Język SQL w bazie PostgreSQL
(poziom zaawansowany)

Michał Mackiewicz
www.gis-support.pl

MINISTERSTWO
ŚRODOWISKA

Sfinansowano ze środków
Narodowego Funduszu
Ochrony Środowiska
i Gospodarki Wodnej

Agenda

1. Wizualizacja i analiza danych
2. Indeksy - rodzaje, zastosowanie, ocena wydajności i celowości użycia
3. Filtrowanie danych według atrybutów i według lokalizacji
4. Przetwarzanie danych: generalizacja, buforowanie, transformacja układu, scalanie, docinanie, obliczanie powierzchni i obwodu
5. Zarządzanie uprawnieniami w bazie danych, konta użytkowników i ich uprawnienia, zasady edycji w wiele osób
6. Pojęcie transakcji, zaawansowana transakcyjność
7. Manipulowanie dużymi zbiorami danych
8. Zarządzanie obiektami bazy danych
9. Generowanie raportów z użyciem zaawansowanych funkcji grupujących
10. Zaawansowane techniki pozyskiwania danych z użyciem podzapytań oraz funkcji analitycznych
11. Wyrażenia regularne
12. Klauzula WITH i zapytania hierarchiczne
13. Instrukcja warunkowa CASE … THEN … ELSE
14. Widoki, ich typy i zastosowanie

Dane

W trakcie szkolenia będą wykorzystane w większości realne dane pochodzące z rejestrów publicznych:

Geoserwis GDOŚ
Bank Danych o Lasach
Państwowy Rejestr Granic
Baza Danych Ogólnogeograficznych

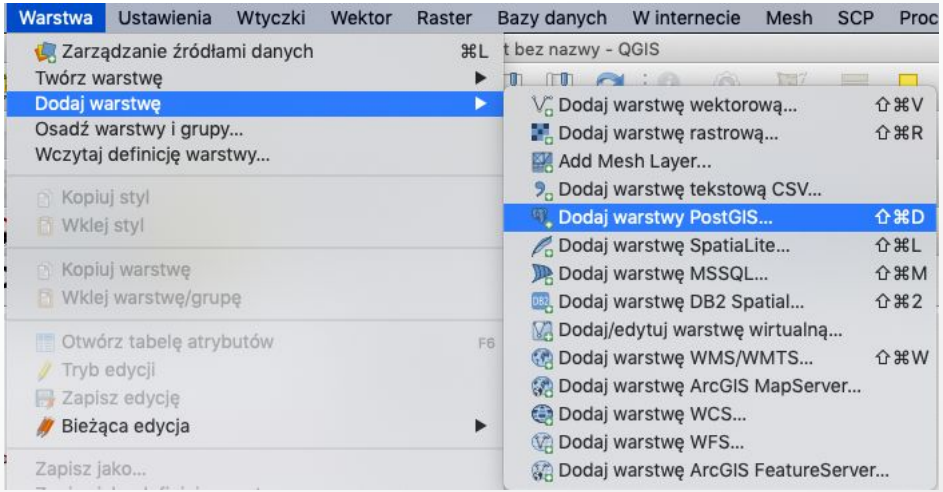
Wizualizacja i analiza danych

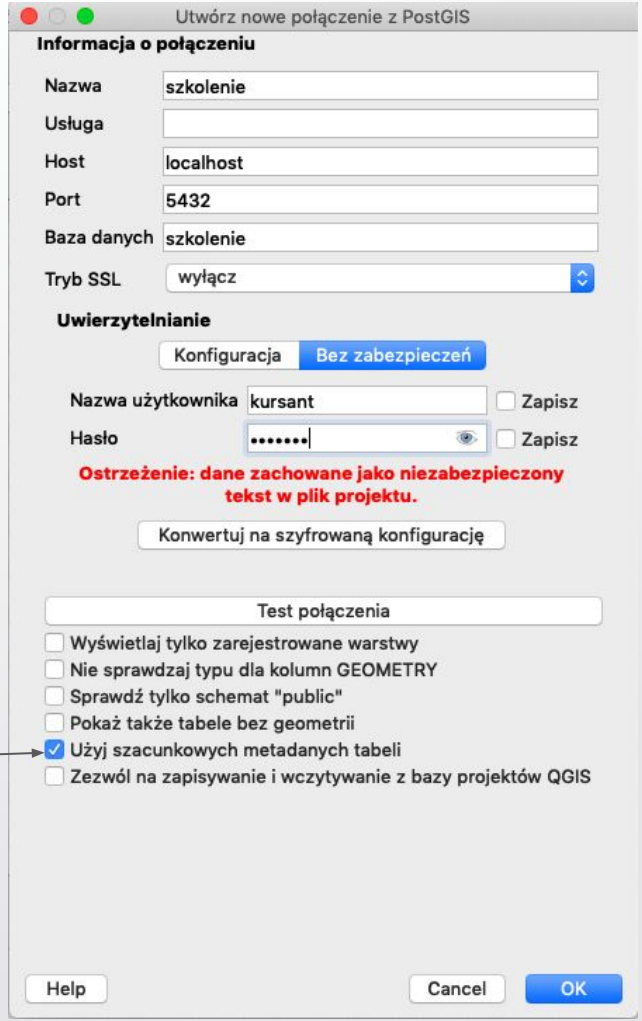
Najbardziej wszechstronnym oprogramowaniem Open Source do wizualizacji danych w bazie PostgreSQL / PostGIS jest obecnie **QGIS.**

- **Mapy** (w tym kartogramy i kartodiagramy)
- **Wykresy**
- **Tabele**

Funkcję wizualizacji danych przestrzennych posiada również pgAdmin 4.

Podłączenie do QGIS



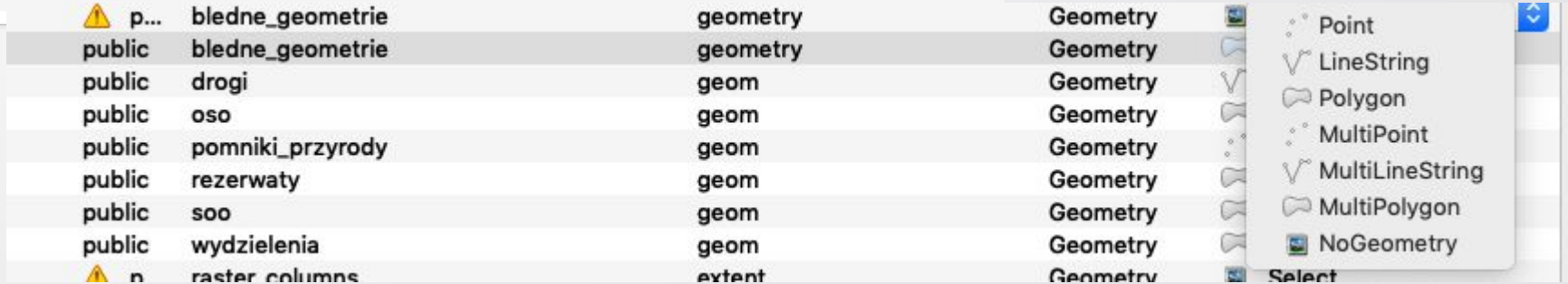


Opcja istotna przy dużych zbiorach danych

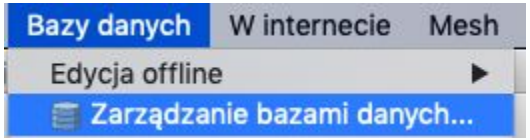
Wizualizacja tabeli

Schemat	Tabela	Kc	Kolumna	Typ danych	Dane przestrzenne
▶ informati...					
▶ pg_catalog					
▼ public					
⚠ p...	bledne_geometrie		geometry	Geometry	Select...
public	bledne_geometrie		geometry	Geometry	Polygon
public	drogi		geom	Geometry	LineString
public	oso		geom	Geometry	MultiPolygon
public	pomniki_przyrody		geom	Geometry	Point
public	rezerwaty		geom	Geometry	MultiPolygon
public	soo		geom	Geometry	MultiPolygon
public	wydzielenia		geom	Geometry	MultiPolygon
⚠ p...	raster_columns		extent	Geometry	Select...
public	gatunki			brak	NoGeometry
public	siedliska			brak	NoGeometry
public	formula_e			brak	NoGeometry
public	cdma			brak	NoGeometry
public	systematyka			brak	NoGeometry
⚠ p...	raster_overviews			brak	NoGeometry
public	spatial_ref_sys			brak	NoGeometry
⚠ p...	geometry_columns			brak	NoGeometry
⚠ p...	geography_columns			brak	NoGeometry

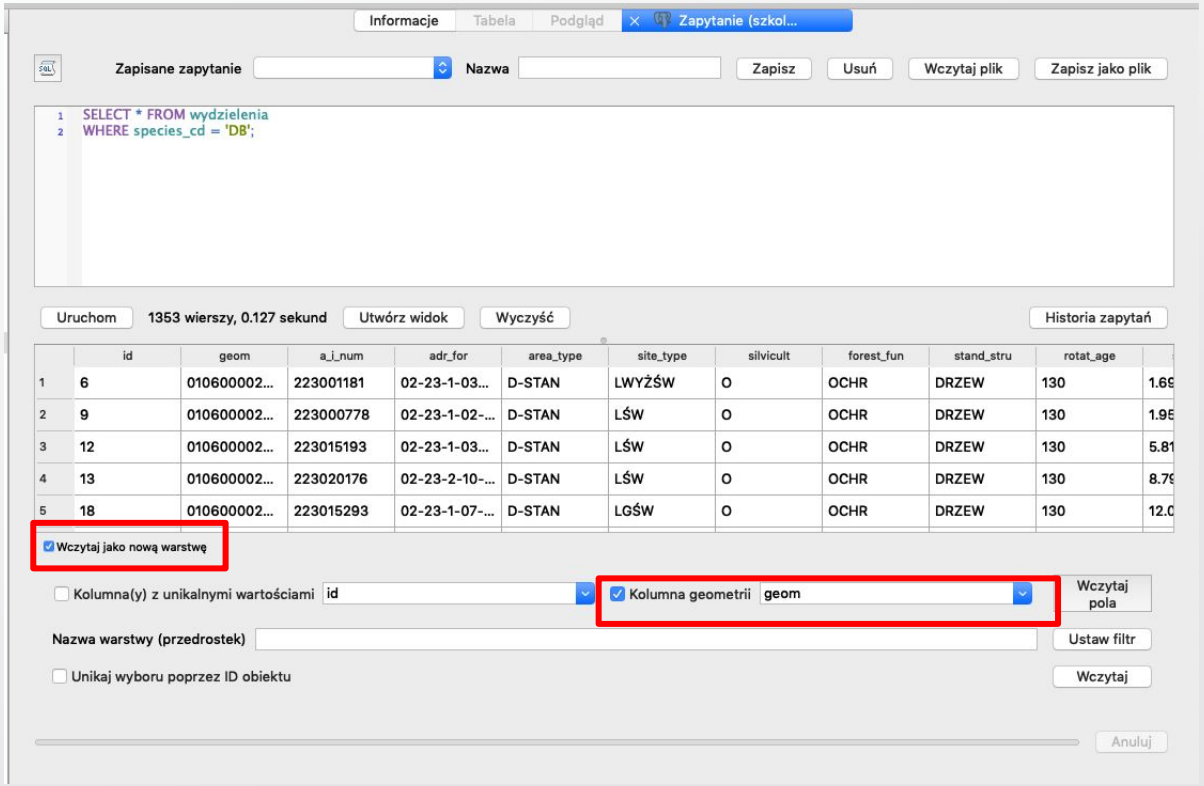
Jeżeli tabela ma dowolny typ geometrii, QGIS wykryje istniejące geometrie, a także pozwoli na wybór typu - w celu dodania obiektów innego typu niż istniejące już w tabeli.



