BAB 15

Modul: PostGIS

PostGIS merupakan ekstensi dari PostgreSQL. PostGIS dapat digunakan untuk menangani dan memproses data geografis. Dalam modul ini, kita akan mempelajari bagaimana membuat dan menggunakan fungsi geografis yang ditawarkan pada PostGIS.

Ketika bekerja pada bagian ini, Anda mungkin ingin menyimpan salinan seperti yang ada pada PostGIS cheat sheet yang tersedia di Boston GIS user group[1]. Cara lain yang dapat digunakan adalah membaca dokumentasi online[2].

Lihat juga PostGIS online[3].

15.1 Pelajaran: Pengaturan PostGIS

Pengaturan fungsi PostGIS akan membuat Anda dapat mengakses fungsi-fungsi spasial dari PostgreSQL.

Tujuan dari pelajaran ini: Dapat menginstal fungsi spasial dan mendemonstrasikan secara singkat efek yang dihasilkan.

15.1.1 Instalasi untuk Ubuntu

PostGIS dapat secara mudah diinstal dari *apt*.

```
sudo apt-get install postgis
sudo apt-get install postgresql-9.1-postgis
```

Sungguh hal ini sangat mudah...

[1] http://www.bostongis.com/postgis_quickguide.bqg
[2] http://postgis.refractions.net/documentation/manual-1.5/
[3] http://postgisonline.org/

15.1.2 Instalasi untuk Windows

Kunjungi halaman download[4].

Sekarang ikuti panduan ini[5].

Sedikit lebih rumit, tetapi masih tidak susah. Harap dicatat bahwa Anda perlu terhubung dengan internet untuk dapat menginstal PostGIS *stack*.

15.1.3 Instal plpgsql

Catatan: Anda dapat memastikan bahwa setiap basisdata yang dibuat pada sistem Anda secara otomatis dapat memanfaatkan ekstensi spasial dengan cara menjalankan perintah (dari bagian ini dan dua bagian selanjutnya) pada `template1` sistem basis data *sebelum* Anda membuat basisdata Anda sendiri.

PostgreSQL dapat menggunakan berbagai bahasa prosedural. Apakah bahasa prosedural itu? Bahasa prosedural adalah bahasa 'dalam basisdata' yang dapat digunakan untuk memperluas fungsionalitas dari basisdata. Sebagai contoh Anda dapat menulis fungsi basisdata yang dipanggil saat transaksi basisdata berlangsung- seperti ketika sebuah record dimasukkan ke dalam basis data (Ingat kembali saat hal ini dilakukan dalam modul sebelumnya)

PostGIS membutuhkan bahasa prosedural PLPGSQL yang harus diinstal. Jadi lakukan ini:

```
createlang plpgsql address
```

Dimana argumen ketiga adalah nama basis data yang bahasa proseduralnya harus diinstal didalamnya.

Catatan: Anda membutuhkan akses administratif untuk dapat melakukan hal ini pada basisdata Anda.

15.1.4 Instal postgis.sql

PostGIS dapat dianggap sebagai kumpulan fungsi basis data yang memperluas kemampuan asli dari PostgreSQL sehingga dapat menangani data spasial. Kata 'menangani', maksudnya adalah menyimpan, mengambil, queri dan memanipulasi. Untuk melakukan ini, sejumlah fungsi perlu diinstal ke dalam basis data. Lakukan ini:

```
psql address < /usr/share/postgresql/9.1/contrib/postgis-1.5/postgis.sql
```

atau

```
psql address < /usr/share/postgresql/9.1/contrib/postgis-2.0/postgis.sql
```

tergantung versi PostGIS yang terinstal. Sekarang lakukan:

[4] http://www.postgresql.org/download/
[5] http://www.bostongis.com/PrinterFriendly.aspx?content_name=postgis_tut01

```
psql address
```

dan, begitu Anda berada di prompt psql ketikkan:

```
\df
```

Kita akan membahas fungsi-fungsi ini secara lebih rinci sembari kita menyelesaikan pelatihan ini.

15.1.5 Instal spatial_refsys.sql

Sebagai tambahan bagi fungsi PostGIS, *script* SQL pembantu kedua perlu dijalankan sehingga basisdata yang akan digunakan dapat dipanggil dengan membawa kumpulan definisi sistem referensi spasial atau *spatial reference system (SRS)* seperti yang didefinisikan oleh *European Petroleum Survey Group (EPSG)*. SRS ini digunakan selama operasi misalnya pada transformasi antar sistem koordinat atau *coordinate reference system (CRS)*.

Anda dapat menambahkan daftar SRS apabila diperlukan, namun demikian sebaiknya daftar yang ada hanyalah merupakan SRS yang betul-betul Anda perlukan (*Google Mercator* dan *lo* adalah pengecualian).

Untuk memuat tabel SRS, pertama pastikan bahwa Anda berada di prompt normal (yaitu, pertama berhenti dari basisdata dengan `q`), kemudian melakukan ini:

```
psql address < /usr/share/postgresql/9.1/contrib/postgis-1.5/spatial_ref_sys.sql
```

gantikan `1.5` dengan `2.0` (merujuk pada versi), jika diperlukan.

Perintah di atas menambahkan sebuah tabel pada basisdata kita. Kita bisa melihat skema dari tabel ini dengan memasukkan perintah berikut pada prompt psql:

```
address=# \d spatial_ref_sys
```

Hasilnya seharusnya menjadi seperti ini:

```
Table "public.spatial_ref_sys"
   Column   |          Type           | Modifiers
 -----------+-------------------------+-----------
  srid      | integer                 | not null
  auth_name | character varying(256)  |
  auth_srid | integer                 |
  srtext    | character varying(2048) |
  proj4text | character varying(2048) |
  Indexes:
"spatial_ref_sys_pkey" PRIMARY KEY, btree (srid)
```

Anda dapat menggunakan query SQL standar (seperti yang telah kita pelajari dari bagian pengantar kita), untuk melihat dan memanipulasi tabel ini - kami menyarankan Anda tidak memperbarui atau menghapus records kecuali Anda tahu apa yang Anda lakukan.

Satu SRID yang mungkin menarik Anda adalah EPSG: 4326 – merupakan sistem referensi geografis / *lat lon* menggunakan *elipsoid WGS 84*. Mari kita melihatnya:

```
select * from spatial_ref_sys where srid=4326;
```

Hasil

```
srid      | 4326
auth_name | EPSG
auth_srid | 4326
srtext    | GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS
84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],TOWGS84[0,
0,0,0,0,0,0],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,
AUTHORITY["EPSG","8901"]],UNIT["degree",0.01745329251994328,
AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]
proj4text | +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
```

Srtext adalah definisi proyeksi peta yang ditulis dalam bentuk teks yang sudah cukup familier bagi pengguna SIG(Anda mungkin menyadari hal ini saat melihat file PRJ dalam koleksi shapefile Anda.).

