Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Zaawansowane problemy baz danych

Optymalizacja danych przestrzennych pod kątem wykorzystania w aplikacjach mobilnych

Sprawozdanie końcowe

Autorzy:
Przemysław Barcikowski
Maciej Suchecki

Prowadzący:
dr inż. Robert Bembenik

16 czerwca 2015

1 Treść zadania

Tytuł Optymalizacja danych przestrzennych pod kątem wykorzystania w aplikacjach mobilnych

Opis Załadować przestrzenne dane z konkursu „Dane po Warszawsku” (https://api.um.warszawa.pl) do wybranego SZPBD razem z odpowiednimi danymi nieprzestrzennymi pod kątem ich wykorzystania w aplikacjach mobilnych (optymalizacja pod kątem częstego korzystania/częstej aktualizacji danych). Przygotować zapytania przestrzenne pokazujące różne aspekty danych.

2 Opis rozwiązania

W ramach zadania skupiliśmy się na danych o stacjach wypożyczania rowerów „Veturilo” znajdujących się na terenie Warszawy. W celu pobrania danych przy użyciu API wspomnianego w treści zadania został napisany skrypt w języku Python, wykorzystujący biblioteki *requests* (obsługa zapytań REST oraz formatu JSON) oraz *psycopg2* (komunikacja z bazą danych). Skrypt po uruchomieniu próbuje połączyć się z bazą danych. Po uzyskaniu połączenia odpytuje on wspomniane API tak, aby pobrać dane o wszystkich zapisanych w bazie stacjach. Przykładowy fragment skryptu zamieszczony jest poniżej.

```
def store_station_in_db(data, cursor, connection):
  object_id = data["featureMemberProperties"][0]["OBJECTID"]
  location = data["featureMemberProperties"][0]["LOKALIZACJA"]
  station_nr = data["featureMemberProperties"][0]["NR_STACJI"]
  bikes = data["featureMemberProperties"][0]["ROWERY"]
  stands = data["featureMemberProperties"][0]["STOJAKI"]
  latitude = data["featureMemberCoordinates"][0]["latitude"]
  longitude = data["featureMemberCoordinates"][0]["longitude"]
  coordinates = "POINT(%s %s)" % (latitude, longitude)
  sql = "INSERT INTO bike_stations "
  sql += "(object_id, location, station_nr, bikes, stands, station_coordinates) "
  sql += "VALUES (%s, %s, %s, %s, %s, ST_Transform(ST_GeomFromText(%s,4326),2059));"
  cursor.execute(sql, (object_id, location, station_nr, bikes, stands, coordinates))
  connection.commit()

# load the bike stations
stations_loaded = 0
for number in range(MIN_STATION_NR, MAX_STATION_NR + 1):
  successful = False
  url = API_URL + "?id=" + MAP_ID + "&apikey=" + API_KEY
  url += "&filter=" + FILTER_PREFIX + str(number) + FILTER_SUFFIX
  while not successful:
    response = requests.get(url)
    if is_response_valid(response):
      store_station_in_db(response.json()["result"], cursor, connection)
      stations_loaded += 1
      successful = True
    else:
      print("ERROR: ", end="")
      if response.status_code == 200:
        print(response.json()["result"])
      else:
        print(str(response.status_code))
```

Listing 1: Fragment skryptu służącego do pobierania danych.

Po pobraniu każdej stacji jest ona zapisywana w bazie danych. W celu przechowywania danych została wykorzystana baza PostgreSQL wraz z wtyczką PostGIS. Każda ze stacji zawiera następujące dane:

- object_id – identyfikator stacji (typ *INT*),
- location – krótki opis dotyczący lokalizacji stacji (typ *VARCHAR(100)*),
- station_nr – numer stacji w systemie w zakresie 6300 - 6467 (typ *INT*),
- bikes – informacja o aktualnej liczbie rowerów znajdujących się na stacji (typ *VARCHAR(10)* z racji na możliwość wystąpienia wartości tekstowej, np. „5+"),
- stands – informacja o dostępnej liczbie stojaków na stacji (typ *INT*),
- station_coordinates – współrzędne geograficzne stacji (typ *GEOMETRY(POINT,2059)*).

Dane o stacjach są przechowywane w tabeli *bike_stations*. W celu optymalizacji zapytań na dane założony został następujący indeks przestrzenny:

```
CREATE INDEX bike_stations_gix ON bike_stations USING GIST (station_coordinates);
```

Listing 2: Indeks przestrzenny założony na danych w tabeli.

3 Testy

W celu przetestowania tak utworzonej bazy danych zostały utworzone następujące przykładowe zapytania SQL, wykonujące obliczenia na podstawie danych przestrzennych:

```
--Zapytanie 1: wyszukanie stacji w zasiegu 5000m od podanego punktu
SELECT * FROM bike_stations
WHERE ST_DWithin(station_coordinates, ST_Transform(
                    ST_GeomFromText('POINT(52.1464422 21.0282027)',4326),2059), 5000);

--Zapytanie 2: odleglosc pomiedzy dwoma stacjami
SELECT ST_Distance(
  (SELECT station_coordinates FROM public.bike_stations WHERE station_nr = 6300),
  (SELECT station_coordinates FROM public.bike_stations WHERE station_nr = 6301))

--Zapytanie 3: znajdz stacje najblizsze dwom punktom
SELECT 'start' AS label, object_id, location, station_nr, bikes, stands, minDist
FROM bike_stations
INNER JOIN (
  SELECT MIN(distance) minDist FROM (
    SELECT *, ST_Distance(station_coordinates,ST_Transform(ST_GeomFromText(
    'POINT(52.1464422 21.0282027)',4326),2059)) distance FROM bike_stations) asd
) minTab ON ST_Distance(station_coordinates,ST_Transform(ST_GeomFromText(
'POINT(52.1464422 21.0282027)',4326),2059)) = minTab.minDist
UNION
SELECT 'end' AS label, object_id, location, station_nr, bikes, stands, minDist
FROM bike_stations
INNER JOIN (SELECT MIN(distance) minDist FROM (
Select *, ST_Distance(station_coordinates,ST_Transform(ST_GeomFromText(
'POINT(52.2773211 20.8567145)',4326),2059)) distance
From bike_stations) asd
) minTab ON ST_Distance(station_coordinates,ST_Transform(ST_GeomFromText(
'POINT(52.2773211 20.8567145)',4326),2059)) = minTab.minDist
```

Listing 3: Zapytania służące do testowania indeksu.

Następnie zapytania zostały wielokrotnie wykonywane odpowiednio z założonym i zdjętym indeksem w celu przetestowania jego wpływu na wydajność zapytań przestrzennych. Z racji na małą ilość dostępnych danych zostały one uprzednio zduplikowane, w celu przybliżenia symulacji do realnych systemów. Ostatecznie zapytania były testowane na tabeli liczącej 21336 wierszy. Wyniki zostały zamieszczone w poniższej tabeli.

Zapytanie	**Wynik z indeksem**	**Wynik bez indeksu**
Zapytanie 1	536ms	540ms
Zapytanie 2	18ms	22ms
Zapytanie 3	92ms	134ms

4 Wnioski