również z funkcji *ST_Length*, która oblicza długość obiektu liniowego z jego geometrii.

1. Wyliczenie zsumowanej długości linii napowietrznych dla sieci niskiego napięcia dla każdej gminy można wykonać zapytaniem.

```
select a.nazwa, sum(ST_Length(b.geom)) from
pzgik.gminy_wybrane a
join qgis.linie_energetyczne b on ST_Intersects(a.geom, b.geom)
where b.rodzaj = 'n/n'
group by 1;
```

	nazwa text	sum double precision
1	Baranowo	113732.24642893784
2	Czarnia	89987.21898340217
3	Czerwin	81584.5737267648
4	Goworowo	122197.48106962723
5	Kadzidło	227406.7566671446
6	Lelis	128874.47194223819
7	Łyse	277028.6130167468
8	Myszyniec	249276.4465223228
9	Olszewo-Borki	146232.86100597918
10	Ostrołęka	1172.7281783291269
11	Rzekuń	115582.64050358538
12	Troszyn	63804.856626035216

2. Poprawmy widok analizy wyrażając wynik w km, bez miejsc po przecinku oraz nazywając atrybuty wynikowe. Dane posortujmy od rosnąco po długości sieci.

```
select a.nazwa "Nazwa gminy",
round((sum(ST_Length(b.geom))/1000)::numeric,0) "Łączna długość sieci n/n
[km]" from
pzgik.gminy_wybrane a
join qgis.linie_energetyczne b on ST_Intersects(a.geom, b.geom)
where b.rodzaj = 'n/n'
group by 1
order by 2;
```

	Nazwa gminy text	Łączna długość sieci n/n [km] numeric
1	Ostrołęka	1
2	Troszyn	64
3	Czerwin	82
4	Czarnia	90
5	Baranowo	114
6	Rzekuń	116
7	Goworowo	122
8	Lelis	129
9	Olszewo-Borki	146
10	Kadzidło	227
11	Myszyniec	249
12	Łyse	277

Ćwiczenie 25. Obliczenie mediany i średniej powierzchni użytkowej domów jednorodzinnych w poszczególnych gminach

Funkcją odpowiedzialną za wyliczenie **mediany** w bazie PostgreSQL jest ***PERCENTILE_CONT***.
Domy jednorodzinne to te budynki, dla których funkcja ogólna przyjmuje wartość 1110 – budynki mieszkalne jednorodzinne.
Skorzystamy tu również z funkcji postgis, która pobiera informację o powierzchni obiektu poligonowego (budynku) z jego geometrii czyli *ST_Area*.
Przy obliczaniu **powierzchni użytkowej budynków** przyjmujemy założenie, **stanowi ona 80% powierzchni zabudowy pomnożonej przez liczbę kondygnacji budynku.**

1. Powierzchnię użytkową dla wszystkich budynków uzyskujemy poleceniem.

```
select *, ST_Area(geom)*liczbakondygnacji*0.8 pow_uzytk from pzgik.budynki
```

	id [PK] integer	budynek text	liczbakondygnacji integer	geom geometry	pow_uzytk double precision
1	1	1110	1	010300002084080...	79.6596800000553
2	2	1110	2	010300002084080...	168.06607999942554
3	3	1110	2	010300002084080...	170.5285600005263
4	4	1110	2	010300002084080...	202.2413599999299
5	5	1110	3	010300002084080...	262.60415999966597

2. Pozostaje jedynie pogrupować dane po gminach, wyliczyć medianę i średnią oraz sformatować wynik zapytania. Wynik posortujmy rosnąco po wartości mediany oraz wyraźmy w m2, z dokładnością do 1 miejsca po przecinku

```
select
a.nazwa as "Nazwa gminy",
round((PERCENTILE_CONT(.5) WITHIN GROUP(ORDER BY
ST_Area(b.geom)*b.liczbakondygnacji*0.8))::numeric,1) as "Mediana [m2]",
round(AVG((ST_Area(b.geom)*b.liczbakondygnacji*0.8)::numeric),1) as
"Średnia [m2]"
from  pzgik.gminy_wybrane a
join pzgik.budynki b on ST_Intersects(a.geom, b.labelpoint)
group by 1
order by 2;
```