15.1.6 Melihat fungsi PostGIS yang telah diinstal

Baik, Dengan dipasangnya PostGIS, saat ini basisdata PostgreSQL kita memungkinkan untuk mendukung fungsi geospasial. Kita akan menggali lebih hal ini dalam beberapa hari mendatang, namun berikut ini kita akan menunjukkan secara cepat bagaimana ini bekerja . Misalnya, kita ingin membuat suatu titik *(point)* dengan mengetik teks. Pertama, kita menggunakan perintah psql untuk menemukan fungsi yang berkaitan dengan titik:

```
\df *point*
```

Dari list fungsi yang ada, berikut adalah salah satu yang saya tangkap : `st_pointfromtext`

Jadi mari kita mencoba dengan:

```
address=# select st_pointfromtext('POINT(1 1)');
```

Hasil:

```
st_pointfromtext
----------------------------------------------
0101000000000000000000F03F000000000000F03F
(1 row)
```

Jadi ada beberapa hal yang menarik terjadi di sini:

- kita mendefinisikan titik di posisi 1,1 (diasumsikan EPSG: 4326) menggunakan perintah `POINT(1 1)`
- kita memproses pernyataan SQL, tetapi tidak pada sembarang tabel, hanya pada tabel dengan data yang dimasukkan melalui prompt SQL
- baris yang dihasilkan terlihat agak aneh

Baris yang dihasilkan tampak aneh karena hasil tersebut ditulis dalam format OGC yang disebut '*Well Known Binary (WKB)* - lebih lanjut tentang itu akan berada ada bagian berikutnya.

Untuk mendapatkan hasil sebagai teks, setelah saya melakukan listing secara cepat untuk mencari daftar fungsi untuk mengembalikan teks:

```
\df *text
```

Salah satu yang menarik bagi saya adalah perintah `st_astext`. Mari kita gabungkan dengan queri sebelumnya:

```
select st_astext(st_pointfromtext('POINT(1 1)'));
```

Hasil:

```
 st_astext
------------
  POINT(1 1)
  (1 row)
```

Jadi apa yang terjadi di sini? Kita mengetikkan teks `POINT(1,1)`, untuk mengubahnya menjadi titik menggunakan `st_pointfromtext()`, dan mengubahnya kembali ke bentuk yang dapat dibaca oleh manusia dengan `st_astext()`, yang memberikan hasil kepada kita berupa *string* asli.

Salah satu contoh terakhir sebelum kita benar-benar masuk ke dalam detail menggunakan PostGIS yaitu:

```
select st_astext(st_buffer(st_pointfromtext('POINT(1 1)'),1.0));
```

Apa yang dilakukan? Bagian ini menciptakan *buffer* sebesar 1 derajat pada sekitar titik, dan mengembalikannya sebagai teks?

15.1.7 Kesimpulan

Anda sekarang memiliki fungsi PostGIS yang telah diinstal pada salinan PostgreSQL. Dengan ini Anda akan dapat menggunakan fungsi spasial secara luas pada PostGIS.

15.1.8 Apa Selanjutnya?

Selanjutnya Anda akan belajar bagaimana *feature* spasial direpresentasikan dalam basis data.

15.2 Pelajaran: Model Fitur Sederhana

Bagaimana caranya kita dapat menyimpan dan merepresentasikan *feature* geografis dalam sebuah basisdata? Dalam pelajaran ini kita akan membahas salah satu pendekatan, Model Fitur Sederhana seperti apa yang didefinisikan oleh OGC.

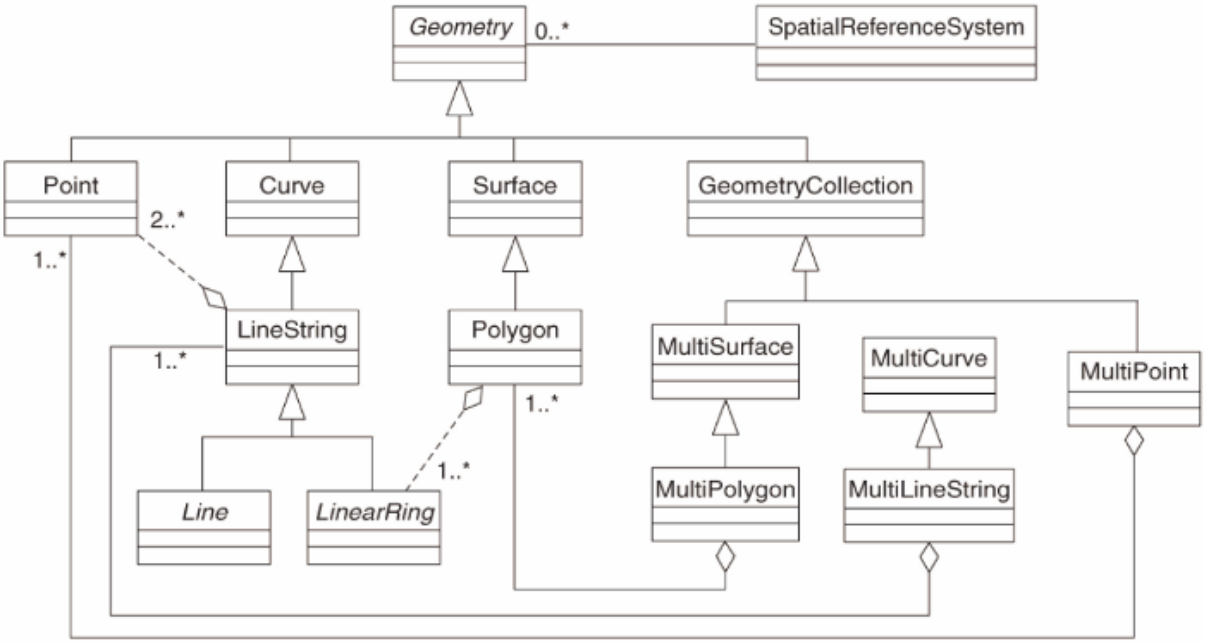
Tujuan dari pelajaran ini: Untuk mempelajari apa itu Model SFS dan bagaimana menggunakannya.

15.2.1 Apa itu OGC

> *The Open Geospatial Consortium (OGC)*, adalah sebuah organisasi standar internasional dengan konsensus sukarela yang berdiri pada tahun 1994. Dalam OGC, bergabung lebih dari 370 organisasi komersial, pemerintah, organisasi nirlaba dan riset di seluruh dunia berkolaborasi melalui proses konsensus terbuka untuk mendorong pengembangan dan penerapan standar bagi konten dan layanan geospasial, pengolahan data SIG dan berbagi data. – *Wikipedia*

15.2.2 Apa itu Model SFS

The Simple feature for SQL (SFS) adalah cara menyimpan data geospasial *tanpa mempertimbangkan topologi* ke dalam basis data dan mendefinisikan fungsi-fungsi untuk akses, operasi, dan pembangunan data geospasial tersimpan.



Model ini menentukan pembentukan data geospasial dari tipe-tipe garis *(Point)*, garis *(Linestring)*, dan Poligon (dan agregasi tipe-tipe tersebut dalam obyek Multi).

Untuk informasi lebih lanjut, kita lihat standar OGC berjudul OGC Simple Feature untuk SQL standar[6].

15.2.3 Menambahkan *field* geometri ke tabel

Mari kita tambahkan *field* berupa titik *(point)* pada tabel *people*:

```
alter table people add column the_geom geometry;
```

[6] http://www.opengeospatial.org/standards/sfs

15.2.4 Menambahkan konstrain berdasarkan tipe geometri

Anda akan melihat bahwa tipe *field* geometri tidak secara implisit menentukan *tipe* geometri untuk fieldnya - oleh karenanya kita perlu konstrain.

```
alter table people
add constraint people_geom_point_chk
    check(st_geometrytype(the_geom) = 'ST_Point'::text OR the_geom IS NULL);
```

Sebenarnya apa yang dilakukan? Perintah di atas menambahkan konstrain pada tabel people untuk mencegah diisinya field *people_geom* dengan nilai selain geometri *point* atau *null*.

