Membuat fitur Search Near Me menggunakan PostgreSQL dengan memanfaatkan Extension Cube dan Earthdistance

wuriyan.to

Ketika kita membangun sebuah aplikasi yang membutuhkan fitur pencarian terdekat dari lokasi pengguna. Misalnya pencarian toko, warung, dan lokasi seseorang, yang paling dekat dengan pengguna aplikasi, kurang lebih flownya sebagai berikut.

- Simpan titik(Longitude dan Latitude) dari tempat-tempat tersebut ke dalam Database .
- Dapatkan **current location** dari si pencari, bisa diambil dari **Mobile device** atau **Web Browser** si pencari. Diasumsikan pencari sudah memberikan **permission-nya**, sehingga **current location** bisa didapatkan.
- Ambil **Longitude dan Latitude** pada **step ke 2**, kirim ke **Backend Server**.
- **Backend Server** melakukan **Query** dan perhitungan jarak antara **Longitude dan Latitude pencari** dengan setiap **Longitude dan Latitude tempat yang ada di Database**. Kemudian simpan setiap jarak dalam variabel **distance** misalnya.
- Kemudian lakukan **Order By secara Ascending** dari semua tempat yang didapatkan dengan variable **distance** tadi.
- **Dapet deh, setiap tempat yang diurutkan dari tempat yang paling dekat dengan lokasi pencari hehe.**

Alhamdulillah sekali, fitur tersebut bisa kita wujudkan hanya dengan memanfaatkan **extension Earthdistance** dan **Cube** pada database **Postgres**. **Tapi jika fitur yang kalian buat butuh perhitungan dengan akurasi tinggi dan butuh fungsionalitas yang lebih kompleks, saya sarankan menggunakan PostGis (https://postgis.net/).**

Pada pembahasan ini kita asumsikan kebutuhan kita hanya untuk mengurutkan setiap lokasi orang, toko, atau warung misalnya, dari lokasi yang paling dekat dengan posisi pencari(pengguna aplikasi yang kita buat).

Fitur ini juga bisa kita manfaatkan ketika anda ingin membuat semacam aplikasi Ojek Online hehe.

Tambahkan **extension earthdistance dan cube**

```
hello=# create EXTENSION IF NOT EXISTS cube;

hello=# create EXTENSION IF NOT EXISTS earthdistance;
```

Setelah menambahkan 2 extension pada slide sebelumnya, seharusnya sekarang kita sudah bisa menghitung **2 titik menggunakan tipe data point**. Misalnya saya akan menghitung jarak dari Bundaran HI ke Stasiun Gambir

Salin Longitude dan Latitude dari dua lokasi tersebut.

Bundaran HI, **Lat** = -6.194664500355808, **Lon** = 106.82343104310507

Stasiun Gambir, **Lat** = -6.176793957131089, **Lon** = 106.83070571150257

Mohon diingat, ketika kita salin dari Google Maps urutannya adalah Latitude kemudian Longitude.

Tetapi ketika kita akan melakukan perhitungan pada tipe data point urutannya adalah Longitude dan Latitude. Jangan sampai terbalik.

```
hello=# select (point(106.82343104310507, -6.194664500355808) <@> point(106.83070571150257, -6.176793957131089)) as distance;
```

Maka akan menghasilkan distance = 1.33201670469851

Mohon diketahui, hasil dari perhitungan menggunakan tipe data point adalah defaultnya adalah **Mile**. Sehingga kita perlu mengkonversinya menjadi **KM**.

1 Mile = 1.609344 KM. Sehingga cukup kalikan hasil tadi dengan 1.609344.

```
hello=# select (point(106.82343104310507, -6.194664500355808) <@> point(106.83070571150257, -6.176793957131089)) * 1.609344 as distance;
```

Maka akan menghasilkan distance = 2.14367309160632 KM, kurang lebih jarak dari **Bundaran HI ke Stasiun Gambir.** Pada Query sebelumnya kita melihat simbol <@>. Simbol tersebut adalah operator qualify yang disediakan extension earthdistance.