	Nazwa gminy text	Mediana [m2] numeric	Średnia [m2] numeric
1	Goworowo	83.7	111.0
2	Baranowo	86.7	114.8
3	Łyse	89.0	121.1
4	Kadzidło	94.6	122.2
5	Czarnia	95.8	118.1
6	Lelis	97.4	124.6
7	Myszyniec	97.7	124.2
8	Czerwin	101.9	143.7
9	Troszyn	103.4	138.9
10	Olszewo-Borki	105.2	140.8
11	Rzekuń	123.7	162.9

11. Optymalizacja zapytań SQL

Ćwiczenie 26. Obliczenie Indeksy bazodanowe

Tworzenie indeksów przestrzennych

Podstawowym sposobem na optymalizację wykonywania zapytań SQL jest tworzenie indeksów atrybutowych i przestrzennych.
Jak robiliśmy to już wielokrotnie indeksy przestrzenne tworzymy używając polecenia:

```
CREATE INDEX sidx_budynki_geom ON qgis.budynki USING gist(geom);
```

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

```
CREATE INDEX gminy_geom_centr_idx ON qgis.budynki USING
gist(ST_Centroid(geom));
```

Przykład czasu trwania zapytania przestrzennego z istniejącymi indeksami przestrzennymi i bez był prezentowany w ćwiczeniu 14 (Obliczenie ilości budynków w gminach) gdzie dodanie indeksu przestrzennego skróciło czas zapytania z prawie 6 min. do 1 sek.

Tworzenie indeksów przestrzennych

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 qgis.punkty_adresowe;
```

```
Seq Scan on punkty_adresowe  (cost=0.00..667.27 rows=28027 width=87)
```

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 qgis.punkty_adresowe;
```

```
Seq Scan on punkty_adresowe  (cost=0.00..667.27 rows=28027 width=87) (actual
time=0.088..29.162 rows=28027 loops=1)
Planning Time: 0.114 ms
Execution Time: 30.117 ms
```

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.. 667.27 - 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=28027 - 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=87 - Oszacowana liczba bajtów jaką średnio zajmuje jeden rekord.
- drugi nawias oznacza faktyczne parametry:
  - actual time=0.088..29.162 - dwie wartości kosztu - tym razem rzeczywistego
  - rows=28027 - 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 qgis.punkty_adresowe order by simc_nazwa;
```

```
Sort  (cost=2737.70..2807.77 rows=28027 width=87) (actual time=138.485..151.117
rows=28027 loops=1)
  Sort Key: simc_nazwa
  Sort Method: external merge  Disk: 2472kB
  ->  Seq Scan on punkty_adresowe  (cost=0.00..667.27 rows=28027 width=87) (actual
time=0.032..3.274 rows=28027 loops=1)
Planning Time: 0.213 ms
```

```
Execution Time: 153.949 ms
```

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 qgis.punkty_adresowe where simc_nazwa = 'Laskowiec';
```

```
Seq Scan on punkty_adresowe  (cost=0.00..737.34 rows=414 width=87)
  Filter: ((simc_nazwa)::text = 'Laskowiec'::text)
```

Następnie utworzymy indeks który obsłuży warunek *where simc_nazwa = 'Laskowiec'*:

```
create index pzgik_punkty_adresowe_miasto_idx on qgis.punkty_adresowe
(simc_nazwa);
```

Ponownie przeanalizujemy wykonanie zapytania:

```
explain select * from qgis.punkty_adresowe where simc_nazwa = 'Laskowiec';
```

```
Bitmap Heap Scan on punkty_adresowe  (cost=7.50..416.10 rows=414 width=87)
  Recheck Cond: ((simc_nazwa)::text = 'Laskowiec'::text)
  ->  Bitmap Index Scan on pzgik_punkty_adresowe_miasto_idx  (cost=0.00..7.39
rows=414 width=0)
        Index Cond: ((simc_nazwa)::text = 'Laskowiec'::text)
```