Sekarang Anda coba:

> Buatlah sebuah tabel baru dengan nama "kota-kota" dan berikan padanya beberapa kolom yang tepat, termasuk sebuah field geometri untuk menyimpan poligon-poligon batas kota. Pastikan konstrain tersebut telah Anda buat untuk memastikan bahwa geometri-geometri yang disimpan adalah bertipe poligon.

Periksa hasil Anda.

15.2.5 Mengisi tabel kolom geometri

Sampai di sini, Anda seharusnya juga mengisi tabel `geometry_columns`:

```
insert into geometry_columns values
  (’’,’public’,’people’,’the_geom’,2,4326,’POINT’);
```

Mengapa? `geometry_columns` digunakan oleh aplikasi-aplikasi tertentu untuk mengetahui tabel mana dalam basisdata yang berisi data geometri.

Catatan: Jika pernyataan `INSERT` di atas menyebabkan munculnya pesan kesalahan, pertama jalankan queri ini :

```
select * from geometry_columns;
```

Jika kolom `f_table_name` berisi nilai `people`, maka tabel ini telah terdaftar dan Anda tidak perlu melakukan apapun lagi.

Nilai `2` mengacu pada jumlah dimensi, dalam hal ini, dua : **x** dan **y**.

Nilai `4326` mengacu pada sistem proyeksi yang kita gunakan, dalam hal ini, WGS 84 dengan koordinat lintang dan bujur, yang disebut dengan jumlah 4326 (lihat pembahasan sebelumnya tentang EPSG tersebut).

Tambahkan isi yang sesuai untuk *geometry_columns* pada layer kota baru Anda

Periksa hasil Anda.

15.2.6 Menambahkan catatan geometri ke tabel menggunakan SQL

Sekarang, tabel kita sudah *geo-enabled*, artinya kita dapat menyimpan geometri pada tabel ini!

```
insert into people (name,house_no, street_id, phone_no, the_geom)
        values ('Fault Towers',
                34,
                3,
                '072 812 31 28',
                'SRID=4326;POINT(33 -33)');
```

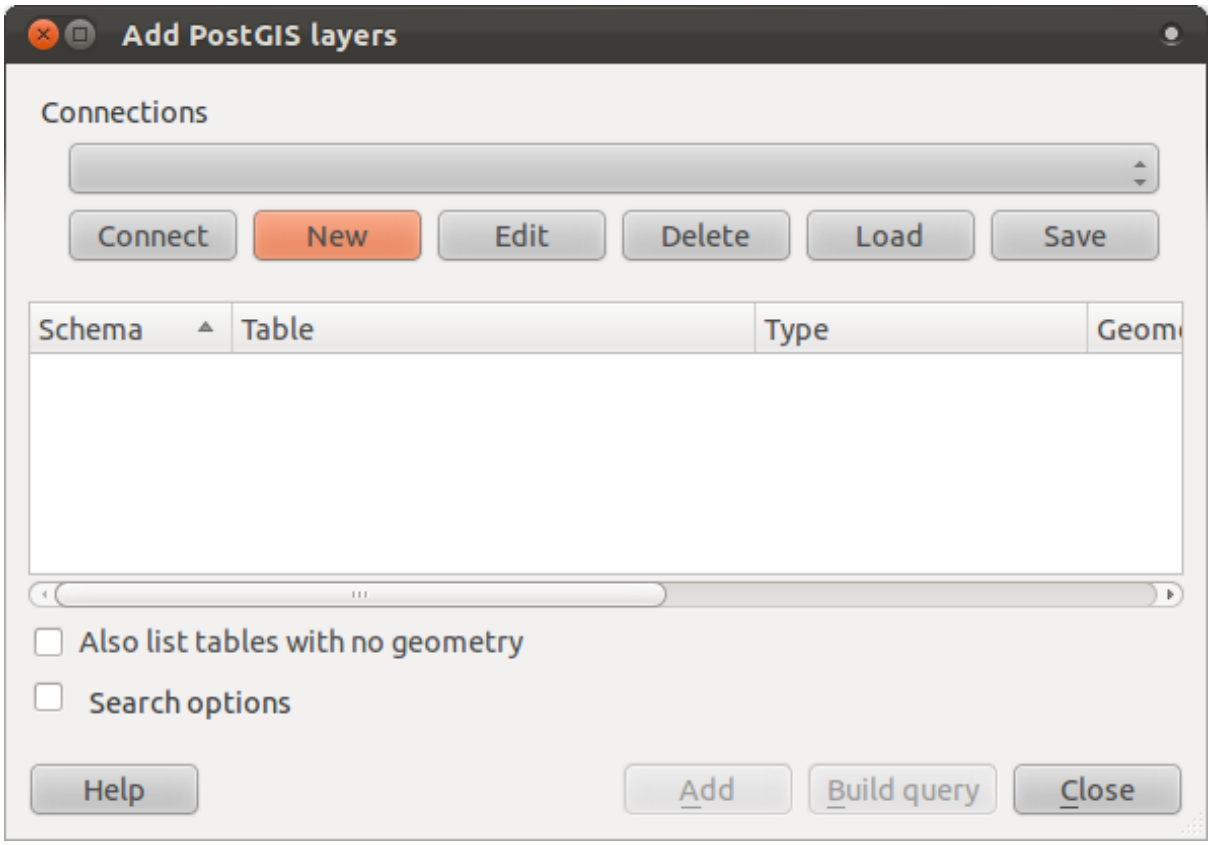
Catatan: Dalam entri baru di atas, Anda perlu menentukan proyeksi (SRID) mana yang Anda ingin gunakan. Hal ini karena Anda menambahkan data geometri berupa titik baru menggunakan string sederhana dari teks, yang tidak secara otomatis menambahkan informasi yang benar tentang sistem proyeksi. Tentu saja, titik-titik baru yang ditambahkan perlu menggunakan SRID yang sama, sehingga Anda harus menentukan itu.

Apabila saat ini Anda menggunakan grafis antarmuka, misalnya, penentuan proyeksi untuk setiap poin akan dilakukan secara otomatis. Dengan kata lain, Anda tidak perlu khawatir untuk menggunakan proyeksi yang benar untuk setiap poin yang ingin Anda tambahkan jika Anda telah tentukan SRID untuk dataset itu, seperti yang kita lakukan sebelumnya.