Wizualizacja zapytania



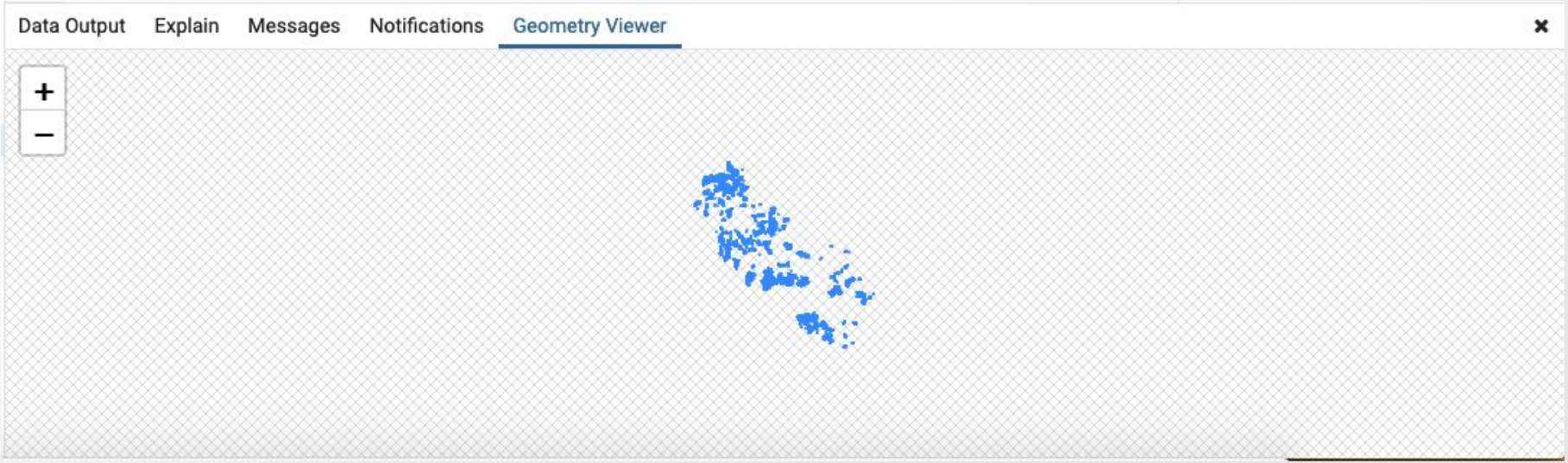
Do wizualizacji wyników zapytań służy narzędzie Zarządzanie bazami danych. Po wyborze połączenia należy otworzyć Okno SQL. Po wykonaniu zapytania zaznaczyć "Wczytaj jako nową warstwę". Narzędzie umożliwia wybór kolumny z geometrią. Jeśli nazwa kolumny to "geom", "the_geom" lub "geometry", zostanie ustawiona automatycznie (z możliwością zmiany)



Wizualizacja pgAdmin

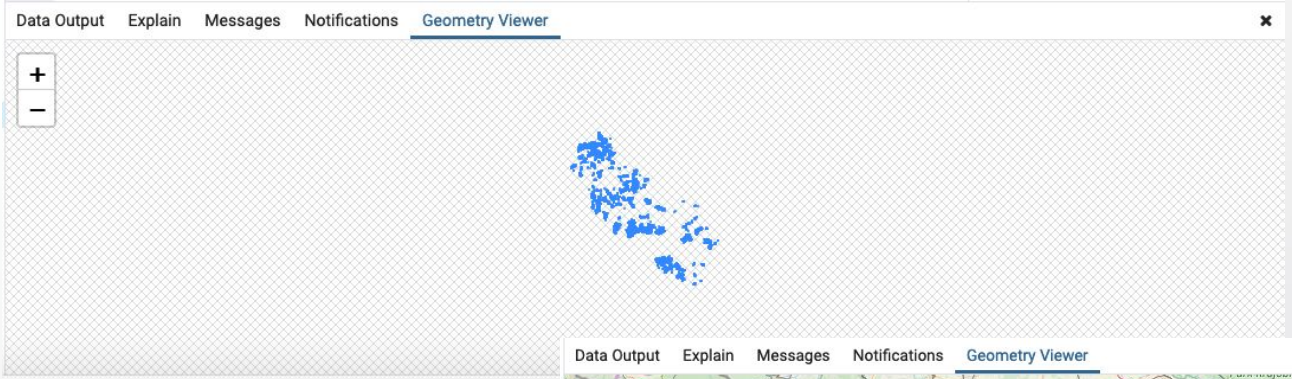
pgAdmin 4 posiada funkcję Geometry Viewer. Jeśli w wyniku zapytania znajduje się kolumna typu GEOMETRY, przy jej nazwie pojawi się ikona 

Po jej kliknięciu pokaże się mapa.



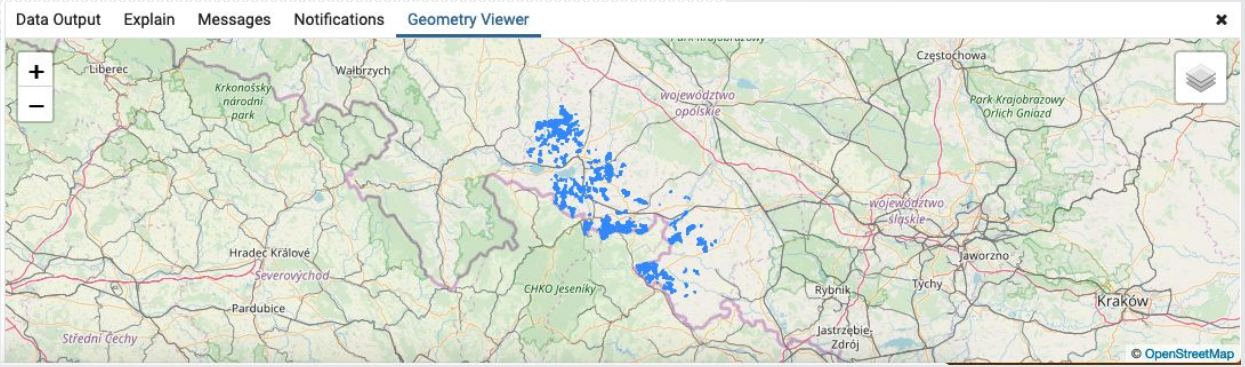
Wizualizacja pgAdmin

Dane przestrzenne w pgAdmin w układzie WGS84 (SRID=4326) pokazują się na podkładzie OpenStreetMap Standard, natomiast w innych układach - bez podkładu mapowego.



SELECT adr_for, geom
FROM wydzielenia;

SELECT adr_for,
ST_Transform(geom,4326)
FROM wydzielenia;



Wstępna analiza

Zapytania do poznania podstawowych informacji o danych:

```
SELECT count(*) FROM wydzielenia;
--liczba obiektów

SELECT column_name, data_type FROM information_schema.columns WHERE table_name = 'wydzielenia';
-- nazwy i typy kolumn

SELECT * FROM geometry_columns WHERE table_name = 'wydzielenia';
-- informacje o kolumnie geometrii (deklarowane)

SELECT DISTINCT ST_GeometryType(geom), ST_SRID(geom) FROM wydzielenia;
-- typ geometrii i układ współrzędnych (rzeczywiste)
```

Indeksy - rodzaje, zastosowanie, ocena wydajności i celowości użycia

Indeksy

- Podstawowym narzędziem do przyspieszenia zapytań w bazie danych są indeksy
- Indeks jest pomocniczą strukturą danych, zmniejsza zapotrzebowanie na moc obliczeniową kosztem zajęcia miejsca na dysku
- PostgreSQL dysponuje różnymi typami indeksów, dostosowanymi do różnego rodzaju danych

Narzędzie EXPLAIN

Korzystając z indeksów, warto weryfikować skuteczność ich działania przy pomocy EXPLAIN.
Narzędzie EXPLAIN służy do oceny *planu zapytania.*
Plan zapytania jest sekwencją niskopoziomowych funkcji (skanowanie tabeli, filtrowanie, łączenie, scalanie) wykorzystywanych przez system PostgreSQL do realizacji zapytania. Każda operacja posiada przypisany określony koszt (w abstrakcyjnych jednostkach), system może analizować kilka alternatywnych planów zapytania w celu wybrania tego o najniższym koszcie.

Wykorzystanie EXPLAIN polega na dodaniu komendy EXPLAIN przed treścią zapytania i wykonaniu takiej komendy w oknie SQL klienta bazy danych (psql, pgAdmin, QGIS).

Jak czytać EXPLAIN

EXPLAIN SELECT * FROM miejscowosci WHERE gmina='Ujazd-obszar wiejski' ORDER BY naz_glowna DESC;

```
QUERY PLAN
Sort  (cost=10439.11..10439.20 rows=39 width=5441)
  Sort Key: naz_glowna DESC
  ->  Seq Scan on miejscowosci  (cost=0.00..10438.08 rows=39 width=5441)
        Filter: ((gmina)::text = 'Ujazd-obszar wiejski'::text)
```

- cost - koszt wykonania zapytania (w abstrakcyjnych jednostkach)
- width - liczba bajtów zwróconych na danym etapie zapytania
- Sort - oznacza operację sortowania
- Seq Scan - oznacza operację przejścia przez całą tabelę
- Filter - oznacza operację filtrowania, za znakiem : jest podana interpretacja klauzuli WHERE z rozpoznanymi typami danych

EXPLAIN z indeksem

EXPLAIN SELECT * FROM miejscowosci WHERE id = 100134;

```
QUERY PLAN
Index Scan using miejscowosci_pkey on miejscowosci  (cost=0.42..8.44 rows=1 width=5441)
  Index Cond: (id = 100134)
```

Gdy jedynym kryterium poszukiwania jest wartość kolumny z indeksem - wystarczy sam indeks, bez skanowania tabeli.

EXPLAIN SELECT * FROM miejscowosci WHERE powiat='strzelecki' AND gmina='Ujazd-obszar wiejski';

```
QUERY PLAN
Bitmap Heap Scan on miejscowosci  (cost=5.97..735.63 rows=1 width=5441)
  Recheck Cond: ((powiat)::text = 'strzelecki'::text)
  Filter: ((gmina)::text = 'Ujazd-obszar wiejski'::text)
  ->  Bitmap Index Scan on miejscowosci_powiat_idx  (cost=0.00..5.97 rows=207 width=0)
        Index Cond: ((powiat)::text = 'Strzelecki'::text)
```

W razie łączenia różnych kryteriów - kolumn z indeksem i bez - PostgreSQL tworzy w pamięci pomocniczą strukturę danych (bitmapę) w której oznacza wiersze potencjalnie pasujące do zapytania: kolumna powiat posiada indeks, a gmina nie.

EXPLAIN i pomiary

- EXPLAIN - tylko przygotowanie planu zapytania
- EXPLAIN ANALYZE - przygotowanie planu i pomiar czasu
- EXPLAIN (ANALYZE, BUFFERS) - przygotowanie planu, pomiar czasu i określenie ile danych znajduje się w pamięci, a ile na dysku

EXPLAIN ANALYZE

EXPLAIN ANALYZE SELECT * FROM miejscowosci WHERE powiat='strzelecki' AND gmina='Ujazd-obszar wiejski';

```
QUERY PLAN
Bitmap Heap Scan on miejscowosci  (cost=5.97..735.63 rows=1 width=5441) (actual
time=0.206..0.816 rows=24 loops=1)
  Recheck Cond: ((powiat)::text = 'strzelecki'::text)
  Filter: ((gmina)::text = 'Ujazd-obszar wiejski'::text)
  Rows Removed by Filter: 170
  Heap Blocks: exact=162
  ->  Bitmap Index Scan on miejscowosci_powiat_idx  (cost=0.00..5.97 rows=207 width=0)
(actual time=0.160..0.160 rows=194 loops=1)
        Index Cond: ((powiat)::text = 'strzelecki'::text)
Planning time: 0.527 ms
Execution time: 1.118 ms
```

actual time - zmierzony czas na wykonanie etapu
loops - zmierzona liczba powtórzeń etapu
Rows Removed by Filter - liczba wierszy odrzuconych na etapie filtrowania
Heap Blocks - liczba użytych 8kB bloków pamięci
Planning time - zmierzony czas na ułożenie planu
Execution time - zmierzony czas na wykonanie zapytania (bez pobrania wyników)

EXPLAIN BUFFERS

Opcja BUFFERS może być użyta wyłącznie razem z ANALYZE.

EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM miejscowosci WHERE powiat='strzelecki' AND gmina='Ujazd-obszar wiejski';

```
QUERY PLAN
Bitmap Heap Scan on miejscowosci  (cost=5.97..735.63 rows=1 width=5441) (actual
time=0.167..0.704 rows=24 loops=1)
  Recheck Cond: ((powiat)::text = 'strzelecki'::text)
  Filter: ((gmina)::text = 'Ujazd-obszar wiejski'::text)
  Rows Removed by Filter: 170
  Heap Blocks: exact=162
  Buffers: shared hit=166
  ->  Bitmap Index Scan on miejscowosci_powiat_idx  (cost=0.00..5.97 rows=207 width=0)
(actual time=0.122..0.122 rows=194 loops=1)
        Index Cond: ((powiat)::text = 'strzelecki'::text)
        Buffers: shared hit=4
Planning time: 0.488 ms
Execution time: 0.854 ms
```

Buffers shared hit - odczytanie danych z pamięci (szybsze)
Buffers shared read - odczytanie danych z dysku (wolniejsze)
Administrator bazy może podnieść wielkość parametru shared_buffers w postgresql.conf, by zwiększyć bufor w pamięci i przyspieszyć niektóre zapytania.

Utworzenie indeksu

- Podstawowy typ indeksu w PostgreSQL to indeks BTREE
- Nadaje się do wszystkich danych, które mogą być logicznie posortowane według jednego wymiaru, np. tekst, liczby, daty
- Działa z operatorami =, >, >=, <, <=
- Minimalna komenda do utworzenia indeksu:

  CREATE INDEX ON miejscowosci(gmina);

  Domyślnie nadawana nazwa indeksu: <nazwa tabeli>_<nazwa kolumny>_idx

  Indeksowi można nadać inną nazwę:

CREATE INDEX nazwa_glowna_indeks **ON** miejscowosci(naz_glowna);

W programach pgAdmin i QGIS prawidłowym wynikiem takiego zapytania jest 0 wierszy. psql odpowiada komunikatem "CREATE INDEX".