Indeks zostanie użyty również przy sortowaniu:

```
explain analyze select * from qgis.punkty_adresowe order by simc_nazwa;
```

```
Index Scan using pzgik_punkty_adresowe_miasto_idx on punkty_adresowe
(cost=0.29..2076.15 rows=28027 width=87) (actual time=0.055..8.462 rows=28027
loops=1)
Planning Time: 0.103 ms
Execution Time: 9.278 ms
```

Zarządzanie indeksami w bazie danych PostgreSQL

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, 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;
```

	relname name	rows_in_bytes text	num_rows real	number_of_indexes bigint	unique text	single_column bigint	multi_column bigint
1	ulice	1020 bytes	1020	3	Y	3	0
2	gminy_wybrane	12 bytes	12	1	N	1	0
3	adresy	27 kB	28027	2	Y	2	0
4	gminy	314 bytes	314	4	Y	4	0
5	budynki_linenerg	399 bytes	399	0	N	0	0
6	budynki	69 kB	70928	5	Y	5	0

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 = 'pzgik'
ORDER BY 1,2;
```

Data Output | Explain | Messages | Notifications

	schemaname name	tablename name	indexname name	num_rows real	table_size text	index_size text	unique text	number_of_scans bigint	tuples_read bigint	tuples_fetched bigint
1	pzgik	adresy	adresy_pkey	28027	5184 kB	1240 kB	Y	0	0	0
2	pzgik	adresy	adresy_geom_idx	28027	5184 kB	2856 kB	N	330	63326	50030
3	pzgik	budynki	budynki_cent_idx	70928	29 MB	3984 kB	N	85	826101	826101
4	pzgik	budynki	budynki_pkey	70928	29 MB	3128 kB	Y	1	100	100
5	pzgik	budynki	budynki_cent_idx	70928	29 MB	3984 kB	N	85	826101	826101
6	pzgik	budynki	budynki_pkey	70928	29 MB	3128 kB	Y	1	100	100
7	pzgik	budynki_linenerg	[null]	399	96 kB	[null]	N	[null]	[null]	[null]
8	pzgik	gminy	gminy_pkey	314	64 kB	16 kB	Y	0	0	0
9	pzgik	gminy	gminy_geom_idx	314	64 kB	32 kB	N	82	12	12
10	pzgik	gminy	gminy_pkey	314	64 kB	16 kB	Y	0	0	0
11	pzgik	gminy	gminy_geom_idx	314	64 kB	32 kB	N	82	12	12
12	pzgik	gminy_wybrane	gminy_wybrane...	12	8192 bytes	8192 bytes	N	55385	91234	91234
13	pzgik	ulice	ulice_pkey	1020	264 kB	40 kB	Y	0	0	0
14	pzgik	ulice	ulice_pkey	1020	264 kB	40 kB	Y	0	0	0

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;
```

12. Usuwanie danych z bazy

Ćwiczenie 27. Usuwanie rekordów z tabeli

Poleceniem, które umożliwia usunięcie wybranych rekordów z tabeli jest polecenie **DELETE**

1. Z tabeli *pzgik.budynki_linenerg* usuń wszystkie rekordy o funkcji budynku = '**1271**'

```
delete from pzgik.budynki_linenerg where budynek = '1271';
```

```
DELETE 176
Query returned successfully in 97 msec.
```

2. Jeżeli w tym samym poleceniu nie zastosujemy warunku *WHERE*, zostaną usunięte wszystkie wartości z tabeli.

```
delete from pzgik.budynki_linenerg;
```

```
DELETE 223
Query returned successfully in 95 msec.
```

3. Możemy to sprawdzić zliczając liczbę obiektów w tabeli.

```
select count(*) from pzgik.budynki_linenerg;
```

	count bigint
1	0

4. Polecenie *DELETE FROM* bez warunku *WHERE* usuwa co prawda wszystkie rekordy w tabeli, ale usuwane wiersze są rejestrowane w tabeli. Ostatecznie więc usunięte wiersze nadal obciążają bazę pojemnościowo, a sam proces usuwania dużej ilości rekordów może być bardzo czasochłonny. Dlatego jeżeli zależy nam na szybkim i całkowitym usunięciu rekordów z tabeli, zamiast *DELETE* używamy polecenia **TRUNCATE**. *TRUNCATE* jest używane do szybkiego i całkowitego usunięcia danych w tabeli. Proces ten jest szybki ale nieodwracalny.