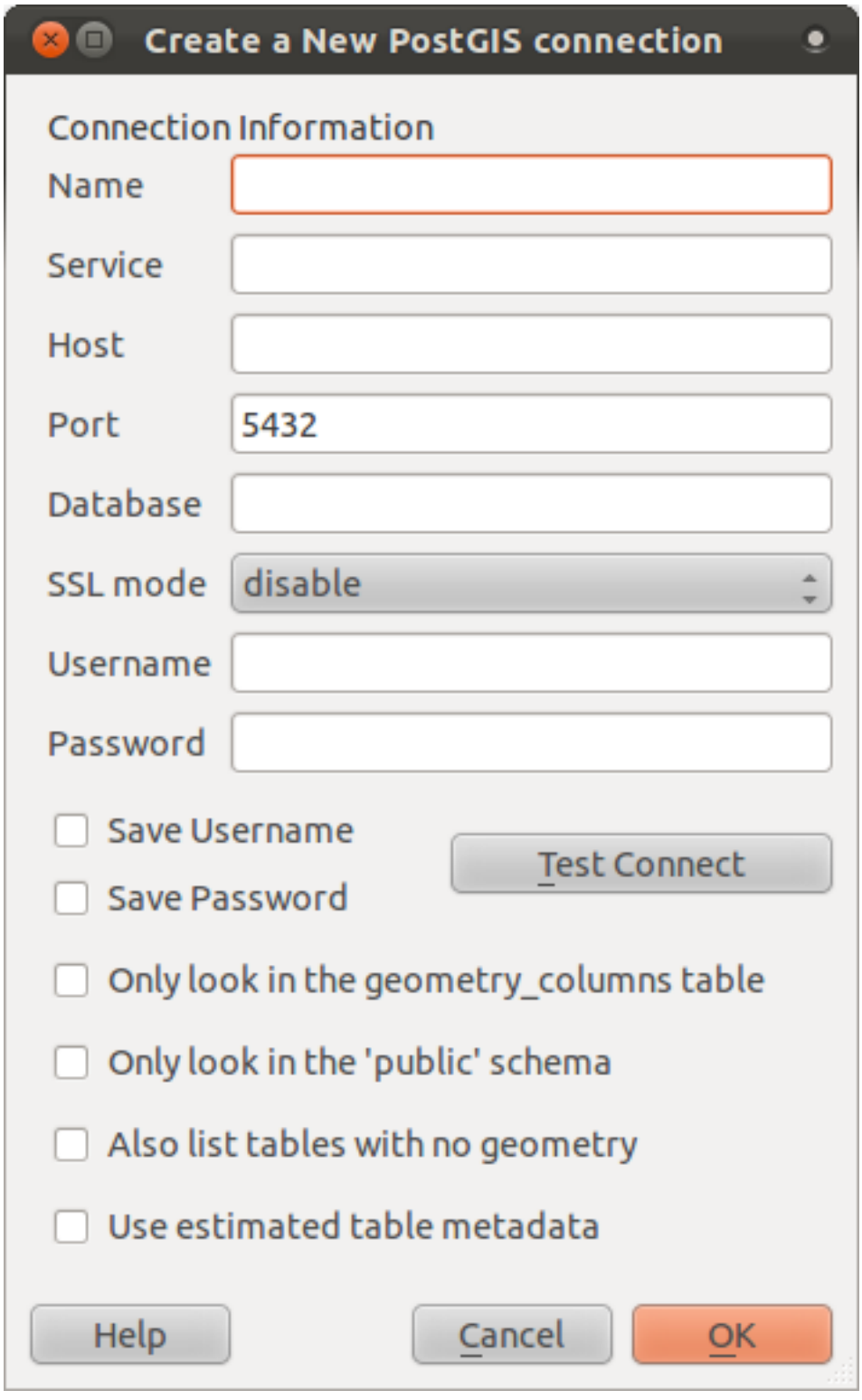
Sekarang mungkin saat yang tepat untuk menjalankan QGIS dan mencoba untuk melihat tabel *people* Anda. Juga, kita harus mencoba mengedit / menambahkan / menghapus records dan kemudian melakukan queri pemilihan terhadap basisdata terbangun untuk melihat bagaimana data telah berubah.

Untuk memuat layer PostGIS di QGIS, gunakan pilihan menu *Layer* → *Add PostGIS Layers* atau tombol toolbar:

Selanjutnya muncul dialog:



Klik pada tombol *New* untuk membuka dialog:



Kemudian definisikan koneksi baru, misalnya:

```
Name: myPG
Service:
Host: localhost
Port: 5432
Database: address
User:
```

`Password:`

Untuk melihat apakah QGIS telah menemukan `address` basisdata dan bahwa *username* dan *password* yang dimasukkan sudah benar, klik *Test Connect*. Jika berhasil, periksa kotak di sebelah *Save Username* dan *Save Password*. Kemudian klik *OK* untuk membuat koneksi ini.

Kembali pada dialog *Add PostGIS Layers* , klik *Connect* dan tambahkan layer yang dikehendaki ke dalam proyek Anda.

> Formulasikan queri untuk menampilkan nama seseorang, nama jalan dan posisi (dari kolom *the_geom*) sebagai teks biasa.

Periksa hasil Anda.

15.2.7 Kesimpulan

Anda telah melihat bagaimana menambahkan obyek spasial pada basisdata Anda dan menampilkannya pada perangkat lunak SIG.

15.2.8 Apa Selanjutnya?

Selanjutnya Anda akan melihat bagaimana melakukan import data ke dalam dan export data dari basis data Anda.

15.3 Pelajaran: *Import* dan *Export*

Tentu saja, basis data tanpa cara untuk memindahkan data ke dan dari nyatidak akan menyenangkan. Terlebih lagi untuk data spasial! Untungnya, tersedia sejumlah piranti yang akan mempermudah Anda untuk memindahkan data ke dan dari PostGIS.

15.3.1 *shp2pgsql*

shp2pgsql merupakan piranti *commandline* untuk melakukan import shapefile ESRI ke dalam basisdata. Di bawah Unix, Anda dapat menggunakan command berikut untuk mengimport tabel PostGIS baru:

```
shp2pgsql -s <SRID> -c -D -I <path to shapefile> <schema>.<table> | \
  psql -d <databasename> -h <hostname> -U <username>
```

Pada Windows, Anda harus melakukannya melalui dua langkah dalam proses import:

```
shp2pgsql -s <SRID> -c -D -I <path to shapefile> <schema>.<table> > import.sql
psql psql -d <databasename> -h <hostname> -U <username> -f import.sql
```

Anda mungkin menemui *error* seperti ini:

```
ERROR:  operator class "gist_geometry_ops" does not exist for access method
"gist"
```

Hal ini merupakan isu populer terkait dengan pembuatan indek spasial secara langsung atau *in situ* untuk data yang Anda import. Untuk menghindari kesalahan, lakukan exclude parameter `-I`. Ini berarti bahwa tidak ada indeks spasial yang dibuat secara langsung, dan Anda harus membuatnya setelah data berhasil diimport (Pembuatan indeks spasial akan dibahas pada pelajaran berikutnya)

15.3.2 *pgsql2shp*

pgsql2shp merupakan piranti *commandline* untuk melakukan export tabel, tampilan atau queri pilihan SQL PostGIS. Untuk melakukannya di bawah Unix:

```
pgsql2shp -f <path to new shapefile> -g <geometry column name> \
  -h <hostname> -U <username> <databasename> <table | view>
```

Untuk melakukan export data menggunakan queri:

```
pgsql2shp -f <path to new shapefile> -g <geometry column name> \
  -h <hostname> -U <username> "<query>"
```

15.3.3 *ogr2ogr*

Ogr2ogr merupakan piranti yang sangat ampuh untuk konversi data ke dalam atau dari PostGIS ke berbagai macam format data. *Org2ogr* merupakan bagian dari perangkat lunak GDAL/OGR

dan harus diisntal secara terpisah. Untuk melakukan export tabel dari PostGIS ke GML, Anda dapat menggunakan perintah ini:

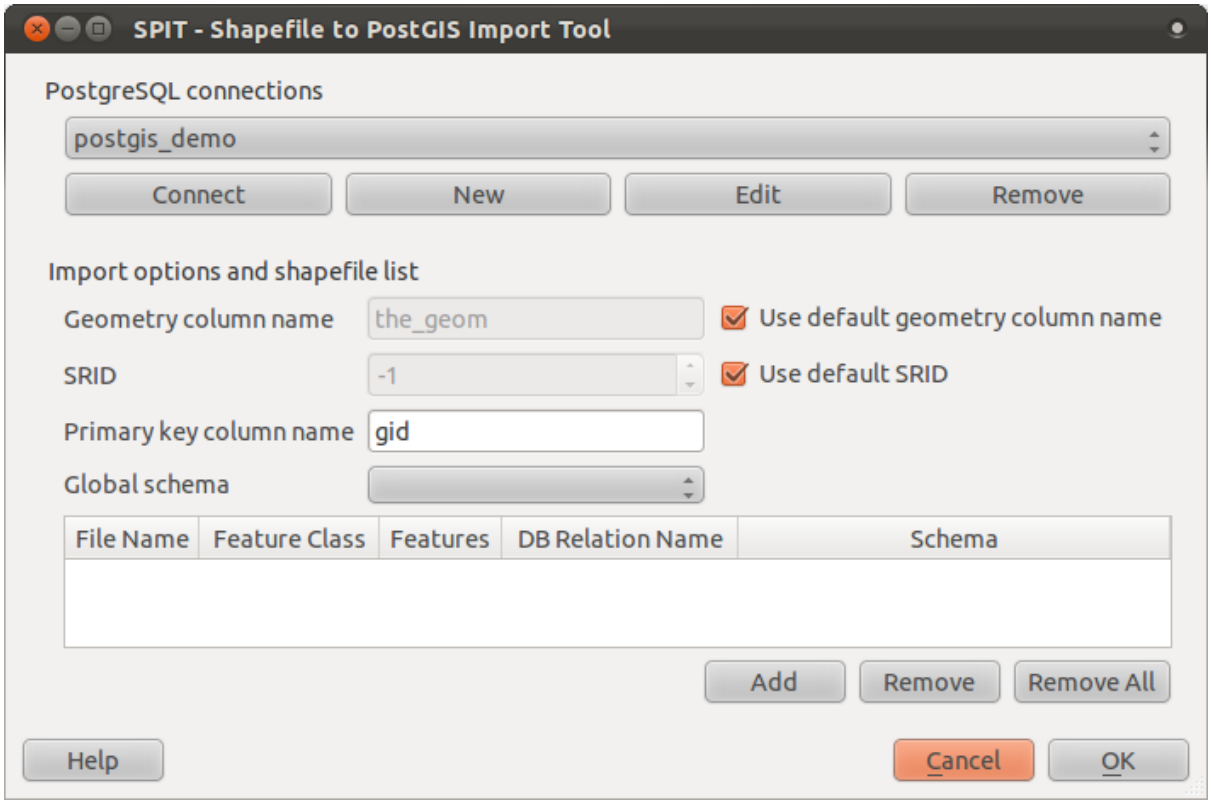
```
ogr2ogr -f GML export.gml PG:'dbname=<databasename> user=<username>
        host=<hostname>' <Name of PostGIS-Table>
```

15.3.4 *SPIT*

SPIT merupakan sebuah *plugin* QGIS yang diinstal bersama dengan QGIS. Anda dapat menggunakan SPIT untuk mengunggah *shapefile ESRI* ke dalam PostGIS.

Setelah Anda menambahkan plugin SPIT via *Plugin Manager*, carilah tombol ini:

Klik pada tombol tersebut, ini akan memberikan dialog SPIT:



Anda dapat menambahkan shapefile ke dalam basisdata dengan mengklik tombol *Add* , yang akan memberikan Anda jendela pencari file.

15.3.5 Kesimpulan

Melakukan import dan export data ke dan dari basis data dapat dilakukan dengan berbagai cara. Khususnya ketika menggunakan sumber data yang tidak sama, Anda mungkin akan menggunakan fungsi ini (atau yang lainnya yang seperti ini) secara rutin.

15.3.6 Apa Selanjutnya?

Selanjutnya, kita akan melihat bagaimana queri terhadap data yang sudah kita buat sebelumnya.

15.4 Pelajaran: Queri Spasial

Queri spasial tidak berbeda dengan queri basisdata yang lain. Anda dapat menggunakan kolom geometri seperti kolom basisdata lain. Dengan instalasi PostGIS pada basisdata kita, kita memiliki fungsi tambahan untuk melakukan queri terhadap basisdata.

Tujuan dari pelajaran ini: Melihat bagaimana fungsi spasial diimplementasikan sebagaimana layaknya melakukan queri tanpa fungsi spasial.

15.4.1 Operator Spasial

Bila Anda ingin mengetahui poin yang berada dalam jarak 2 derajat ke titik (X, Y), Anda dapat melakukan ini dengan:

```
select *
from people
where st_distance(the_geom,'SRID=4326;POINT(33 -34)') < 2;
```

Hasil:

```
 id |     name      | house_no | street_id |   phone_no     |   the_geom
----+---------------+----------+-----------+----------------+------------------
  6 | Fault Towers |       34 |         3 | 072 812 31 28 | 01010008040C0
(1 row)
```

Catatan: Nilai *the_geom* di atas itu terpotong karena keterbatasan ruang di halaman ini. Jika Anda ingin melihat titik sebagai pasangan koordinat yang dapat dibaca oleh manusia, coba sesuatu yang mirip dengan apa yang Anda lakukan pada bagian "Lihat Titik sebagai WKT" di atas.

Bagaimana kita tahu bahwa queri di atas menghasilkan semua titik yang berada pada kisaran 2 *derajat*? Mengapa tidak 2 *meter*? Atau unit lain?

Periksa hasil Anda.

15.4.2 Indeks Spasial

Kita juga dapat mendefinisikan indeks-indeks spasial. Sebuah indeks spasial membuat queri spasial jauh lebih cepat. Untuk membuat indeks spasial pada kolom geometri gunakan:

```
CREATE INDEX people_geo_idx
  ON people
```

```
  USING gist
  (the_geom);
```

Hasil:

```
address=# \d people
Table "public.people"
   Column   |         Type          |                 Modifiers
 -----------+-----------------------+-------------------------------------------
  id        | integer               | not null default
            |                       | nextval('people_id_seq'::regclass)
  name      | character varying(50) |
  house_no  | integer               | not null
  street_id | integer               | not null
  phone_no  | character varying     |
  the_geom  | geometry              |
Indexes:
  "people_pkey" PRIMARY KEY, btree (id)
  "people_geo_idx" gist (the_geom)  <-- new spatial key added
  "people_name_idx" btree (name)
Check constraints:
  "people_geom_point_chk" CHECK (st_geometrytype(the_geom) = 'ST_Point'::text
  OR the_geom IS NULL)
Foreign-key constraints:
  "people_street_id_fkey" FOREIGN KEY (street_id) REFERENCES streets(id)
```

Sekarang Anda coba – memodifikasi tabel kota sehingga kolom geometrinya memiliki indeks pasial.

Periksa hasil Anda.

15.4.3 Demo fungsi spasial PostGIS

Untuk keperluan demo fungsi spasial PostGIS, kita akan membuat basisdata baru yang berisi beberapa data fiksi.