Untuk contoh yang lebih real, kita akan buat satu tabel dengan nama **toko**.

Masuk ke console PostgreSQL

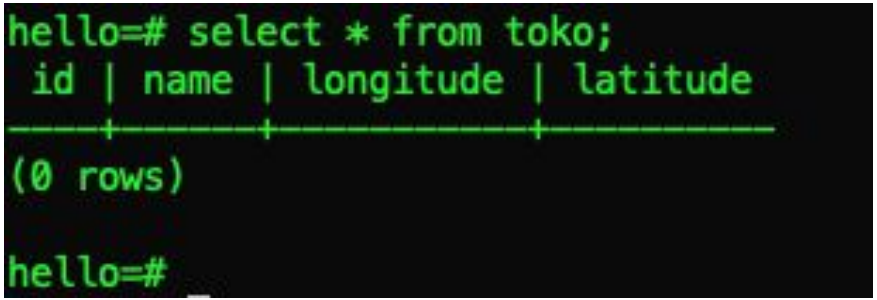
```
$ psql --username=hello --dbname=hello --host=localhost --password

bunga=# \c hello;
```

Buat satu tabel dengan nama **toko**

```
hello=# CREATE TABLE toko (
    ID SERIAL,
    NAME VARCHAR,
    LONGITUDE DECIMAL(10,6),
    LATITUDE DECIMAL(10,6)
);
```

Kurang lebih bentuk tabel toko yang kita buat seperti ini,



Masukan beberapa sampel data lengkap beserta Longitude dan Latitude-nya.

Berikut saya insert beberapa data dengan lokasi random di sekitar Jakarta Selatan.

INSERT INTO toko (name, latitude, longitude) VALUES ('Toko Sabun', -6.196527225698041, 106.832445246591);

INSERT INTO toko (name, latitude, longitude) VALUES ('Toko Baju', -6.208399150275281, 106.83508299988405);

INSERT INTO toko (name, latitude, longitude) VALUES ('Warung Cinta', -6.211069104504896, 106.82496361906891);

INSERT INTO toko (name, latitude, longitude) VALUES ('Nasi Rames', -6.184416633878712, 106.81935239842733);

INSERT INTO toko (name, latitude, longitude) VALUES ('Warung Kopi', -6.243345871677079, 106.83872789525392);

Sebagai contoh, current location pencari yang didapatkan dari Mobile device-nya adalah:

Lat = -6.20868521792064

Lon = 106.82553912882511

Supaya kita tidak mengulangi proses perhitungan, akan lebih baik jika kita buatkan satu fungsi untuk menghitung jarak 2 titik, sekaligus kita konversi menjadi KM.

```
CREATE OR REPLACE FUNCTION get_distance(lon1 FLOAT, lat1 FLOAT, lon2 FLOAT, lat2 FLOAT)
  RETURNS FLOAT
  LANGUAGE plpgsql
AS
$$
DECLARE
  one_mile FLOAT = 1.609344;
  distance_in_mile FLOAT = (point(lon1,lat1) <@> point(lon2,lat2));
BEGIN
  RETURN one_mile * distance_in_mile;
END;
$$;
```

Dengan memanfaatkan fungsi yang sudah kita buat pada slide sebelumnya, maka kita bisa mencari dan mengurutkan setiap tempat dari yang terdekat sampai yang terjauh menggunakan Query berikut.

```
hello=# SELECT name, get_distance(106.82553912882511, -6.20868521792064, toko.longitude, toko.latitude) as jarak from toko ORDER BY jarak ASC;
```

Tempat yang ditampilkan sudah sekaligus diurutkan dari jarak yang terdekat dari current location pengguna aplikasi.

Hasil Query pada slide sebelumya:

```
   name    |     jarak
--------------+-------------------
 Warung Cinta | 0.272581726313255
 Toko Baju    | 1.05548354275359
 Toko Sabun   | 1.55257661510981
 Nasi Rames   | 2.78382528990246
 Warung Kopi  | 4.12061543169028
```

Sekian, semoga bermanfaat