|  |
| --- |
| Aplikacje i usługi gis |
| Porównanie indeksów przestrzennych |
| Dokumentacja |
|  |
| **Karol Dzitkowski** |
| **2014-02-06** |

|  |
| --- |
| Dokument zawiera dokumentację z porównania różnego rodzaju indeksów z indeksem przestrzennym w QGIS. Opisana została aplikacja stworzona w celu wykonania testów, wraz ze sposobem jej instalacji, a także wyniki przeprowadzonych testów. |

# Opis projektu

Program testowy napisany jest głównie w technologii ASP.NET MVC 4. Do wyświetlania wykresów wykorzystano bibliotekę DotNet.Highcharts. Dane składowane są w bazie danych PostgreSQL z rozszerzeniem PostGIS. Aplikacja łączy się z bazą bezpośrednio za pomocą biblioteki Npgsql w wersji 2.1.3. Do testowania indeksu używanego przez QGIS wykorzystywane jest API w języku Python wystawiane przez aplikację QGIS w wersji 2.2.0. Program uruchamia odpowiednie skrypty Pythonowe łączące się z bazą PostgreSQL i korzystające z API udostępnionego przez QGIS w celu utworzenia indeksu i wykonania odpowiedniego zapytania. Program śledzi czas wykonania zapytań i zwraca wyniki w formie wykresów. Ma również możliwość generowania raportu z testów.

# Konfiguracja

Program napisano w środowisku systemu operacyjnego Microsoft Windows 8.1 w wersji Professional. Do poprawnego działania systemu niezbędne jest zainstalowanie programów:

* PostgreSQL w wersji 9.3.4 – (postgresql-9.3.4-3-windows-x64)
* QGIS w wersji 2.2.0 – (QGIS-OSGeo4W-2.2.0-1-Setup-x86\_64)
* Python w wersji 2.7.6 – (python-2.7.6.msi)
* Konieczne może być także zainstalowanie pakietu scipy-0.14 oraz numpy – (scipy-0.14.0-win32-superpack-python2.7)

Programy należy zainstalować dokładnie w tej kolejności. Inna konfiguracja nie była testowana.

Ponadto kluczowe jest ustawienie odpowiednich zmiennych środowiskowych w następujący sposób:

* Do zmiennych środowiskowych użytkownika należy dodać/ustawić:
  + PYTHONPATH=C:\Program Files\QGIS Valmiera\apps\Python27\Lib\site-packages;%PYTHONPATH%
  + PYTHONPATH=C:\Program Files\QGIS Valmiera\apps\Python27;%PYTHONPATH%
  + PYTHONPATH=C:\Program Files\QGIS Valmiera\apps\qgis\python\qgis;%PYTHONPATH%
  + PYTHONPATH= C:\Program Files\QGIS Valmiera\apps\qgis\python;%PYTHONPATH%
  + PYTHONPATH=C:\Program Files\QGIS Valmiera\bin;%PYTHONPATH%
  + PYTHONPATH= C:\Program Files\QGIS Valmiera\apps\qgis\bin;%PYTHONPATH%
  + PATH=%PYTHONPATH%;%PATH%
  + Może być konieczne ustawienie również zmiennej PYTHONHOME na lokalizację instalacji Pythona np. PYTHONHOME=%PYTHONHOME%;C:\Python27
* Przedstawione zmienne zależne są od lokalizacji instalacji oraz nazwy programu QGIS wybranego do instalacji. Zatem C:\Program Files\QGIS Valmiera\ jest głównym folderem instalacji programu QGIS. Może pojawić się konieczność ustawienie powyższych zmiennych środowiskowych również jako globalne zmienne środowiskowe.
* Zmienne środowiskowe można ustawić klikając prawym przyciskiem myszy na „Mój komputer” i wybierając Właściwości, następnie wchodząc w „Zaawansowane ustawienia systemu” -> „Zmienne środowiskowe”

Ponieważ aplikacja napisana jest w technologii ASP.NET konieczne jest posiadanie zainstalowanej usługi IIS (najlepiej najnowszej), .NET Framework v4.5 oraz MVC 4.