Untuk memulai, buat basis data baru:

```
createdb postgis_demo
```

Jangan lupa untuk menginstal PLPGSQL:

```
createlang plpgsql postgis_demo
```

Kemudian instal fungsi PostGIS dan sistem referensi spasial. Sebagai contoh, di Linux dengan PostgreSQL 9.1 dan PostGIS 1,5:

```
psql postgis_demo < /usr/share/postgresql/9.1/contrib/postgis-1.5/postgis.sql
psql postgis_demo < /usr/share/postgresql/9.1/contrib/postgis-1.5/spatial_ref_sy
```

Selanjutnya, import data yang disediakan dalam `exercise_data/postgis/`. Lihat kembali pelajaran sebelumnya sebagai petunjuk Anda. Anda dapat mengimport dari terminal atau melalui SPIT. Import file ke dalam tabel basis data berikut:

- `points.shp` = `building`
- `lines.shp` = `road`
- `polygons.shp` = `region`

Panggil tiga layer basisdata ini ke dalam QGIS melalui dialog *Add PostGIS Layers*, seperti biasa. Bila Anda membuka tabel atributnya, Anda akan melihat bahwa tiga layer tersebut memiliki kolom `id` dan kolom `gid` yang diciptakan saat proses import PostGIS.

Sekarang setelah tabel berhasil diimport, kita dapat menggunakan PostGIS untuk queri data. Kembali ke terminal *(command line)* dan masukkan prompt psql dengan mengetikkan:

```
psql postgis_demo
```

Kita akan mendemokan beberapa pernyataan pilihan *(select statements)* dengan membuat tampilan hasil *(View)*, sehingga Anda dapat membukanya di QGIS dan melihat hasil pemilihan secara visual.

Select by location

Memilih semua bangunan daerah *Kwazulu*.

```
SELECT a.id, a.name, st_astext(a.the_geom) as point
  FROM building a, region b
    WHERE WITHIN(a.the_geom, b.the_geom)
    AND b.name = 'KwaZulu';
```

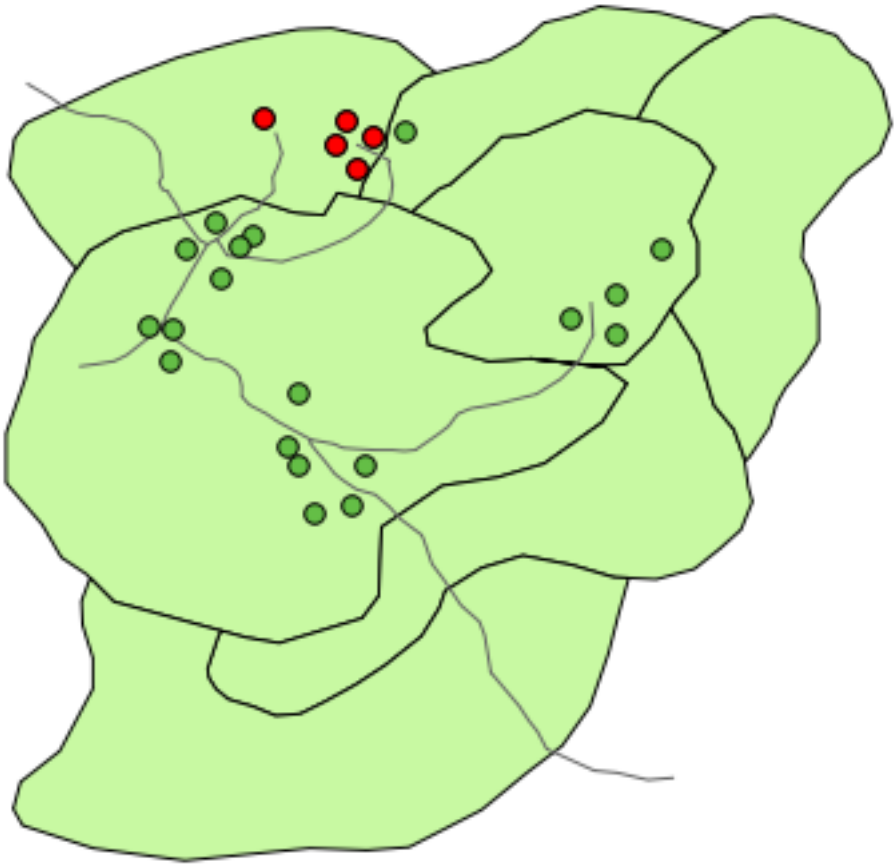
Hasil:

```
 id | name |                  point
----+------+-----------------------------------------
 30 | York | POINT(1622345.23785063 6940490.65844485)
 33 | York | POINT(1622495.65620524 6940403.87862489)
 35 | York | POINT(1622403.09106394 6940212.96302097)
 36 | York | POINT(1622287.38463732 6940357.59605424)
 40 | York | POINT(1621888.19746548 6940508.01440885)
(5 rows)
```

Atau jika kita membuat tampilan *(view)* dari ini maka:

```
CREATE VIEW vw_select_location AS
  SELECT a.gid, a.name, a.the_geom
    FROM building a, region b
      WHERE WITHIN(a.the_geom, b.the_geom)
      AND b.name = 'KwaZulu';
```

dan tampilan pada QGIS adalah:



Select neighbors

Tampilkan list berisi nama-nama daerah yang berdampingan dengan daerah *Hokkaido*.

```
SELECT b.name
  FROM region a, region b
    WHERE TOUCHES(a.the_geom, b.the_geom)
    AND a.name = 'Hokkaido';
```

Hasil:

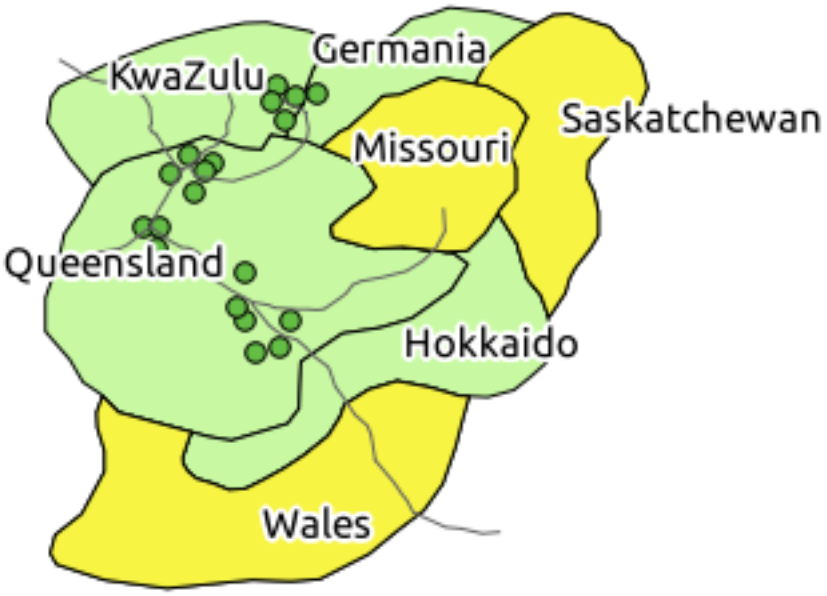
```
    name
-------------
 Missouri
 Saskatchewan
 Wales
(3 rows)
```