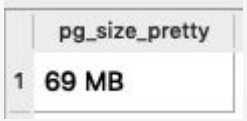
Rozmiar na dysku

```
SELECT pg_size_pretty(pg_relation_size('nazwa_glowna_indeks'));
```

	pg_size_pretty
1	3664 kB

dla porównania z rozmiarem danych w tabeli:

```
SELECT pg_size_pretty(pg_relation_size('miejscowosci'));
```



	pg_size_pretty
1	69 MB

Funkcja pg_relation_size zwraca rozmiar relacji w bajtach. Zastosowanie dodatkowo funkcji pg_size_pretty formatuje wynik do odpowiedniej jednostki (kB, MB, GB, TB)

EXPLAIN ANALYZE SELECT * FROM miejscowosci WHERE naz_glowna LIKE 'Wąch%';

```
QUERY PLAN
Seq Scan on miejscowosci  (cost=0.00..10438.08 rows=11 width=5441) (actual
time=101.243..121.504 rows=7 loops=1)
  Filter: ((naz_glowna)::text ~~ 'Wąch%'::text)
  Rows Removed by Filter: 124639
Planning time: 0.282 ms
Execution time: 121.538 ms
```

Seq Scan oznacza, że indeks nie został użyty, nastąpiło skanowanie całej tabeli.

Indeks typu BTree bez modyfikacji **nie działa** z operatorem LIKE.

Wyszukiwanie przybliżone

```
CREATE INDEX miejscowosci_like ON miejscowosci(naz_glowna varchar_pattern_ops);
EXPLAIN ANALYZE SELECT * FROM miejscowosci WHERE naz_glowna LIKE 'Wąch%';
QUERY PLAN
Index Scan using miejscowosci_like on miejscowosci (cost=0.42..8.44 rows=11
width=5441) (actual time=2.040..2.050 rows=7 loops=1)
   Index Cond: (((naz_glowna)::text ~>=~ 'Wąch'::text) AND ((naz_glowna)::text ~<~
'Wąci'::text))
  Filter: ((naz_glowna)::text ~~ 'Wąch%'::text)
Planning time: 7.931 ms
Execution time: 4.028 ms
EXPLAIN ANALYZE SELECT * FROM miejscowosci WHERE naz_glowna LIKE '%ąch%';
QUERY PLAN
Seq Scan on miejscowosci (cost=0.00..10438.08 rows=11 width=5441) (actual
time=18.206..53.301 rows=33 loops=1)
  Filter: ((naz_glowna)::text ~~ '%ąch%'::text)
  Rows Removed by Filter: 124613
Planning time: 0.444 ms
Execution time: 53.353 ms
```

Wyszukiwanie przybliżone

Dla wyszukiwania **początkowego** fragmentu tekstu wystarczy indeks z operatorem varchar_pattern_ops (lub text_pattern_ops)

Dla wyszukiwania **dowolnego** fragmentu tekstu konieczne jest zastosowanie dodatkowego rozszerzenia - pg_trgm oraz indeksu GiST z modyfikacją. Aktywację rozszerzenia w bazie musi wykonać administrator komendą CREATE EXTENSION IF NOT EXISTS pg_trgm; - w bazie ćwiczeniowej już jest to wykonane.

```
CREATE INDEX miejscowosci_like_2 ON miejscowosci USING gist(naz_glowna gist_trgm_ops);
EXPLAIN ANALYZE SELECT * FROM miejscowosci WHERE naz_glowna LIKE '%ąch%';
QUERY PLAN
Bitmap Heap Scan on miejscowosci  (cost=4.37..47.34 rows=11 width=5441) (actual time=11.826..11.870 rows=33 loops=1)
  Recheck Cond: ((naz_glowna)::text ~~ '%ąch%'::text)
  Heap Blocks: exact=18
   ->  Bitmap Index Scan on miejscowosci_like_2  (cost=0.00..4.36 rows=11 width=0) (actual time=11.774..11.774 rows=33 loops=1)
        Index Cond: ((naz_glowna)::text ~~ '%ąch%'::text)
Planning time: 0.599 ms
Execution time: 11.961 ms
```

Indeks złożony

W przypadku zapytań wykorzystujących wiele kolumn jednocześnie warto rozważyć utworzenie indeksu złożonego (compound index) - wówczas nie ma potrzeby tworzenia bitmapy.
Przykład: województwo, powiat i gmina

CREATE INDEX ON miejscowosci(woj,powiat,gmina);
EXPLAIN SELECT * FROM miejscowosci WHERE woj='opolskie' AND powiat='strzelecki' AND gmina='Ujazd-obszar wiejski';

```
QUERY PLAN
Index Scan using miejscowosci_woj_powiat_gmina_idx on miejscowosci  (cost=0.42..8.44 rows=1 width=5441)
  Index Cond: (((woj)::text = 'opolskie'::text) AND ((powiat)::text = 'strzelecki'::text) AND ((gmina)::text = 'Ujazd-obszar wiejski'::text))
```

Indeks częściowy

Możliwe jest utworzenie indeksu obejmującego tylko wybrane wiersze:

```
CREATE INDEX ON miejscowosci WHERE rodzaj_obi = 'wieś';
```

Zastosowanie - dla zapytań obejmujących powtarzalne, konkretne wartości

Indeks przestrzenny

- Struktura danych umożliwiająca szybkie odnalezienie odpowiedzi na pytanie "czy dwa prostokąty się przecinają"
- Przez wiele narzędzi do importu danych tworzony jest automatycznie
- QGIS - opcja "stwórz go" w informacjach o tabeli
- SQL:

  CREATE INDEX ON pomniki_przyrody USING GIST(geom);

Bez indeksu

EXPLAIN ANALYZE SELECT a.* FROM miejscowosci a, wydzielenia_iglaste b WHERE ST_Intersects(a.geom, b.geom);

```
QUERY PLAN
Gather  (cost=1000.00..16166232.08 rows=1012 width=5441) (actual time=24102.465..24102.465
rows=0 loops=1)
  Workers Planned: 2
  Workers Launched: 2
      ->  Nested Loop  (cost=0.00..16165130.88  rows=422  width=5441)  (actual
time=24080.743..24080.743 rows=0 loops=3)
        Join Filter: ((a.geom && b.geom) AND _st_intersects(a.geom, b.geom))
        Rows Removed by Join Filter: 37061411
         ->  Parallel Seq Scan on miejscowosci a  (cost=0.00..9399.36 rows=51936 width=5441)
(actual time=0.047..53.541 rows=41549 loops=3)
         ->  Seq Scan on wydzielenia_iglaste b  (cost=0.00..76.92 rows=892 width=438) (actual
time=0.002..0.215 rows=892 loops=124646)
Planning time: 0.479 ms
Execution time: 24107.560 ms
```

Z indeksem

EXPLAIN ANALYZE SELECT a.* FROM miejscowosci a, wydzielenia b WHERE ST_Intersects(a.geom, b.geom);

```
QUERY PLAN
Gather  (cost=1000.15..33129.66 rows=1125 width=5441) (actual time=320.318..377.612 rows=6
loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Nested Loop  (cost=0.15..32017.16 rows=469 width=5441) (actual time=171.457..350.978
rows=2 loops=3)
        ->  Parallel Seq Scan on miejscowosci a  (cost=0.00..9399.36 rows=51936 width=5441)
(actual time=0.068..42.366 rows=41549 loops=3)
        ->  Index Scan using sidx_wydzielenia_geom on wydzielenia b  (cost=0.15..0.43 rows=1
width=437) (actual time=0.007..0.007 rows=0 loops=124646)
              Index Cond: (a.geom && geom)
              Filter: _st_intersects(a.geom, geom)
              Rows Removed by Filter: 0
Planning time: 0.742 ms
Execution time: 382.368 ms
```

Kiedy tworzyć indeksy?

- Dla kolumn, które będą używane w klauzuli WHERE
- Dla kolumn, które będą używane w klauzuli ORDER BY
- Dla kolumny z kluczem głównym - do edycji
- Indeks przestrzenny dla danych GIS przydaje się praktycznie zawsze, choćby dla sprawnego wyświetlania danych w QGIS. Jest absolutnie konieczny jeśli chcemy prowadzić analizy przestrzenne w SQL oraz publikować dane za pomocą usług sieciowych (WMS, WFS).

Kiedy nie tworzyć indeksu?

- Dla wszystkich kolumn "na zapas"
- Dla małych tabel (do ok. 1000 wierszy) - **nie dotyczy indeksu przestrzennego**

Usunięcie indeksu

- Jeśli EXPLAIN ANALYZE nie wykazało zysku czasowego
- Dla zwolnienia miejsca na dysku
- Przed masowym importem danych

```
SELECT pg_size_pretty(pg_relation_size('miejscowosci_like'));
-- ile miejsca zajmuje indeks?
DROP INDEX miejscowosci_like;
```

Filtrowanie danych według atrybutów i według lokalizacji

Filtrowanie danych według atrybutów

- Sposób 1: użycie funkcji Filtr w QGIS - podanie samej klauzuli WHERE
- Sposób 2: utworzenie zapytania poprzez Zarządzanie bazami danych

Filtrowanie

- W klauzuli WHERE można używać operatorów: =, !=, >, >=, <, <=, IN (lista wartości), LIKE (wyszukiwanie przybliżone)

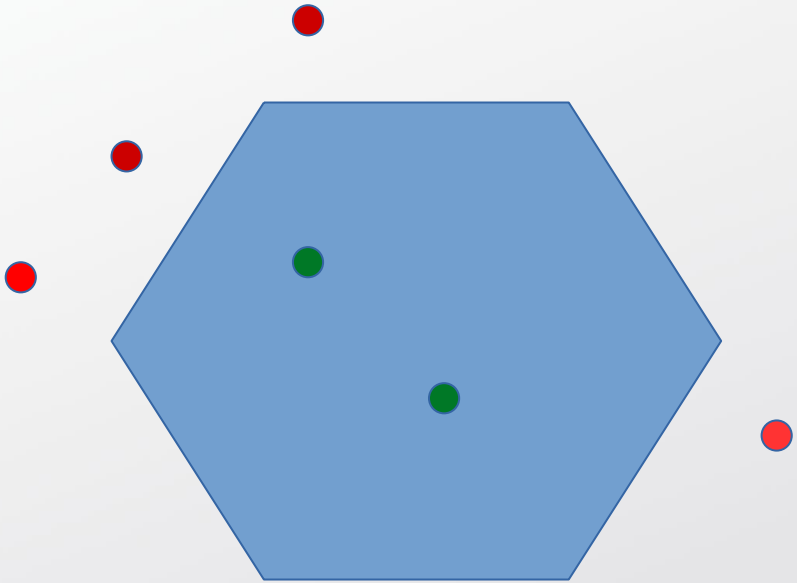
- Wymienione operatory nie działają dla wartości NULL - zawsze zwrócą fałsz. Do porównania z wartością NULL trzeba stosować specjalne operatory IS NULL oraz IS NOT NULL.

- Warunki można łączyć wykorzystując operatory logiczne: AND, OR, NOT
- Wartości numeryczne podajemy bez cudzysłowu
- Wartości tekstowe i inne podajemy w pojedynczym cudzysłowie
- Nazwy kolumn, tabel, schematów podaje się co do zasady w podwójnym cudzysłowie (chyba, że nie zaczynają się od cyfry, zawierają same małe litery, cyfry i znak _, wówczas można cudzysłów pominąć)

Filtrowanie wg lokalizacji

- Filtrowanie według lokalizacji jest realizowane przez funkcje relacji przestrzennych
- Funkcje zostały zdefiniowane w normie ISO 19125-2 „Geographic information – Simple feature access – Part 2: SQL option i standardzie OpenGIS
- Funkcje relacji przestrzennych biorą 2 argumenty typu GEOMETRY
- Zwracają wartość BOOLEAN (TRUE / FALSE)
- Przykład klauzuli: **WHERE** ST_Intersects(a.geom,b.geom);
- Gdy poszukiwane jest przeciwieństwo: **WHERE** ST_Overlaps(a.geom,b.geom) = FALSE;

SELECT a.nazwa AS nazwa_rezerwatu, b.nazwa AS nazwa_natura FROM rezerwaty a, soo b WHERE ST_Within(a.geom, b.geom);

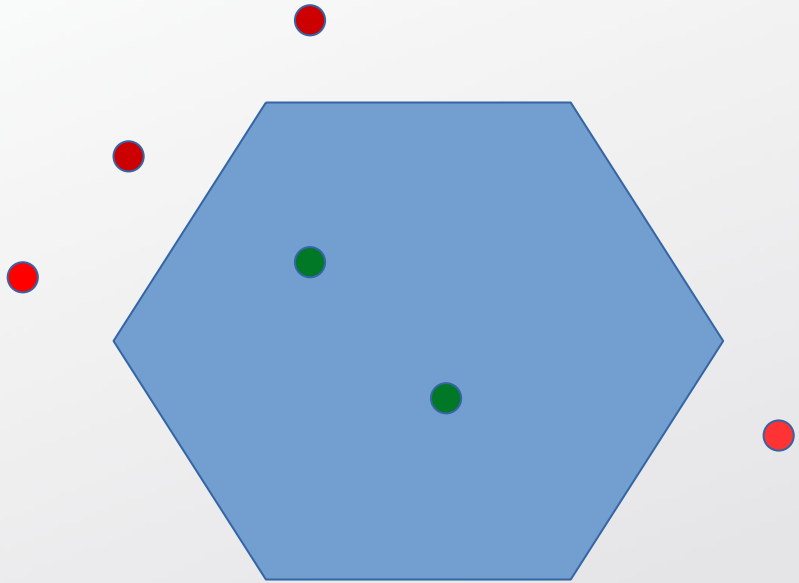


Wszystkie punkty geometrii A muszą znajdować się wewnątrz geometrii B.