Konfiguracja bazy danych PostgreSQL powinna być następująca:

* Nazwa użytkownika: postgres
* Hasło: boss
* Nazwa bazy: spatial\_index\_comparison
* Port: 5432 Host: localhost
* Konfigurację tą można zmienić w kodzie w konstruktorze klasy PostgreDbService

Ostatnią rzeczą potrzebną do poprawnego działania programu jest dodanie tabeli krajów „countries” do bazy. Dodajemy ją importując plik countries.shp z folderu /data/CountryBoundaries. Reszta tabel powinna stworzyć się sama po uruchomieniu programu. Dlatego też pierwsze uruchomienie programu może trwać długo ze względu na konieczność stworzenia indeksów oraz wygenerowania ponad 10 milionów rekordów.

# Implementacja i wynik pracy

Główną częścią programu są dwa serwisy:

* **PostgreDBService** – serwis łączący się z bazą danych PostgreSQL za pomocą biblioteki Npgsql, umożliwiający wykonywanie zapytań na bazie obliczając przy tym czas ich wykonania. Ponadto serwis ten umożliwia tworzenie tabel oraz indeksów. Dzięki temu serwisowy możliwe jest również „wyłączanie/włączanie” poszczególnych indeksów bez konieczności ich usuwania/tworzenia. Wykorzystywana jest do tego kwerenda:  
  UPDATE pg\_index SET indislive = {true/false}, indisvalid = {true/false} where indexrelid = '{nazwa indeksu}'::regclass;
* **QGisService** – serwis umożliwiający wykonywanie skryptów napisanych w języku Python. Serwis przyjmuje tekst kwerendy, zapisuje go jako plik wykonywalny i już jako pełnoprawny skrypt wykonuje w osobnym procesie w konsoli czekając na wynik w postaci czasu trwania zapytania. Serwis czeka na odpowiedź tylko przez określony czas (timeout), jeśli skrypt wykonywał się za długo proces jest zabijany.

W skryptach „pythonowych” użyto bibliotek:

* Queue – kolejka priorytetowa,
* Timeit – pomiar czasu wykonania zapytania,
* Random – generowanie losowych punktów
* Qgis.core – API aplikacji QGIS

Ponadto program składa się z klas rozszerzających klasę Query czyli kwerend. Każda kwerenda musi umieć wykonać się na obu serwisach. Zarówno tekst kwerend w postaci SQL, jak również w postaci skryptu Python’owego trzymany jest bezpośrednio w kodzie tych klas.

Ponieważ nie można zserializować indeksu przestrzennego tworzonego przez API wystawione przez QGIS, wykonanie każdego skryptu związane jest z utworzeniem go na nowo za pomocą kodu:

index = QgsSpatialIndex()

for f in layer.getFeatures():

index.insertFeature(f)

który niestety wykonuje się dość długo, zatem testy dla dużych zbiorów danych wykonują się bardzo długo. Okazuje się że utworzenie indeksu GIST w PostgreSQL dla tabeli zawierającą 5 milionów rekordów zajmuje znacznie mniej czasu niż utworzenie indeksu RTree z QGIS dla tych samych danych. Dokładnie zbudowanie indeksu GIST zajęło 154,5 sekundy, zaś zbudowanie indeksu RTree za pomocą QGIS 243,3 sekundy.

Czas ten zbadano za pomocą zapytania:

DO $proc$

DECLARE

StartTime timestamptz;

EndTime timestamptz;

Delta interval;

BEGIN

StartTime := clock\_timestamp();

CREATE INDEX random\_points\_5000000\_gist\_idx

ON public.random\_points\_5000000 USING gist (geom);

EndTime := clock\_timestamp();