Dengan tampilan:

```
CREATE VIEW vw_regions_adjoining_hokkaido AS
  SELECT b.gid, b.name, b.the_geom
    FROM region a, region b
```

```
      WHERE TOUCHES(a.the_geom, b.the_geom)
      AND a.name = 'Hokkaido';
```

Dalam QGIS:

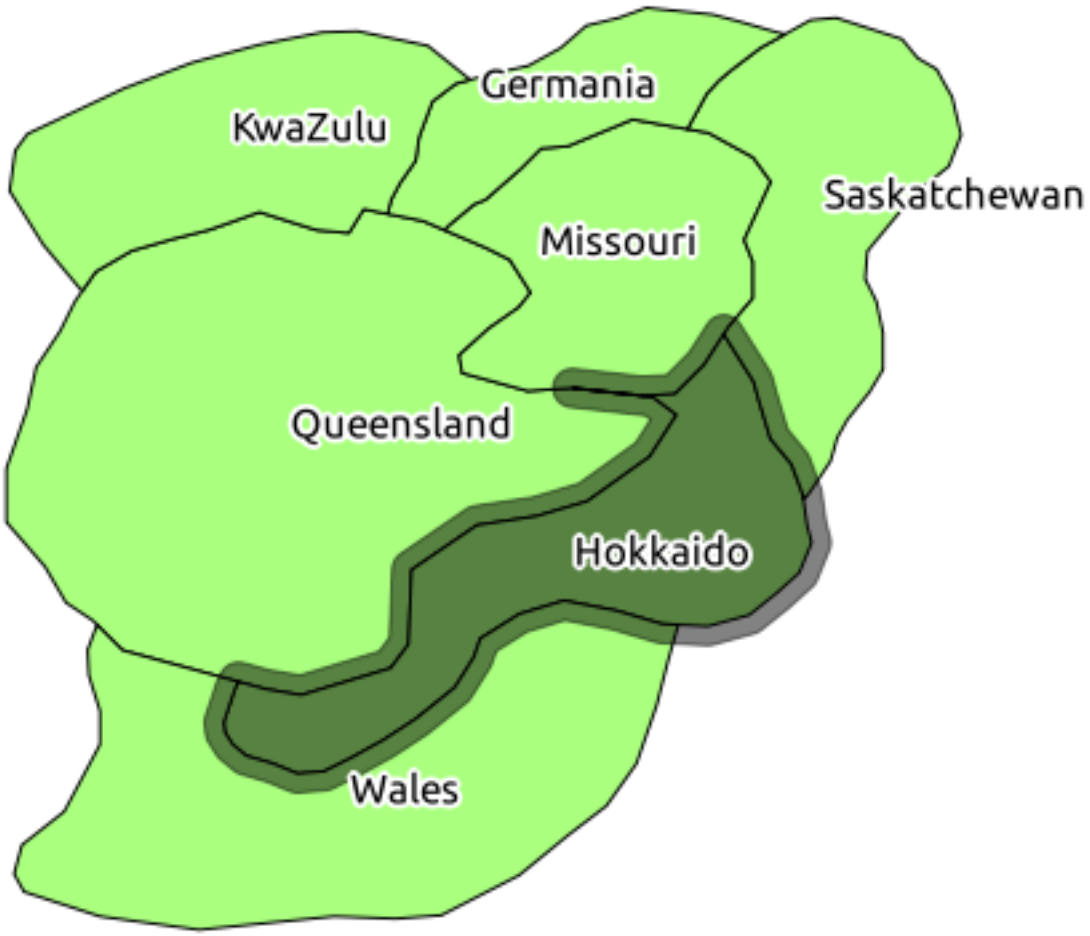


Apabila diperhatikan hasil yang diperoleh, ada wilayah yang hilang *(Queensland)*. Ini mungkin disebabkan karena kesalahan topologi. Hasil seperti ini mengingatkan kita akan potensi masalah dalam data yang kita miliki. Untuk memecahkan teka-teki ini tanpa terjebak dalam anomali data, kita dapat menggunakan *buffer intersect* sebagai gantinya:

```
CREATE VIEW vw_hokkaido_buffer AS
  SELECT gid, ST_BUFFER(the_geom, 100) as the_geom
    FROM region
      WHERE name = 'Hokkaido';
```

Ini membuat *buffer* sejauh 100 m di sekeliling daerah *Hokkaido*.

Area yang lebih gelap adalah hasil buffernya:

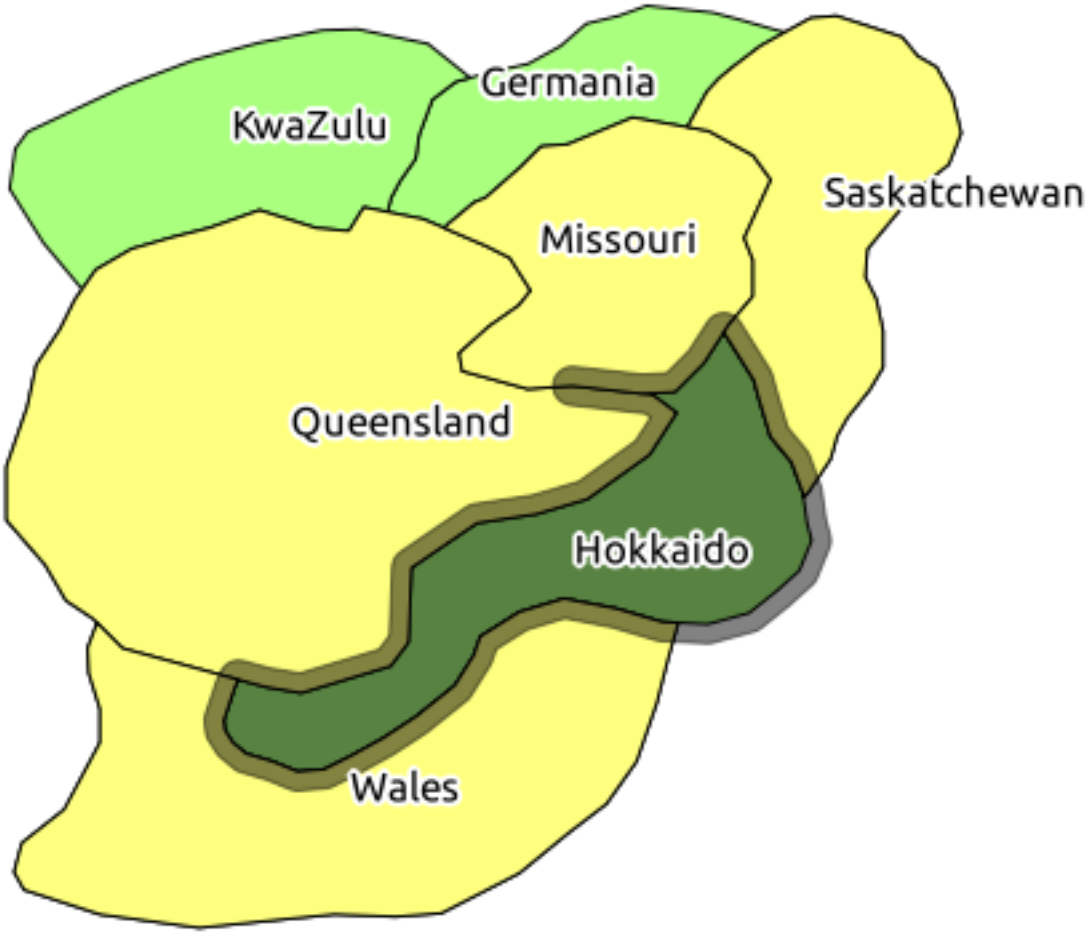


Pilih dengan menggunakan *buffer*:

```
CREATE VIEW vw_hokkaido_buffer_select AS
  SELECT b.gid, b.name, b.the_geom
    FROM
    (
      SELECT * FROM
        vw_hokkaido_buffer
    ) a,
    region b
    WHERE ST_INTERSECTS(a.the_geom, b.the_geom)
    AND b.name != 'Hokkaido';
```

Dalam query ini, tampilan buffer asli digunakan sebagai tabel lain. Hal ini diberikan alias `a`, dan field geometri, `a.the_geom`, digunakan untuk memilih poligon dalam tabel `region` (`b`) yang memotongnya. Namun, Hokkaido itu sendiri dikeluarkan dari pernyataan pilihan, karena kita tidak menginginkannya, kita hanya ingin daerah yang berdampingan dengannya.