SELECT a.nazwa AS nazwa_rezerwatu, b.nazwa AS nazwa_natura FROM rezerwaty a, soo b WHERE ST_Contains(a.geom, b.geom);

ST_Contains sprawdza ten sam warunek, co ST_Within, ale inna jest kolejność argumentów

Wszystkie punkty geometrii B muszą zawierać się wewnątrz geometrii A.

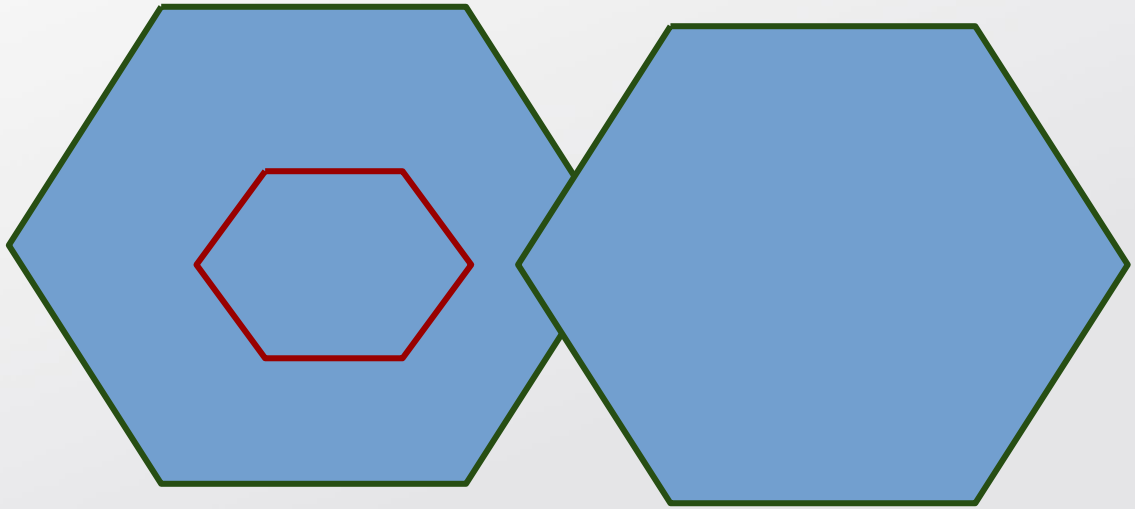


ST_Overlaps

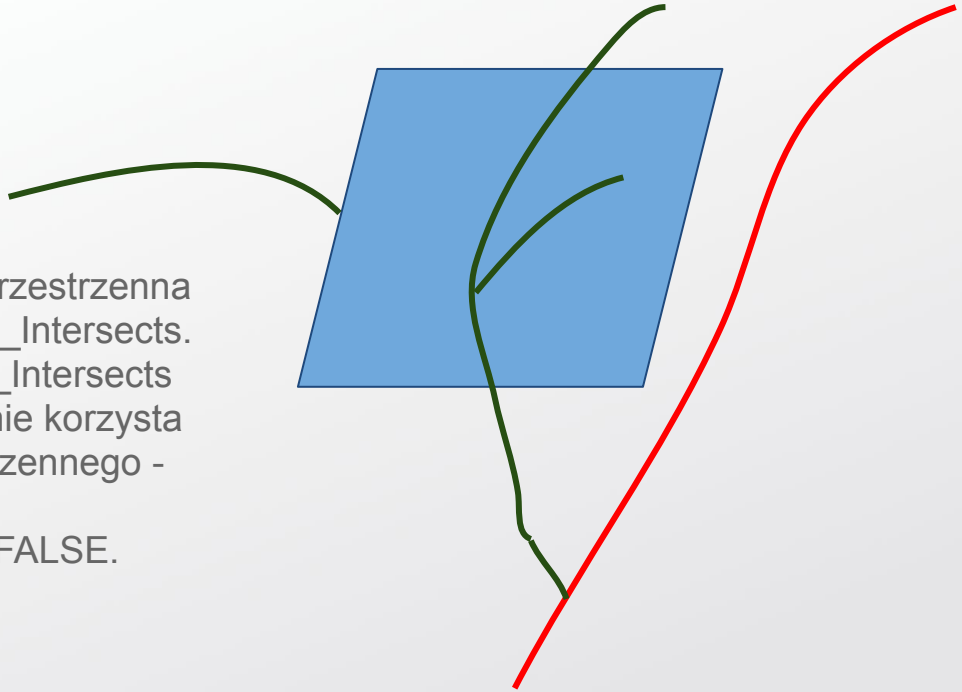
SELECT a.nazwa AS nazwa_oso, b.nazwa AS nazwa_soo FROM oso a, soo b WHERE ST_Overlaps(a.geom, b.geom);

Obiekty nakładają się, ale jeden nie może się znajdować całkowicie wewnątrz drugiego.

Wynik przecięcia musi mieć taki sam wymiar (typ geometrii) jak wymiary wejściowej, np. z przecięcia dwóch geometrii powierzchniowych musi powstać również powierzchnia.



SELECT a.adr_for, b.geom FROM wydzielenia a, drogi b WHERE ST_Intersects(a.geom, b.geom);



Jakakolwiek relacja przestrzenna zwróci TRUE przy ST_Intersects. Przeciwieństwem ST_Intersects jest ST_Disjoint, ale nie korzysta ona z indeksu przestrzennego - lepiej użyć warunku ST_Intersects(a,b) = FALSE.

SELECT a.* FROM drogi a, drogi b WHERE ST_Crosses(a.geom,b.geom) AND b.id = 123;

Geometrie muszą mieć wspólną część (ale nie wszystkie) punktów wewnętrznych.

Wynik przecięcia dwóch obiektów musi mieć mniej wymiarów, niż największy z wymiarów wejściowych, przy czym za wymiar w rozumieniu specyfikacji uważa się typ geometrii (punktowa - 1 wymiar, liniowa - 2 wymiar, powierzchniowa - 3 wymiar).

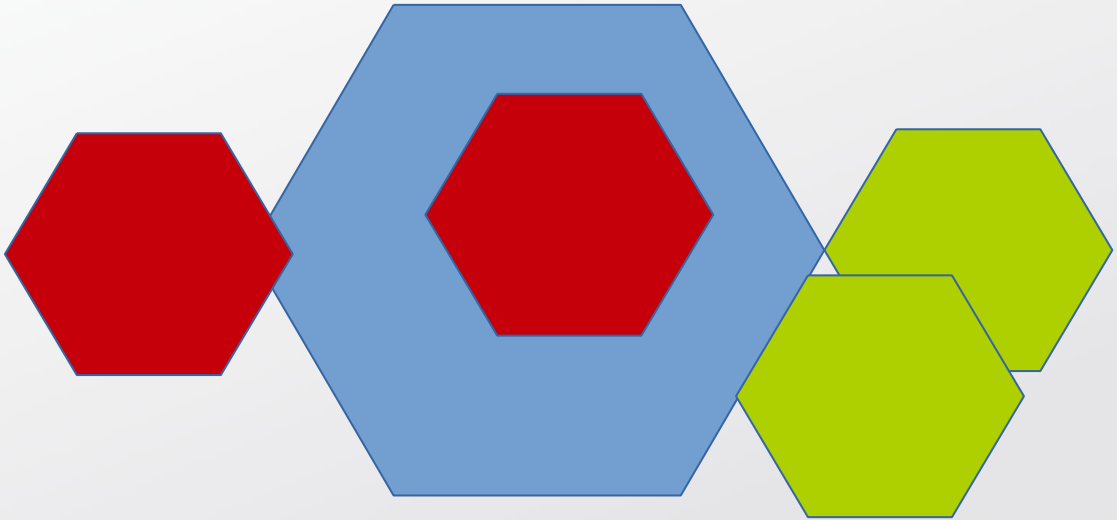
poligon z poligonem - wynik musi być linią

poligon z linią - wynik musi być punktem
linia z linią - wynik musi być punktem

poligon z punktem, linia z punktem, punkt z punktem - nigdy nie będzie spełniony

SELECT * FROM wydzielenia WHERE ST_Touches(geom, (SELECT geom FROM wydzielenia WHERE id=6));

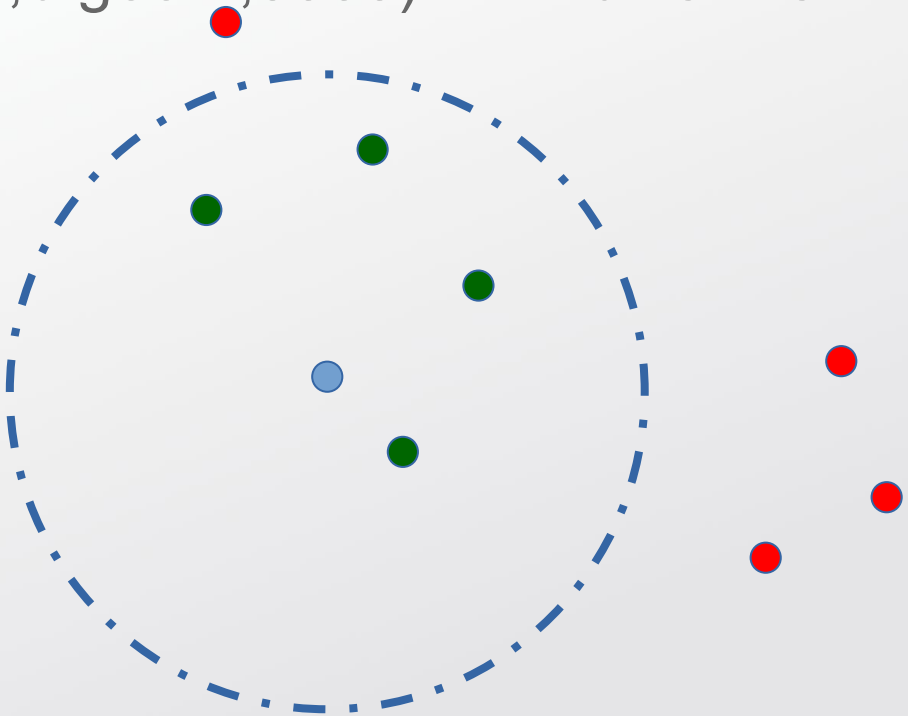
Zapytanie wyszukuje wydzielenia sąsiadujące z wydzieleniem o ID=6. Geometria badanego wydzielenia została wyekstrahowana z użyciem **podzapytania** (***subquery***).
Inne techniki podstawiania wyników zapytania jako parametrów zostaną podane w dalszej części szkolenia.



ST_DWithin

```
SELECT a.naz_glowna, ST_Distance(a.geom, b.geom) , a.geom
FROM miejscowosci a , oso b WHERE
ST_Dwithin(a.geom,b.geom,5000) AND b.nazwa = 'Zbiornik Nyski';
```

Zapytanie wyszukuje miejscowości znajdujące się w promieniu 5000 m od obszaru "Zbiornik Nyski". Funkcja ST_Dwithin buduje w pamięci bufor wokół geometrii A i porównuje z geometrią B. Podobny efekt można uzyskać z wykorzystaniem samej funkcji ST_Distance lub tworząc bufor w sposób jawny funkcją ST_Buffer, jednak ST_Dwithin jest najszybszym sposobem - korzysta bowiem z indeksu przestrzennego. Funkcja nie jest zdefiniowana w normie ISO, jest specyficzna dla PostGIS i MS SQL.



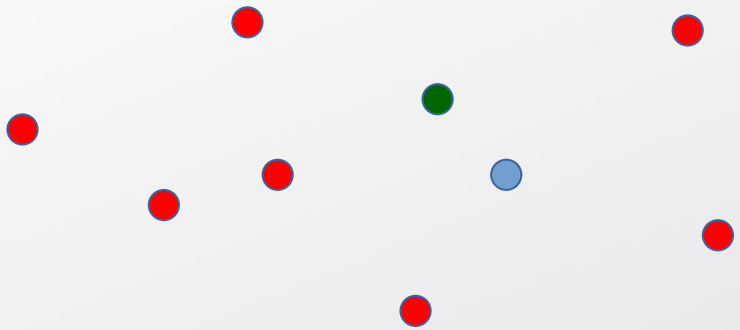
Podanie współrzędnych

```
SELECT * FROM miejscowosci WHERE
ST_DWithin(geom,
    ST_Transform(
        ST_SetSRID(
            ST_MakePoint(17.23997,51.45263),
        4326),
    2180),
200);
                    )
```

ST_MakePoint – utworzenie geometrii punktowej
ST_SetSRID – ustalenie układu współrzędnych
ST_Transform – transformacja układu.