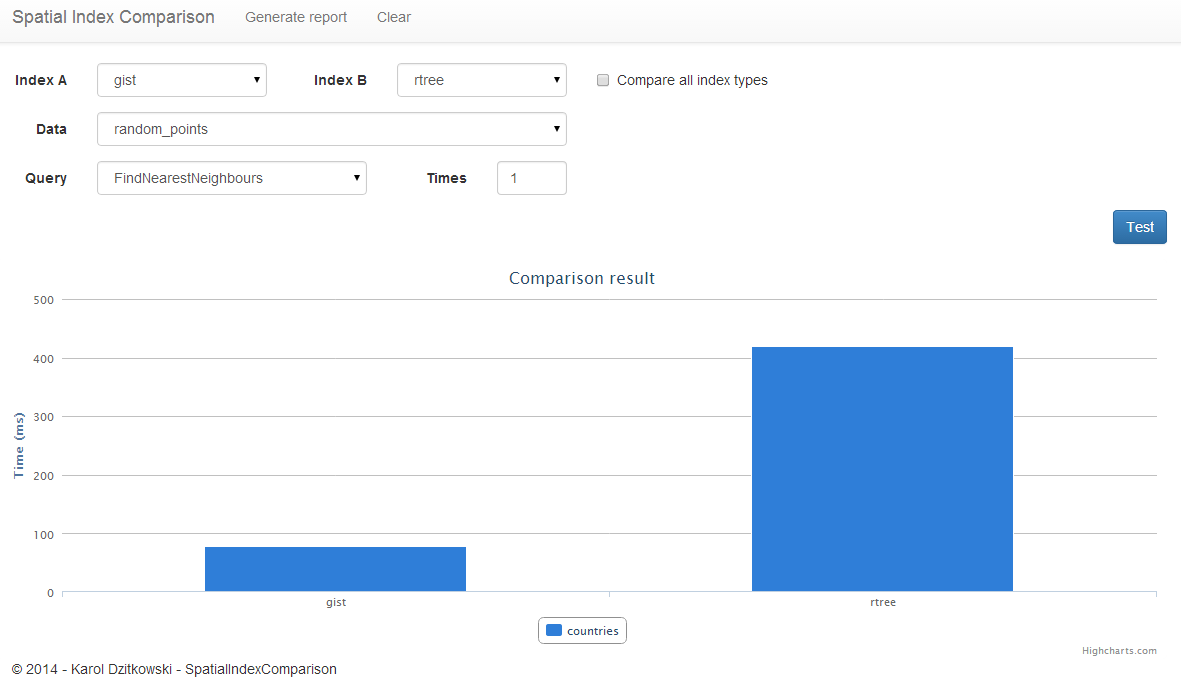
Delta := 1000 \* ( extract(epoch from EndTime) - extract(epoch from StartTime) );

RAISE NOTICE 'Duration in millisecs=%', Delta;

END;

$proc$;

**Rezultat wykonania kwerendy(wygląd głównego widoku programu):**



# Przeprowadzone testy

### Użyte dane:

Testy przeprowadzone zostały na 2 zbiorach danych o różnej charakterystyce. Pierwszym zbiorem danych jest tabela „countries” zawierająca granice krajów, czyli duże i złożone obiekty geometryczne. Drugim zbiorem są tabele zawierające losowe punkty. Każda tabela zawiera określoną ilość punktów, od 500 do 5 000 000 różnych punktów, w celu zbadania czasu wykonania od wielkości danych.

### Zbadane indeksy:

Zbadano indeksy Btree oraz GIST z PostgreSQL oraz indeks RTree z QGIS, a także zbadano czas wykonania zapytań bez żadnego indeksu.

### Rodzaje zapytań:

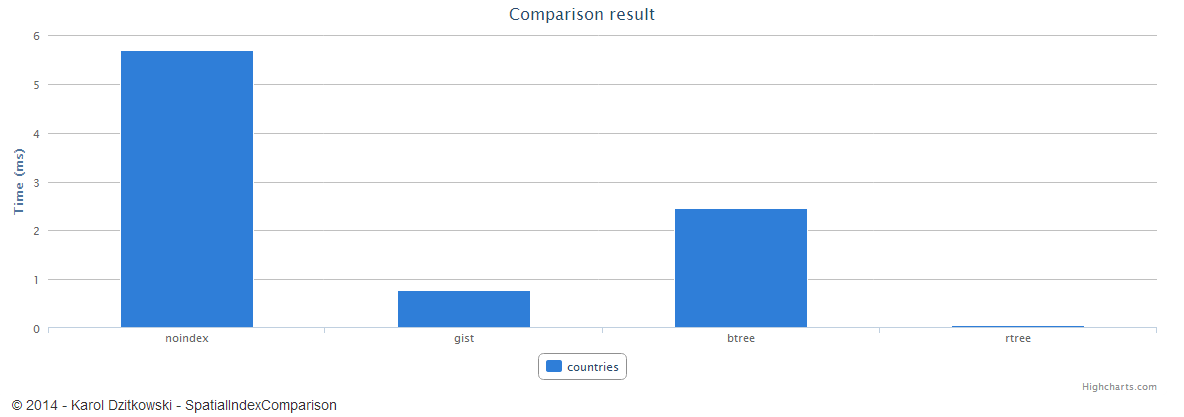
Przeprowadzono testy na trzech różnych zapytaniach:

* FindPointsNearRandomPoints – Wyszukiwanie najbliższych punktów/figur geometrycznych od losowo wybranego punktu.
* FindNearestNeighbours – Znajdowanie najbliższych 5 punktów dla wszystkich punktów po kolei dla danego zbioru
* TouchesAllToAll – Sprawdzanie każdy z każdym elementem ze zbioru czy się dotykają (funkcja ST\_TOUCHES() z PostgreSQL oraz touches() z QGIS

# Wyniki testów

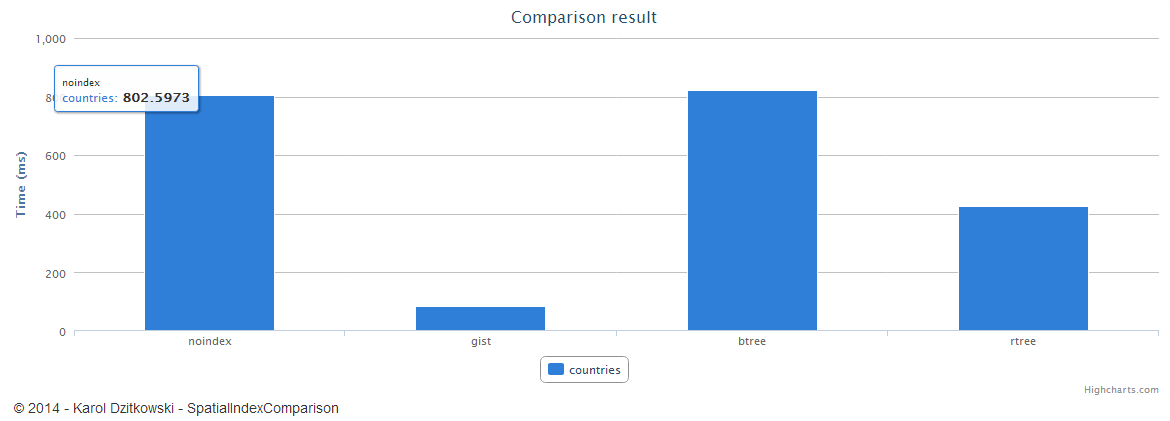
### Dane countries:

Zapytanie FindPointsNearRandomPoints:



|  |  |  |  |
| --- | --- | --- | --- |
| Brak indeksu | B-drzewo | GIST | R-drzewo |
| 5,676 | 2,4518 | 0,76555 | 0,05733 |

Zapytanie FindNearestNeighbours:



|  |  |  |  |
| --- | --- | --- | --- |
| Brak indeksu | B-drzewo | GIST | R-drzewo |
| 802,5973 | 820,0833 | 83,066 | 423,09037 |

Zapytanie TouchesAllToAll dla danych typu countries z zastosowaniem ST\_Touches na bazie PostgreSQL stawiało duże problemy. Dla zapytania:

SELECT \* FROM (SELECT \* FROM countries LIMIT 5) AS A WHERE A.geom IN

(

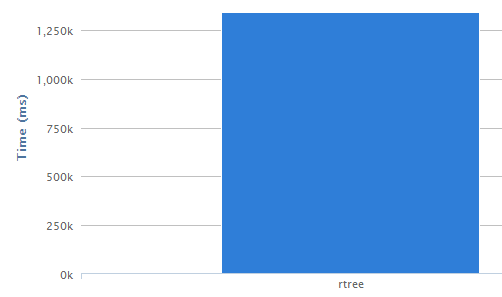
SELECT B.geom

FROM (SELECT \* FROM countries WHERE A.geom && geom ) AS B

WHERE ST\_TOUCHES(A.geom, B.geom)

)