```
truncate table pzgik.ulice
```

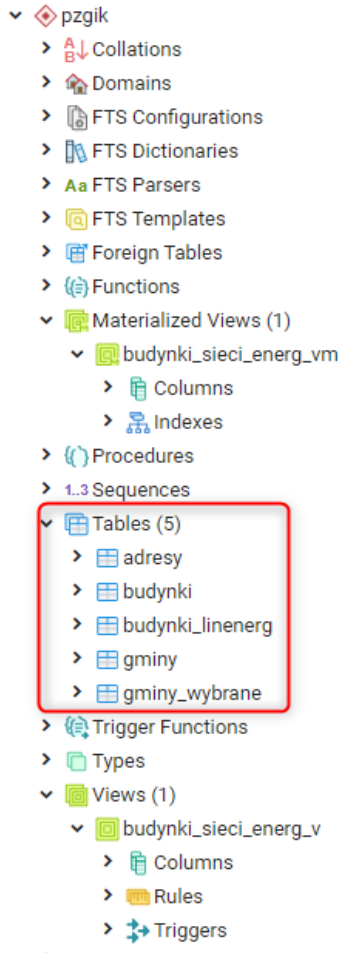
Ćwiczenie 28. Usuwanie tabel i widoków

Poleceniem, które umożliwia usunięcie całych tabel lub widoków jest polecenie **DROP**

1. Usuń pustą tabelę *pzgik.ulice*.

```
drop table pzgik.ulice
```

2. Po odświeżeniu zawartości tabel w schemacie *pzgik* tabeli *ulice* już nie ma.



3. Usuń tabelę *budynki* ze schematu *pzgik*.

```
drop table pzgik.budynki;
```

```
ERROR: cannot drop table pzgik.budynki because other objects depend on it DETAIL:
view pzgik.budynki_sieci_energ_v depends on table pzgik.budynki materialized view
pzgik.budynki_sieci_energ_vm depends on view pzgik.budynki_sieci_energ_v HINT: Use
DROP ... CASCADE to drop the dependent objects too. SQL state: 2BP01
```

4. Tabela nie została usunięta, ponieważ jest ona używana do utworzenia widoku *pzgik.budynki_sieci_energ_v*, a ten z kolei do utworzenia widoku zmaterializowanego

pzgik.budynki_sieci_energ_vm. Dzieję się tak podobnie w przypadku tabel powiązanych ze sobą kluczami. Aby usunąć tabelę *pzgik.budynki* konieczne jest wielokrotne zastosowanie polecenie DROP w odpowiedniej kolejności, czyli najpierw dla widoku *pzgik.budynki_sieci_energ_vm*, potem dla widoku *pzgik.budynki_sieci_energ_v*, a dopiero na końcu dla tabeli *pzgik.budynki*. Komend tych nie będziemy wykonywać, ale wyglądałoby to następująco:

```
DROP MATERIALIZED VIEW pzgik.budynki_sieci_energ_vm;
DROP VIEW pzgik.budynki_sieci_energ_v;
DROP TABLE pzgik.budynki;
```

5. Powyższa ścieżka postępowania jest zalecana, gdyż jesteśmy świadomi usuwania wszystkich kolejnych obiektów w bazie danych. Istnieje jednak polecenie kasujące obiekt docelowy i wszystkie powiązane z nim obiekty.

```
DROP TABLE pzgik.budynki CASCADE;
```

Wszystkie obiekty zostały usunięte i otrzymaliśmy komunikat:

```
NOTICE: drop cascades to 2 other objects DETAIL: drop cascades to view
pzgik.budynki_sieci_energ_v drop cascades to materialized view
pzgik.budynki_sieci_energ_vm DROP TABLE Query returned successfully in 133 msec.
```

6. Zniknęła zarówno tabela *pzgik.budynki* jak i powiązane z nią widoki.