Indexed Nearest Neighbor

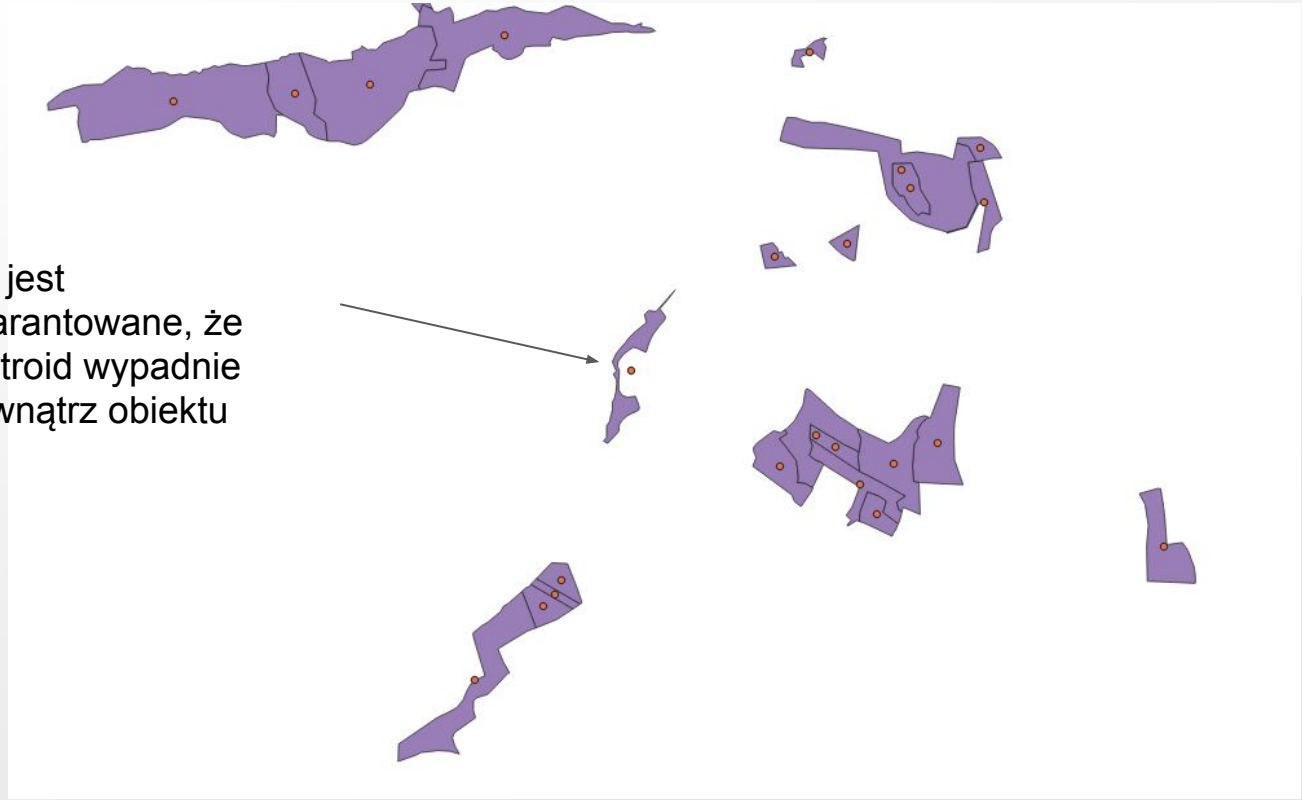
```
SELECT * FROM miejscowosci ORDER BY geom <->
ST_Transform(ST_SetSRID(ST_MakePoint(17.23997,51.45263),4326),
2180) LIMIT 1;
```



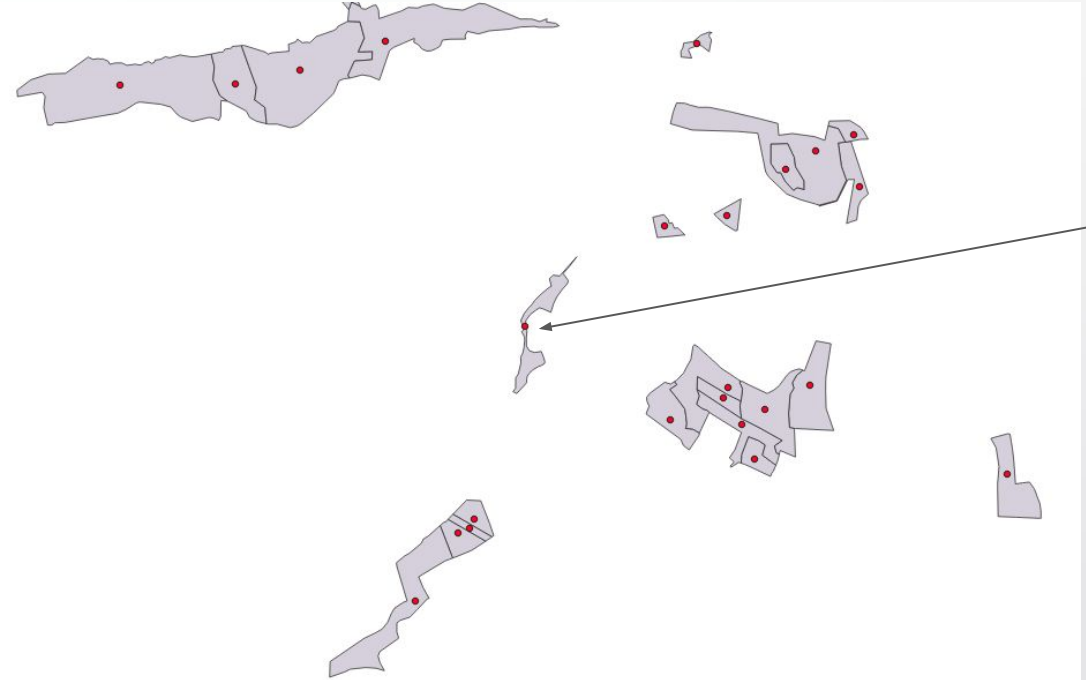
Operator <-> jest uproszczoną wersją ST_Distance (porównuje nie same geometrie, lecz ich zasięgi - BBOX) oraz korzysta z indeksu przestrzennego - działa przez to szybciej od ST_Distance.

Przetwarzanie danych: generalizacja, buforowanie, transformacja układu, scalanie, docinanie, obliczanie powierzchni i obwodu

SELECT adr_for, ST_Centroid(geom) FROM wydzielenia;

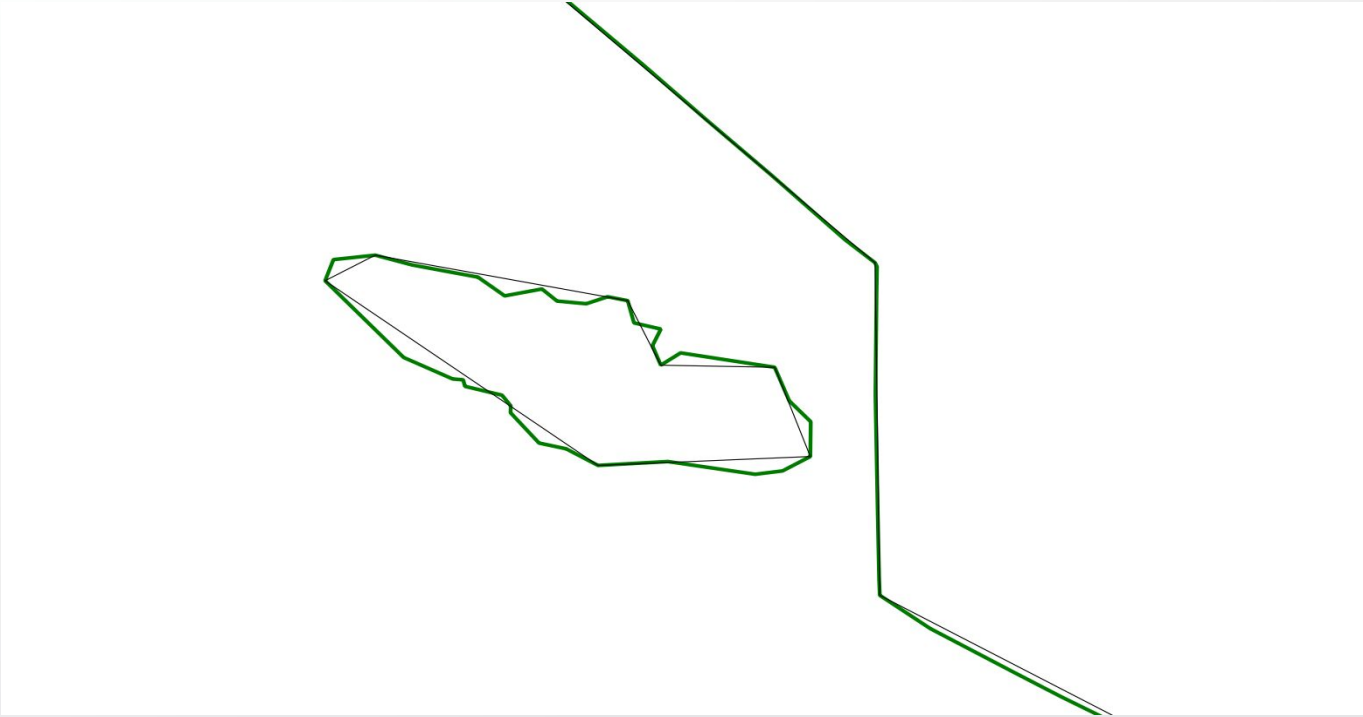


SELECT adr_for, ST_PointOnSurface(geom) FROM wydzielenia;



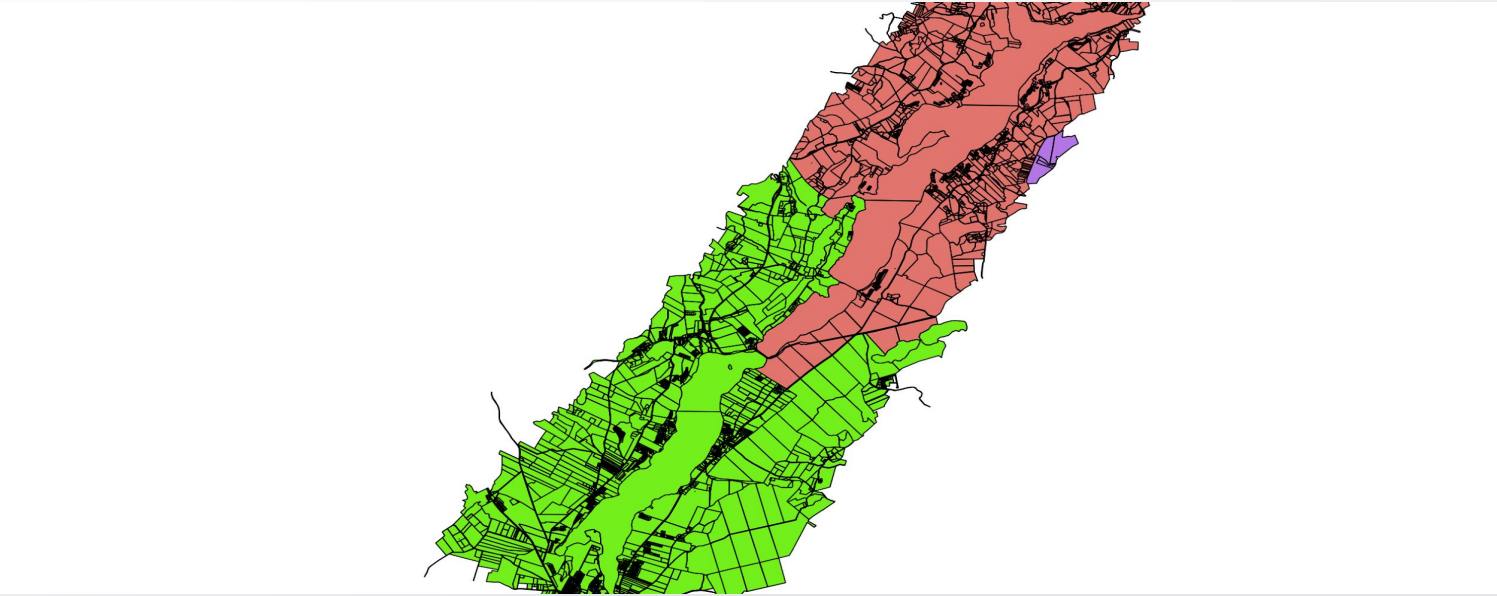
PointOnSurface niekoniecznie leży w środku masy obiektu, ale zawsze zawiera się w jego granicach

SELECT adr_for, ST_Simplify(geom,10) FROM wydzielenia;