Program zwraca wynik po około 20 sekundach, natomiast na wynik prawdziwego zapytania trzeba czekać ponad 15 minut. Ponadto zastosowanie indeksu w tym indeksu GIST nie ma żadnego znaczenia dla czasu wykonania zapytania! Pomimo że operacja ST\_TOUCHES powinna korzystać z indeksu według dokumentacji PostGIS. Dla porównania zastosowanie indeksu z QGIS zwraca wynik po około sekundzie dla całych danych:



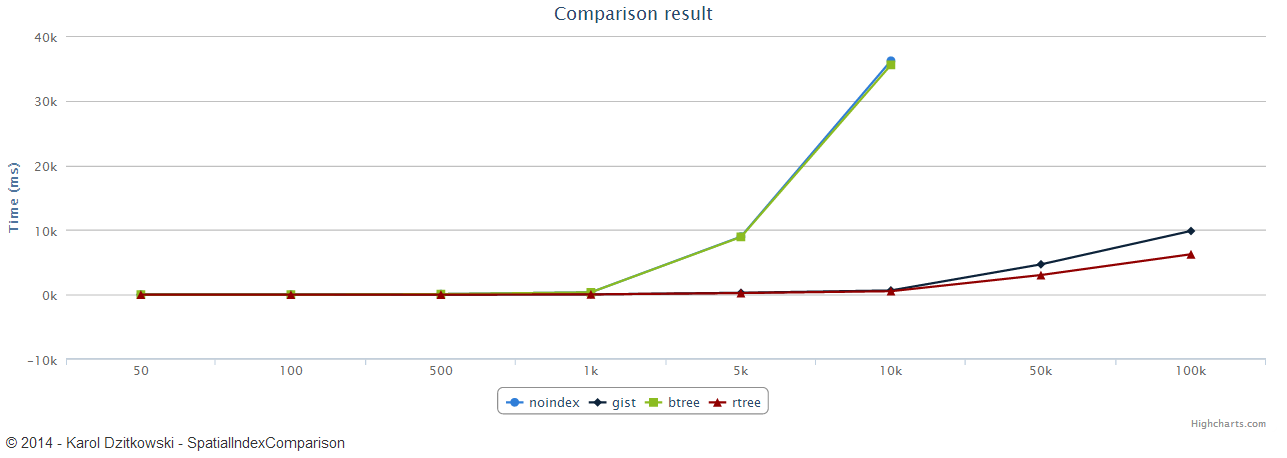
Dzieje się tak najprowdopodobniej dlatego, że ST\_TOUCHES nie radzi sobie dobrze z typami złożonymi jakimi są Multipolygons, szczególnie tak dużymi.

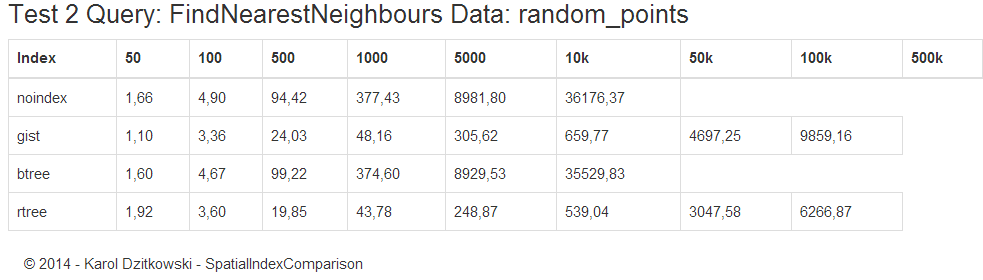
### Dane random\_points:

#### Znajdowanie najbliższych 5 sąsiadów dla każdego punktu

# 

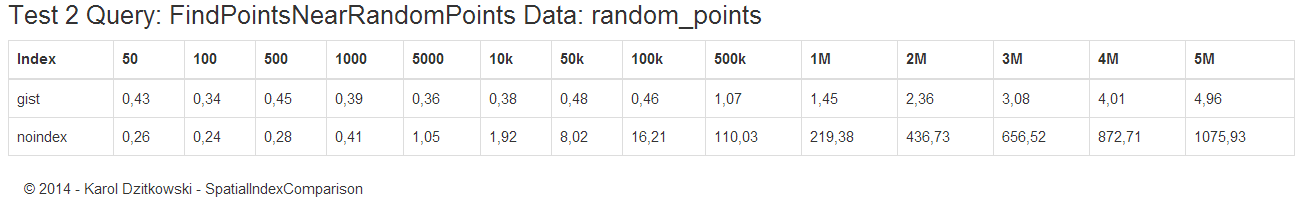
# 



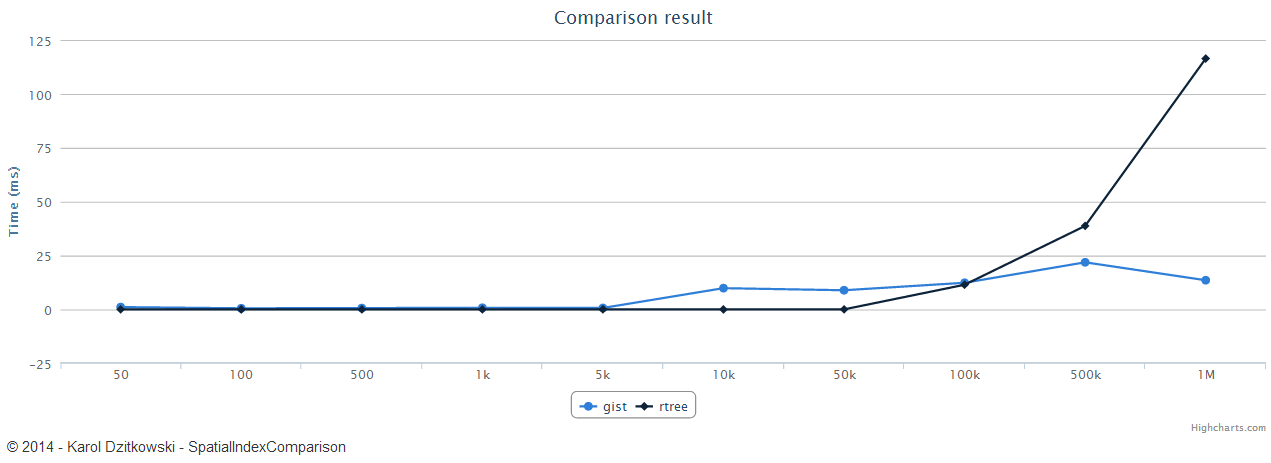


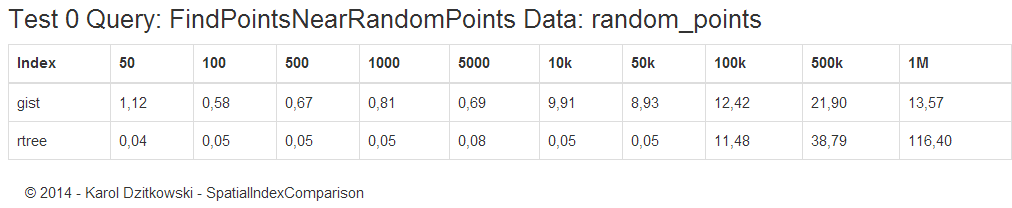
#### Wyszukiwanie punktów najbliższych do losowo wygenerowanego punktu

# 



# 





# Podsumowanie

Wykorzystanie indeksów przestrzennych w znaczącej większości przypadków spowodowało zauważalny wzrost wydajności (skrócenie czasu wykonywania zapytań). Testy wykazały że wzrost wydajności w szczególnych przypadkach może być zauważalny nawet przy konieczności tworzenia indeksu na bieżąco (API QGIS for Python). Dla niektórych zapytań zauważono ponad 1000 krotne przyspieszenie. Jednak pomimo zastosowania indeksu dla niektórych zapytań przy specyficznych danych nie odnotowano przyspieszenia (dane: countries, funkcja ST\_TOUCHES dla PostgreSQL). Z testów wynika również że nie można mówić o przewadze żadnego z indeksów: zastosowanym w PostGIS indeksie GIST, a indeksem wykorzystywanym w QGIS (RTree), ponieważ zależnie od sytuacji jeden albo drugi indeks uzyskiwał lepsze czasy. Na uwagę zasługuje fakt że w historycznych wersjach PostGIS jako indeks używany był indeks rtree zamieniony później na gist. Oczywistym jest fakt że indeks BTree dla danych przestrzennych nie przyniósł zauważalnych korzyści.