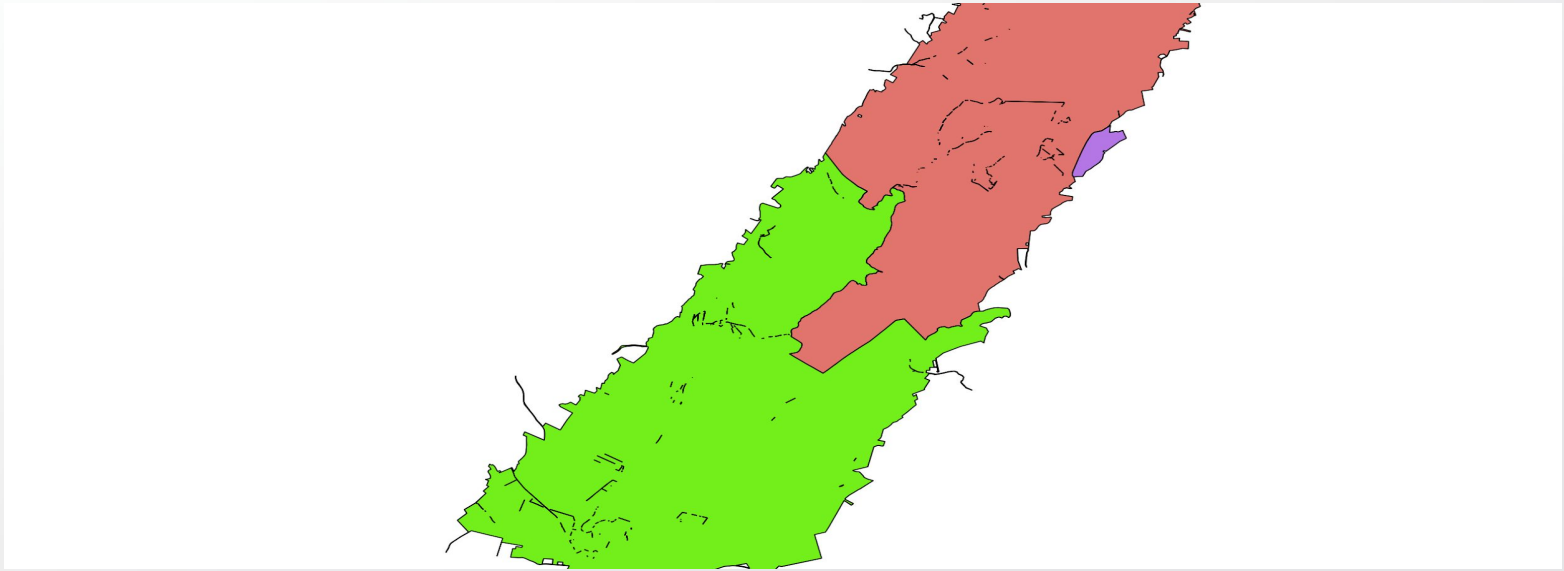
```
SELECT numer, ST_Length(geom)/1000 FROM drogi;
--długość (w km)

SELECT adr_for, ST_Area(geom)/10000 FROM wydzielenia;
--powierzchnia (w ha)

SELECT adr_for, ST_Perimeter(geom)/1000 FROM
wydzielenia; --obwód (w km)
```

```
SELECT split_part(adr_for,' ',1), ST_Collect(geom) AS geom
FROM wydzielenia GROUP BY split_part(adr_for,' ',1);
```



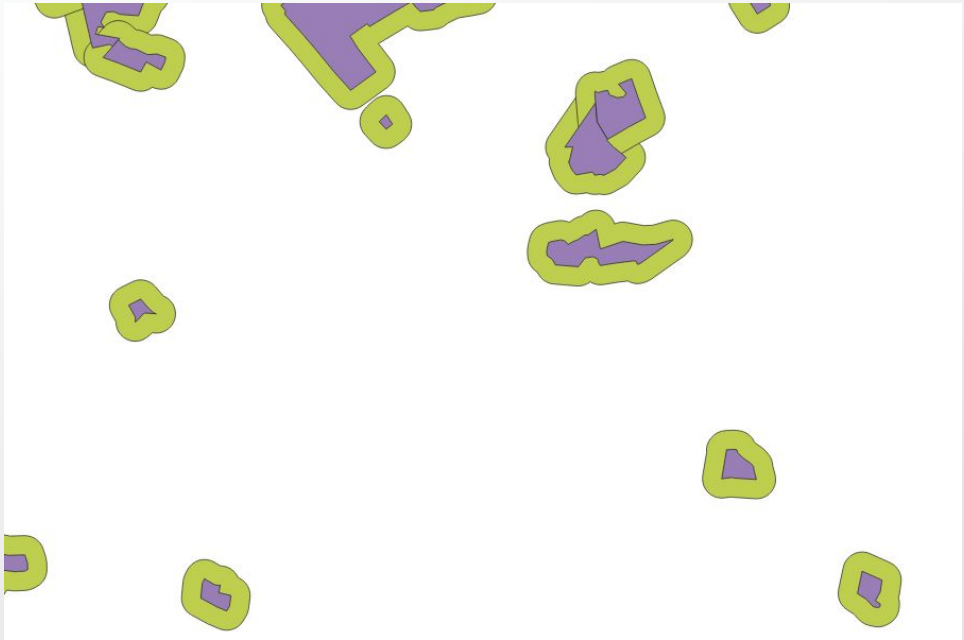
Wynikiem są geometrie typu MULTI

```
SELECT split_part(adr_for,' ',1), ST_Union(geom) AS geom
FROM wydzielenia GROUP BY split_part(adr_for,' ',1);
```



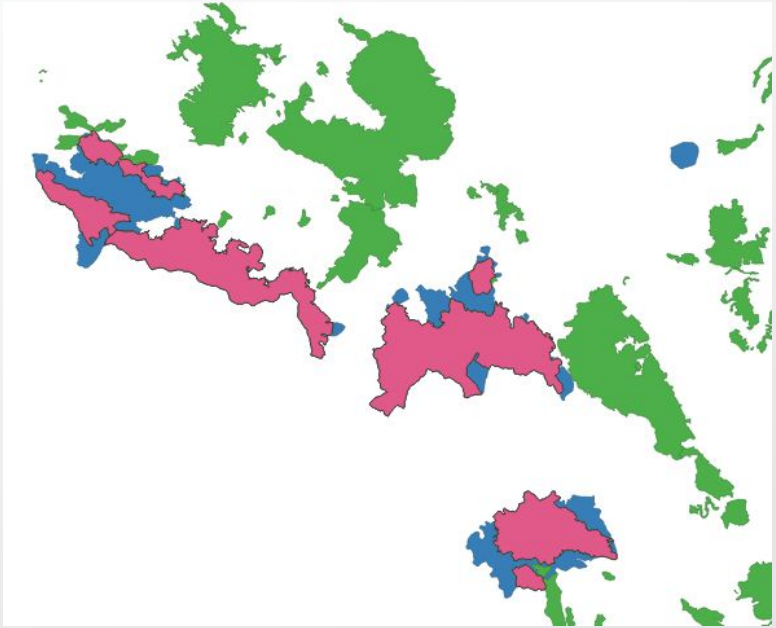
Geometrie zostają scalone (na ile pozwala topologia...)

SELECT adr_for, spec_age, ST_Buffer(geom, 100) FROM wydzielenia WHERE spec_age > 100;



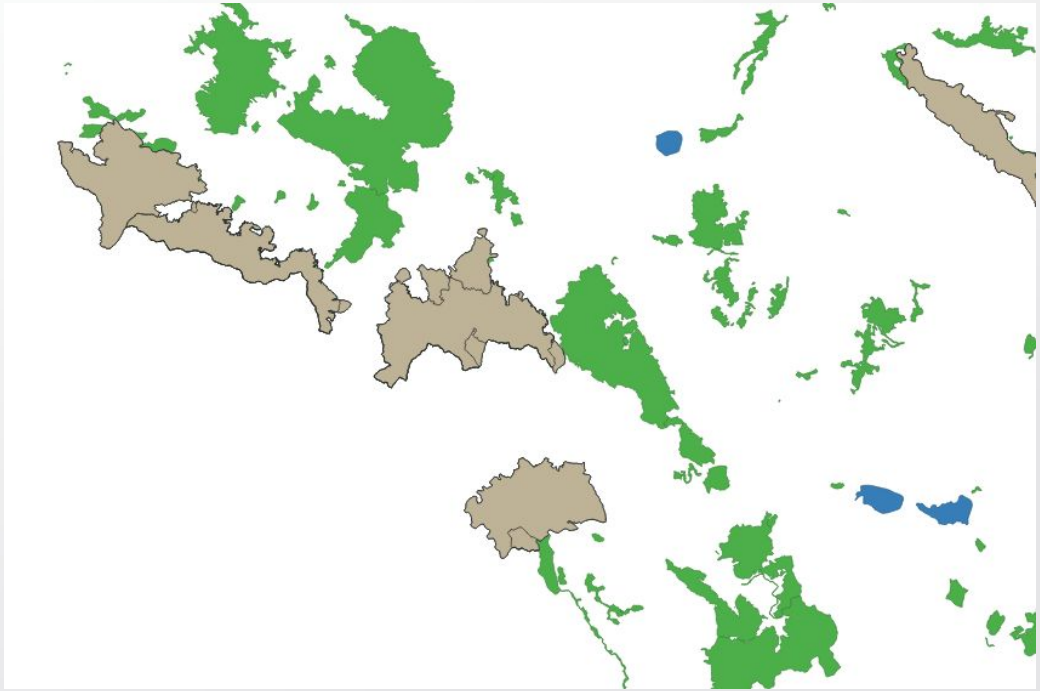
Iloczyn przestrzenny

```
SELECT a.nazwa AS nazwa_oso, b.nazwa AS nazwa_soo,
ST_Intersection(a.geom, b.geom) FROM oso a, soo b WHERE
ST_Overlaps(a.geom, b.geom);
```



Różnica przestrzenna

```
SELECT a.nazwa AS nazwa_oso, b.nazwa AS nazwa_soo,
ST_Difference(a.geom, b.geom) FROM oso a, soo b WHERE
ST_Overlaps(a.geom, b.geom);
```



Zarządzanie uprawnieniami w bazie danych, konta użytkowników i ich uprawnienia, zasady edycji w wiele osób

Uprawnienia

- PostgreSQL posiada rozbudowany system uprawnień
- Najważniejsze koncepcje:
  - Podstawową jednostką jest rola (ROLE)
  - Użytkownik (USER) to ROLE z prawem LOGIN
  - Role bez prawa LOGIN mogą być wykorzystane jako grupy użytkowników
  - Każdy użytkownik jest członkiem grupy "public"
  - Nieograniczone uprawnienia daje prawo SUPERUSER, przy czym domyślnie jest to użytkownik "postgres", ale można utworzyć więcej ról z takim statusem

Zarządzanie użytkownikami

- **CREATE USER** lesniczy **WITH PASSWORD** 'las' **CREATEDB CREATEROLE**;
  - --uprawnienie CREATEDB do tworzenia baz danych
  - --uprawnienie CREATEROLE do tworzenia kolejnych użytkowników
- **CREATE USER** gajowy **WITH PASSWORD** 'gaj';
- **CREATE USER** drwal **WITH PASSWORD** 'siekiera';
- **CREATE ROLE** sluzba_lesna;
- **GRANT** sluzba_lesna **TO** lesniczy;
- **GRANT** sluzba_lesna **TO** gajowy;

Zarządzanie uprawnieniami

- **GRANT SELECT ON** wydzielenia **TO** public;
- **GRANT ALL ON** wydzielenia **TO** sluzba_lesna;
- **GRANT SELECT ON** siedliska **TO** sluzba_lesna;
- **GRANT INSERT, UPDATE, DELETE ON** siedliska **TO** lesniczy;

Edycja wieloosobowa

PostgreSQL umożliwia edycję przez wiele osób (i/lub programów) jednocześnie.

Nie posiada jednak mechanizmów blokowania danych ani rozwiązywania konfliktów. Ostatecznie zapisana zostanie wersja tego użytkownika, który jako ostatni zapisał swoje zmiany.

Narzędzia wspomagające pracę wieloosobową:

- GeoGig (współpraca jak nad kodem źródłowym oprogramowania)
- pgVersion
- własne funkcje trigger (np. Wersjonowanie w QGIS)

Pytania kontrolne

- Czy użytkownik "gajowy" może edytować tabelę "siedliska"?
- Czy użytkownik "gajowy" może edytować tabelę "wydzielenia"?
- Czy użytkownik "drwal" może przeglądać dane tabeli "wydzielenia"?
- Czy użytkownik "lesniczy" może przeglądać dane tabeli "wydzielenia"?

Pojęcie transakcji, zaawansowana transakcyjność

Transakcje

Pod pojęciem transakcji w bazach danych kryje się pewien blok instrukcji, który musi być wykonany w całości lub wcale.

Najczęściej podawanym przykładem jest przelew bankowy:

- saldo rachunku nadawcy musi być pomniejszone o kwotę przelewu
- saldo rachunku odbiorcy musi być powiększone o kwotę przelewu
- nie może dojść do sytuacji, kiedy saldo rachunku nadawcy zostanie pomniejszone, ale kwota przelewu nie powiększy salda rachunku odbiorcy.

Zasady ACID

Atomicity - transakcja musi wykonać się w całości, albo w ogóle

Consistency - transakcja nie może naruszać powiązań między danymi

Isolation - transakcja nie ma dostępu do niezatwierdzonych informacji w innych, przebiegających równolegle, transakcjach

Durability - wyniki zatwierdzonych transakcji muszą być zapisane na stałe.

Rozpoczęcie transakcji odbywa się przez komendę **BEGIN;** lub **BEGIN TRANSACTION;**

Zatwierdzenie zmian następuje poprzez komendę **COMMIT;**

Anulowanie zmian jest wykonywane komendą **ROLLBACK;**

Komendy wpisuje się w oknie SQL lub w przypadku skryptów - w treści skryptu.

Transakcje w praktyce

- Jeśli nie korzysta się z transakcji, to każde zapytanie jest traktowana jako jednoelementowa transakcja.
- Zatwierdzenie transakcji jest dość "kosztowną" operacją (w zakresie operacji dyskowych), dlatego seria zapytań zmieniających dane wykonywanych bez transakcji wykona się istotnie wolniej, niż z użyciem jednej większej transakcji.
- Jeśli wystąpi błąd: `current transaction is aborted, commands ignored until end of transaction block` - by przywrócić możliwość wykonywania kolejnych zapytań należy wykonać ROLLBACK;

Wewnątrz transakcji można stosować punkty zapisu (SAVEPOINT), które pozwalają powrócić do zapisanego stanu transakcji w razie wystąpienia błędu, bez konieczności anulowania całej transakcji.

Przykład:

```
BEGIN;
CREATE TABLE tabela_testowa (id serial, val float);
INSERT INTO tabela_testowa(val) VALUES(0.1);
INSERT INTO tabela_testowa(val) VALUES(1/5);
INSERT INTO tabela_testowa(val) VALUES(1/0); -- błąd!
INSERT INTO tabela_testowa(val) VALUES(1/2);
```

```
ERROR:  current transaction is aborted, commands ignored until end of transaction block
SQL state: 25P02
```

Przykład:

```
BEGIN;
CREATE TABLE tabela_testowa (id serial, val float);
INSERT INTO tabela_testowa(val) VALUES(0.1);
INSERT INTO tabela_testowa(val) VALUES(1/5);
SAVEPOINT nie_dziel_przez_zero;
INSERT INTO tabela_testowa(val) VALUES(1/0); -- błąd!
ROLLBACK TO SAVEPOINT nie_dziel_przez_zero;
INSERT INTO tabela_testowa(val) VALUES(1/2);
COMMIT;
```

Jeśli punkt zapisu przestanie być potrzebny jeszcze przed zatwierdzeniem całej transakcji, można uwolnić zasoby przez niego zajmowane komendą:

```
RELEASE SAVEPOINT <nazwa>;
```

Manipulowanie dużymi zbiorami danych

Pod pojęciem "duże zbiory danych" rozumiemy dane o liczności powyżej 1 miliona wierszy.

PostgreSQL może być używany do obsługi dużych zbiorów:

- projekt OpenStreetMap - 45 GB danych
- Agencja Restrukturyzacji i Modernizacji Rolnictwa - 38 milionów działek, 57 milionów *pól zagospodarowania*

Narzędzia przydatne w pracy z dużymi zbiorami danych:

- typ BIGSERIAL - jeśli jest możliwe, że identyfikatory będą przekraczały wartość 2 mld
- indeks BRIN - indeks zużywający mało miejsca na dysku, wymaga "naturalnego posortowania" danych (np. dziennik zdarzeń i filtrowanie po dacie)
- TABLESAMPLE - szybszy od LIMIT sposób pobrania próbki danych z dużych tabel, np. ok. 1% tabeli "miejscowosci":
- SELECT * FROM miejscowosci TABLESAMPLE system(1);

Narzędzia przydatne w pracy z dużymi zbiorami danych:

- szacowana liczba wierszy - przy dużych zbiorach count(*) jest zapytaniem bardzo wolnym, szacowaną liczbę wierszy można sprawdzić:
- SELECT reltuples FROM pg_class WHERE relname ='miejscowosci';
- W QGIS należy zaznaczyć opcję "użyj szacunkowych metadanych tabeli" przy połączeniu z bazą

Partycjonowanie jest sposobem zarządzania dużymi zbiorami danych - podziałem jednej dużej tabeli na mniejsze, rozdzielone według logicznego klucza.

Zalety partycjonowania:

- zmniejsza rozmiary potrzebnych indeksów, przyspiesza niektóre zapytania
- masowe usuwanie danych według klucza partycjonowania jest szybsze (DROP TABLE zamiast DELETE)
- ułatwia archiwizację danych - w przypadku partycjonowania według daty.
- **pełne wsparcie od wersji 11**

Typy partycjonowania:

- RANGE: partycja przyjmuje dane o określonym zakresie wartości (liczbowych lub daty)
- LIST: partycja przyjmuje dane o określonych wartościach (np. tekstowych)
- HASH: definiowana jest liczba partycji, dane są rodzielane po równo.

```
CREATE TABLE pomniki_part (LIKE pomniki_przyrody) PARTITION BY range(data_utwor);

CREATE TABLE pomniki_p89 PARTITION OF pomniki_part FOR VALUES FROM ('1952-10-21'::date) TO ('1989-12-31'::date);
CREATE TABLE pomniki_p95 PARTITION OF pomniki_part FOR VALUES FROM ('1990-01-01'::date) TO ('1995-12-31'::date);
CREATE TABLE pomniki_p00 PARTITION OF pomniki_part FOR VALUES FROM ('1996-01-01'::date) TO ('2000-12-31'::date);
CREATE TABLE pomniki_p05 PARTITION OF pomniki_part FOR VALUES FROM ('2001-01-01'::date) TO ('2005-12-31'::date);
CREATE TABLE pomniki_p20 PARTITION OF pomniki_part FOR VALUES FROM ('2006-01-01'::date) TO ('2020-12-31'::date);
INSERT INTO pomniki_part SELECT * FROM pomniki_przyrody;
```

Trwałe usunięcie partycji:

DROP TABLE pomniki_p89;

"Odpięcie" partycji:

ALTER TABLE pomniki_part DETACH PARTITION popc_p95;

Przed dużym zasileniem bazy, dla poprawy jego wydajności:

- ustawić parametr konfiguracyjny "synchronous_commit" na "off":

poprzez zmianę wpisu w pliku postgresql.conf lub przez wykonanie komendy SQL SET synchronous_commit = 'off';

- jeśli zasilana jest istniejąca tabela, usunąć z niej indeksy
- upewnić się, że używane narzędzie do zasileń używa polecenia COPY a nie INSERT (np. QGIS nie używa, ogr2ogr i shp2pgsql mogą używać. Dla ogr2ogr użycie COPY determinuje parametr PG_USE_COPY, a dla shp2pgsql - parametr -D.)

Zarządzanie obiektami bazy danych

Komendy zarządzania obiektami są określane jako **Data Definition Language (DDL)**.

Porównanie komend DML i DDL:

Operacja	**Dane**	**Obiekty**
Odczyt	SELECT	brak (w innych systemach: DESCRIBE)
Modyfikacja	UPDATE	ALTER
Utworzenie	INSERT	CREATE
Usunięcie	DELETE	DROP

Przykłady komend DDL w PostgreSQL:

```
CREATE TABLE test(id serial primary key, atrybut varchar);
-- tworzy nową, pustą tabelę

CREATE TABLE bory AS SELECT adr_for, species_cd, site_type FROM wydzielenia WHERE site_type LIKE 'B%';
-- tworzy tabelę z wyniku zapytania

ALTER TABLE bory ADD COLUMN zbiorowisko VARCHAR;
-- dodaje kolumnę do tabeli

ALTER TABLE bory ALTER COLUMN zbiorowisko SET DATA TYPE text;
-- zmienia typ danych

ALTER SEQUENCE test_id_seq RESTART WITH 9999;
-- zmienia sekwencję identyfikatorów (np. w razie wystąpienia konfliktu)
```

```
ALTER TABLE bory RENAME COLUMN zbiorowisko TO syntakson;
-- zmienia nazwę kolumny
-- nie istnieje komenda do zmiany kolejności kolumn, należy utworzyć tabelę od nowa

CREATE SCHEMA testowe_dane;
-- tworzy nowy, pusty schemat

ALTER TABLE bory SET SCHEMA testowe_dane;
-- przenosi tabelę "bory" do schematu "testowe_dane"

DROP SCHEMA testowe_dane CASCADE;
-- usuwa schemat i wszystkie tabele w nim zawarte
-- DROP TABLE … CASCADE usuwa tabelę i wszystkie widoki od niej zależne
```

PostgreSQL udostępnia modyfikatory IF EXISTS oraz IF NOT EXISTS, które umożliwiają uzależnienie wykonania komend DDL w zależności od istnienia (lub nie) obiektów których mają dotyczyć. Pozwala to na uniknięcie błędów krytycznych i przerwania transakcji.

Przykłady:

```
DROP TABLE IF EXISTS zmyslona_tabela;
```

```
ALTER TABLE siedliska ADD COLUMN IF NOT EXISTS site_type_cd VARCHAR;
```

Zaawansowane techniki pozyskiwania danych z użyciem podzapytań oraz funkcji analitycznych

Generowanie raportów z użyciem zaawansowanych funkcji grupujących

Przykład raportu z zastosowaniem funkcji grupujących: **powierzchnia obszarów Natura 2000 w podziale na województwa**

Należy obliczyć powierzchnię obszarów Natura 2000 w każdym województwie, odnieść do powierzchni ogólnej województwa i posortować listę malejąco według procentowego udziału.

Dane:

- dane o obszarach Natura 2000 zgromadzone w tabelach "soo" i "oso"
- dane o województwach w tabeli "wojewodztwa"
- obszary Natura 2000 mogą przecinać granice województw
- powierzchnia województw podana w kolumnie shape_area w "stopniach kwadratowych" - bezużyteczna

Etap 1: połączenie obszarów "siedliskowych" i "ptasich"

SELECT kod, geom FROM oso

UNION ALL

SELECT kod, geom FROM soo

	kod	geom
141	PLB100002	010600002...
142	PLB280004	010600002...
143	PLB300017	010600002...
144	PLC120002	010600002...
145	PLB320018	010600002...
146	PLH060075	010600002...
147	PLH120017	010600002...

Dlaczego UNION ALL zamiast UNION, chociaż wynik nie będzie zawierał powtarzających się wierszy? Z powodów wydajnościowych: UNION ALL nie wykonuje dodatkowej operacji sortowania.
Dla chętnych: wykonane EXPLAIN ANALYZE dla wariantu z UNION i UNION ALL.

Etap 2: wykonanie przecięcia geometrycznego

```
WITH natura AS (SELECT kod, geom FROM oso

UNION ALL

SELECT kod, geom FROM soo)

SELECT a.jpt_nazwa_, sum(
        ST_Area(ST_Intersection(a.geom,b.geom)))
    AS powierzchnia_n2000 FROM wojewodztwa a, natura b
WHERE ST_Intersects(a.geom, b.geom)
GROUP BY a.jpt_nazwa_;
```

	jpt_nazwa_	powierzchnia_n2000
1	dolnośląskie	6475306454.95144
2	kujawsko-pomorskie	2459435024.66916
3	lubelskie	5005658605.16835
4	lubuskie	5033908427.53849
5	mazowieckie	6215650167.72788

Etap 3:
optymalizacja wydajności, obliczenie stosunku powierzchni

```
SELECT jpt_nazwa_ AS wojewodztwo, powierzchnia_n2000/1000000 AS
powierzchnia_n2000 , (powierzchnia_n2000 /  powierzchnia_woj)*100 AS procent_n2000
FROM (
 WITH natura AS (SELECT kod, geom FROM oso
UNION ALL
SELECT kod, geom FROM soo) SELECT a.jpt_nazwa_, sum(
        CASE WHEN ST_Overlaps(a.geom,b.geom) THEN
ST_Area(ST_Intersection(a.geom,b.geom))
    WHEN ST_Contains(a.geom, b.geom) THEN ST_Area(b.geom)
    ELSE 0 END
        )
    AS powierzchnia_n2000, ST_Area(a.geom) AS powierzchnia_woj FROM wojewodztwa a,
natura b
WHERE ST_Intersects(a.geom, b.geom)
GROUP BY a.id) raport
ORDER BY procent_n2000 DESC;
```

obliczenie iloczynu przestrzennego tylko, gdy to konieczne

podzapytanie jest konieczne, gdyż nie można odwołać się do aliasu w tym samym zapytaniu

Wynik końcowy (pierwsze 10 wierszy):

	wojewodztwo	powierzchnia_n2000	procent_n2000
1	podlaskie	11230.7339503069	55.6165104987485
2	zachodniopomorskie	11187.1071709377	48.8615402453189
3	podkarpackie	8611.82518639876	48.2625389814038
4	lubuskie	5033.90842753849	35.9826620836396
5	warmińsko-mazurskie	8341.36920367375	34.539004106911
6	dolnośląskie	6475.30645495144	32.4801887283024
7	pomorskie	5418.33798686243	29.6007588578261
8	wielkopolskie	6507.68165490778	21.8398415827719
9	lubelskie	5005.65860516835	19.9158830479143
10	małopolskie	2864.11950766589	18.8850868769806

Tabele przestawne

PostgreSQL umożliwia stworzenie tabeli przestawnej z pomocą rozszerzenia tablefunc.
Nie jest ono dostępne standardowo w bazie - wymagane jest wykonanie przez superużytkownika: CREATE EXTENSION tablefunc; w każdej bazie, w której ma być używane.

Przykład: łączna powierzchnia drzewostanów w podziale na typy siedliskowe lasu i gatunki główne

Etap 1. Należy przygotować zapytanie, które zwróci dokładnie 3 kolumny:

1. identyfikatory wierszy
2. identyfikatory kolumn
3. wartości

Etap 1: przygotowanie kategorii i wartości

```
SELECT species_cd, site_type, sum(sub_area)::float
FROM wydzielenia
WHERE species_cd IS NOT NULL AND site_type IS NOT NULL
GROUP BY species_cd, site_type
ORDER BY 1,2;
```

	species_cd	site_type	sum
1	AK	LMŚW	1.28
2	AK	LŚW	0.87
3	BEZ.C	LWYŻW	0.36
4	BEZ.C	LWYŻŚW	0.58
5	BEZ.C	LŁ	3.98
6	BK	BMŚW	0.63

Etap 2: przygotowanie definicji kolumn tabeli przestawnej

Trzeba je wpisać w zapytaniu, można je wygenerować zapytaniem:

```
SELECT 'gatunek VARCHAR,' || string_agg(DISTINCT '"' || site_type || '"
FLOAT', ',') FROM wydzielenia;
```

	?column?
1	gatunek VARCHAR,"BMGŚW" FLOAT,"BMŚW" FLOAT,"LGW" FLOAT,"LGŚW" FLOAT,"LMGŚW" FLOAT,"LMW" FLOAT,"LMWYŻW" FLOAT,"...

Typ dla pierwszej kolumny musi odpowiadać typowi dla kategorii osi Y
Typy dla wartości muszą odpowiadać typowi dla wyniku Etapu 1 (najlepiej dla pewności zastosować rzutowanie - stąd ::float w poprzednim zapytaniu)

Etap 3: przygotowanie właściwego zapytania crosstab

```
SELECT * FROM crosstab ('SELECT species_cd, site_type,
sum(sub_area)::float
FROM wydzielenia
WHERE species_cd IS NOT NULL AND site_type IS NOT NULL
GROUP BY species_cd, site_type
ORDER BY 1,2') AS final_result (
gatunek VARCHAR,"BMGŚW" FLOAT,"BMŚW" FLOAT,"LGW"
FLOAT,"LGŚW" FLOAT,"LMGŚW" FLOAT,"LMW" FLOAT,"LMWYŻW"
FLOAT,"LMWYŻŚW" FLOAT,"LMŚW" FLOAT,"LW" FLOAT,"LWYŻW"
FLOAT,"LWYŻŚW" FLOAT,"LŁ" FLOAT,"LŁG" FLOAT,"LŚW"
FLOAT,"LŁWYŻ" FLOAT,"OL" FLOAT,"OLJ" FLOAT
);
```

Tabele przestawne

	gatunek character varying	BMGŚW double precision	BMŚW double precision	LGW double precision	LGŚW double precision	LMGŚW double precisi
12	JD	4.36	2.65	16.43	[null]	
13	JS	7.24	2.39	4.07	4.83	
14	JW	3	24.01	31.82	16.34	
15	KL	2.33	[null]	[null]	[null]	
16	LP	12.83	11.69	4.69	12.84	
17	LSZ	0.14	0.76	1	[null]	
18	MD	4.7	7.1	284.2	34.59	
19	OL	7.49	7.87	26.01	13.65	
20	OL.S	2.77	4.25	1.24	[null]	
21	OS	0.6	2.54	10.43	1.84	
22	SO	314.8	236.6	39.79	22.97	
23	SO.WE	3.07	[null]	[null]	[null]	
24	TP	2.73	23.8	6.2	0.5	
25	WB	2.66	0.56	11.06	1.69	
26	WZ	5.57	0.45	[null]	[null]	
27	ŚL.A	5.52	2.29	[null]	[null]	
28	ŚL.T	0.82	0.05	0.58	[null]	
29	ŚW	27.78	1.99	69.93	92.02	

Mianem **window function** określa się funkcje operujące na posortowanych grupach wierszy, ale w odniesieniu do pojedynczych wartości - bez agregacji.

Przykład: tabela wyników wyścigu

Oczekiwany wynik:

- miejsce w wyścigu
- strata do poprzedniego zawodnika
- strata do najlepszego zawodnika

	id [PK] integer	zawodnik character varying	czas real
1	1	Jean-Éric Vergne	46.0555
2	2	Oliver Rowland	46.0755
3	3	Felipe Massa	47.3355
4	4	Pascal Wehrlein	48.8855
5	5	Sébastien Buemi	55.3
6	6	Mitch Evans	71.5
7	7	André Lotterer	88.3
8	8	Alex Lynn	106.1
9	9	Stoffel Vandoorne	124.6
10	10	José María López	143.46

Rozwiązanie:

```
SELECT zawodnik, rank() over(order by czas) as miejsce,
czas - lag(czas) over (order by czas) as strata,
lead(czas) - czas over (order by czas) as zysk,
czas - first_value(czas) over (order by czas) as do_najlepszego
FROM formula_e;
```

Funkcja rank określa rangę danej wartości w określonym zbiorze,
funkcja lag odwołuje się do poprzedniego wiersza w zbiorze,
funkcja lead odwołuje się do następnego wiersza w zbiorze,
funkcja first_value odwołuje się do pierwszego wiersza w zbiorze

Wyrażenia regularne

Wyrażenia regularne

PostgreSQL posiada wsparcie dla wyrażeń regularnych w standardzie POSIX.
Do stosowania w klauzuli WHERE przewidziano następujące operatory:

~ - wartość pasuje do wyrażenia z uwzględnieniem wielkości liter

```
SELECT * FROM rezerwaty WHERE nazwa ~ '.prof(\.|esora).';
```

~* - wartość pasuje do wyrażenia bez uwzględnienia wielkości liter

```
SELECT * FROM rezerwaty WHERE nazwa ~* '.prof(\.|esora).';
```

!~ - wartość nie pasuje do wyrażenia z uwzględnieniem wielkości liter

```
SELECT * FROM rezerwaty WHERE nazwa ~ '.im(\.|ienia).' AND nazwa !~ '.prof(\.|esora).';
```

!~* - wartość nie pasuje do wyrażenia bez uwzględnienia wielkości liter

```
SELECT * FROM rezerwaty WHERE nazwa ~ '.im(\.|ienia).' AND nazwa !~* '.prof(\.|esora).';
```

Wyrażenia regularne

Dostępne są również funkcje wykorzystujące wyrażenia regularne:

Funkcja **regexp_replace** - zamienia znaki pasujące do wyrażenia na zadane, np. ekstrakcja cyfr z tekstu za pomocą zamiany wszystkich znaków nie-numerycznych na puste:

```
SELECT kod, regexp_replace(kod, '[[:alpha:]]','','g') FROM oso;
```

Funkcja **regexp_matches** - znajduje dopasowania do wyrażenia wewnątrz tekstu:

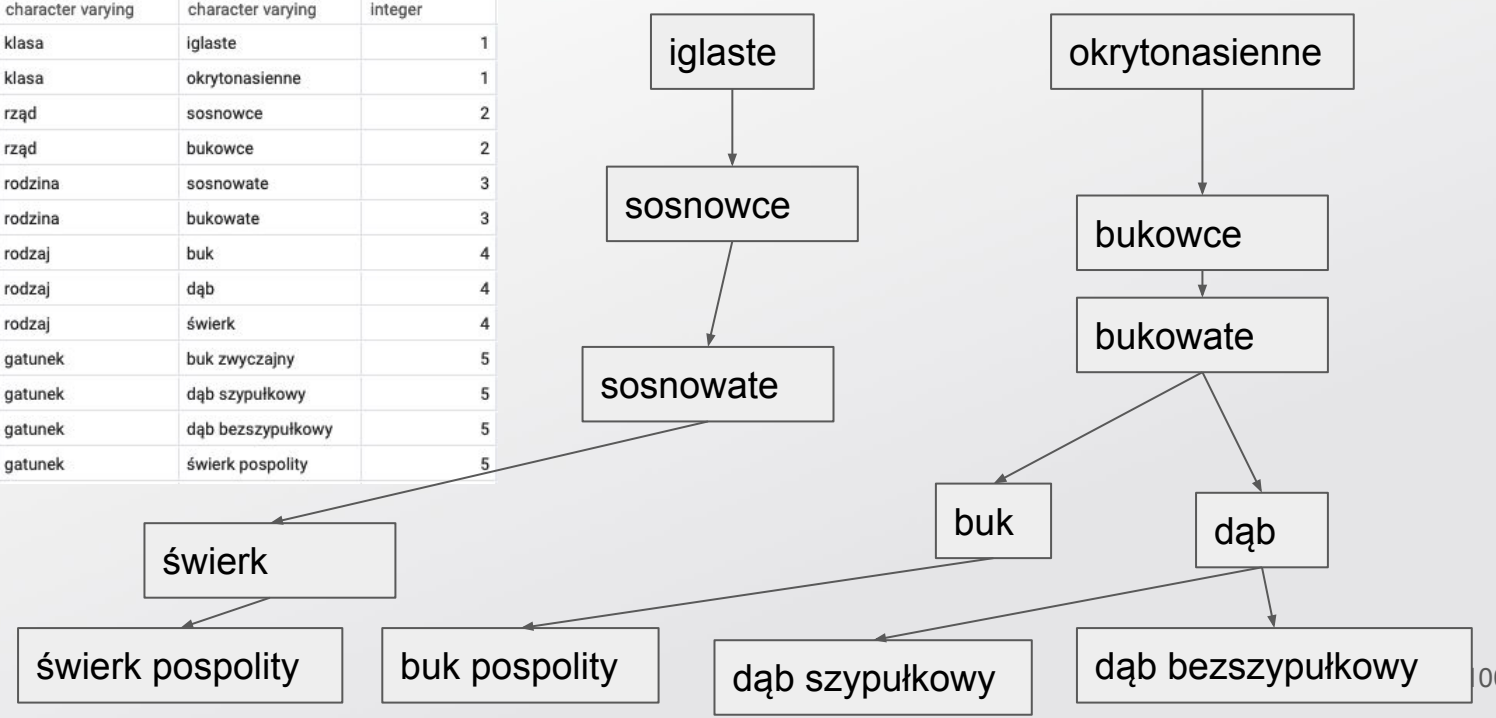
```
SELECT kodinspire, regexp_matches(kodinspire, '(\d+)', 'g') FROM oso;
```

Klauzula WITH, zapytania hierarchiczne i "chodzenie po drzewie"

Struktura drzewa

Do modelowania struktury drzewa w SQL używa się zwykle tabeli posiadającej odwołanie do wierszy w tej samej tabeli, określające poziom nadrzędny.

	id [PK] integer	id_nadrzedny integer	poziom character varying	nazwa character varying	nr_poziomu integer
1	1	[null]	klasa	iglaste	1
2	2	[null]	klasa	okrytonasienne	1
3	3	1	rząd	sosnowce	2
4	4	2	rząd	bukowce	2
5	5	3	rodzina	sosnowate	3
6	6	4	rodzina	bukowate	3
7	7	6	rodzaj	buk	4
8	8	6	rodzaj	dąb	4
9	9	5	rodzaj	świerk	4
10	10	7	gatunek	buk zwyczajny	5
11	11	8	gatunek	dąb szypułkowy	5
12	12	8	gatunek	dąb bezszypułkowy	5
13	13	9	gatunek	świerk pospolity	5



Klauzula WITH

PostgreSQL umożliwia stosowanie tzw. **Common Table Expressions** (CTE) czyli tymczasowych zbiorów danych, które mogą być użyte w zapytaniach.

Do zdefiniowania CTE służy klauzula WITH.

Przykład: obliczenie różnicy przestrzennej wydzieleń i bufora 10 m od dróg.

```
WITH bufor AS (SELECT id, ST_Buffer(geom,10) AS geom FROM drogi)
SELECT w.adr_for, ST_Difference(w.geom, b.geom) FROM wydzielenia w
LEFT JOIN bufor b ON ST_Overlaps(w.geom, b.geom);
```

Instrukcja warunkowa CASE

Klauzula CASE

Instrukcja warunkowa CASE służy do przypisania wartości według podanych reguł

Ma postać:

CASE WHEN *warunek1* THEN *wartość1*
WHEN *warunek2* THEN *wartość2*
ELSE *wartość domyślna* END

Przykład:

```
SELECT adr_for, site_type, forest_fun,
CASE WHEN species_cd = 'DB' THEN 'dąbrowa'
WHEN species_cd = 'BK' THEN 'buczyna'
WHEN species_cd IN ('SO', 'MD', 'ŚW') THEN 'bór'
WHEN site_type IS NULL THEN NULL
ELSE 'las mieszany' END AS opis_lasu FROM wydzielenia;
```

Klauzula CASE

W klauzuli CASE obowiązkowe jest podanie co najmniej jednego warunku (WHEN … THEN) oraz słowa kluczowego END na końcu.

Podanie wartości domyślnej (ELSE) oraz aliasu dla wyniku (AS) jest nieobowiązkowe.

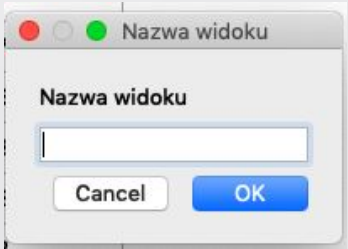
Widoki i ich zastosowanie

Widok (VIEW) jest zapisanym w bazie zapytaniem, którego wynik może być używany jako tabela.
Każdorazowe użycie widoku powoduje ponowne wykonanie zapytania.

CREATE VIEW miasta **AS SELECT** * **FROM** miejscowosci **WHERE** rodzaj_obi = 'miasto';

SELECT * **FROM** miasta;

QGIS dysponuje przyciskiem Utwórz widok który pozwala na utworzenie widoku na podstawie już wpisanego w oknie SQL zapytania, po podaniu nazwy dla widoku:



Zmaterializowany widok

Zmaterializowany widok (MATERIALIZED VIEW) jest rodzajem tabeli, dla której przechowywane jest zapytanie, na podstawie którego została stworzona.
Szczególnie dobrym zastosowaniem dla widoków zmaterializowanych są wyniki analiz przestrzennych, które wymagają dużo czasu i mocy obliczeniowej do przeliczenia.

Dane zmaterializowanego widoku są pobierane z dysku, nie są odświeżane za każdą zmianą danych źródłowych.

CREATE MATERIALIZED VIEW oso_gen50 **AS SELECT** nazwa, kod, ST_SimplifyPreserveTopology(geom,50) **FROM** oso;

Odświeżenie:

REFRESH MATERIALIZED VIEW oso;

Dziękuję za uwagę