Dalam QGIS:



Hal ini juga memungkinkan untuk memilih semua obyek dalam jarak tertentu, tanpa langkah ekstra untuk menciptakan *buffer*:

```
CREATE VIEW vw_hokkaido_distance_select AS
  SELECT b.gid, b.name, b.the_geom
    FROM region a, region b
      WHERE ST_DISTANCE (a.the_geom, b.the_geom) < 100
      AND a.name = 'Hokkaido'
      AND b.name != 'Hokkaido';
```

Queri ini menghasilkan hasil yang sama, tanpa perlu untuk langkah *buffer* sementara:

Select Uniques

Tampilkan daftar nama kota yang unik untuk semua bangunan di wilayah *Queensland*.

```
SELECT DISTINCT a.name
  FROM building a, region b
    WHERE WITHIN (a.the_geom, b.the_geom)
    AND b.name = 'Queensland';
```

Hasil:

```
  name
---------
 Beijing
 Berlin
 Atlanta
(3 rows)
```

Contoh lebih lanjut...

```
CREATE VIEW vw_shortestline AS
  SELECT b.gid AS gid, ST_ASTEXT(ST_SHORTESTLINE(a.the_geom, b.the_geom)) as
    text, ST_SHORTESTLINE(a.the_geom, b.the_geom) AS the_geom
    FROM road a, building b
      WHERE a.id=5 AND b.id=22;

CREATE VIEW vw_longestline AS
  SELECT b.gid AS gid, ST_ASTEXT(ST_LONGESTLINE(a.the_geom, b.the_geom)) as
    text, ST_LONGESTLINE(a.the_geom, b.the_geom) AS the_geom
    FROM road a, building b
      WHERE a.id=5 AND b.id=22;
```

```
CREATE VIEW vw_road_centroid AS
  SELECT a.gid as gid, ST_CENTROID(a.the_geom) as the_geom
    FROM road a
      WHERE a.id = 1;

CREATE VIEW vw_region_centroid AS
  SELECT a.gid as gid, ST_CENTROID(a.the_geom) as the_geom
    FROM region a
      WHERE a.name = 'Saskatchewan';

SELECT ST_PERIMETER(a.the_geom)
  FROM region a
    WHERE a.name='Queensland';

SELECT ST_AREA(a.the_geom)
  FROM region a
    WHERE a.name='Queensland';

CREATE VIEW vw_simplify AS
  SELECT gid, ST_Simplify(the_geom, 20) AS the_geom
    FROM road;

CREATE VIEW vw_simplify_more AS
  SELECT gid, ST_Simplify(the_geom, 50) AS the_geom
    FROM road;

CREATE VIEW vw_convex_hull AS
  SELECT
    ROW_NUMBER() over (order by a.name) as id,
    a.name as town,
    ST_CONVEXHULL(ST_COLLECT(a.the_geom)) AS the_geom
    FROM building a
    GROUP BY a.name;
```

15.4.4 Kesimpulan

Anda telah melihat bagaimana melakukan queri obyek spasial menggunakan fungsi basisdata baru dari PostGIS.

15.4.5 Apa Selanjutnya?

Selanjutnya, kita akan melakukan investigasi struktur geometri yang lebih kompleks dan bagaimana membuatnya dengan menggunakan PostGIS.

15.5 Pelajaran: Konstruksi Geometri

Pada bagian ini kita akan mempelajari sedikit lebih mendalam bagaimana geometri sederhana dibangun di SQL. Pada prakteknya, Anda mungkin akan menggunakan piranti lunak SIG seperti QGIS untuk membuat geometri yang kompleks menggunakan piranti digitasi yang disediakan. Namun demikian, memahami bagaimana geometri dibentuk dapat berguna untuk menulis query dan menambah pemahaman bagaimana basisdata disusun.

Tujuan dari pelajaran ini: Untuk lebih memahami bagaimana membuat entitas spasial langsung dengan PostgreSQL / PostGIS.

15.5.1 Membuat *Linestrings*

Sebelum kita memulai, mari kita ambil tabel jalan yang cocok dengan lainnya; yaitu, memiliki konstrain pada kolom atau *field* geometrinya, memiliki indeks dan sebuah entri dalam tabel *geometry_columns*.

Latihan:

- Memodifikasi tabel jalan sehingga memiliki kolom geometri dengan tipe *ST_LineString*.
- Jangan lupa untuk melakukan update tabel *geometry_coloumns*!
- Juga tambahkan konstrain untuk mencegah geometri yang diisikan bukan *LINESTRINGS* atau *null*.
- Buat indeks spasial pada kolom geometri baru

Periksa hasil Anda.

Sekarang mari masukkan sebuah *linestring* ke dalam tabel jalan. Dalam hal ini kita akan melakukan *update* record jalan terkini :

```
update streets set the_geom = 'SRID=4326;LINESTRING(20 -33, 21 -34, 24 -33)'
where streets.id=2;
```

Lihatlah hasilnya dalam QGIS. (Anda mungkin perlu klik kanan pada layer jalan di panel *'Layers'*, dan pilih *'Zoom to layer extent'*)

Sekarang buat beberapa entri lagi untuk jalan - beberapa di QGIS dan beberapa dari *command line*.

15.5.2 Membuat Poligon

Membuat poligon itu mudah. Satu hal yang perlu diingat adalah bahwa menurut definisi, poligon memiliki minimal empat *vertex*, dengan akhir dan awal menjadi satu lokasi.

```
insert into cities (name, the_geom)
values ('Tokyo', 'SRID=4326;POLYGON((10 -10, 5 -32, 30 -27, 10 -10))');
```

Catatan: Poligon membutuhkan kurung ganda di sekitar daftar koordinatnya, hal ini memungkinkan Anda untuk menambahkan poligon kompleks dengan daerah ganda yang tidak berhubungan. Misalnya:

```
insert into cities (name, the_geom)
values ('Tokyo Outer Wards', 'SRID=4326;POLYGON((20 10, 20 20, 35 20, 20 10),
        (-10 -30, -5 0, -15 -15, -10 -30))');
```

Jika Anda ikuti langkah ini, Anda dapat memeriksa secara visual apa yang terjadi dengan memuat dataset kota ke QGIS, lalu buka tabel atribut, dan pilih entri baru. Perhatikan bagaimana dua poligon baru berperilaku seperti satu poligon.

15.5.3 Latihan: Menghubungkan *City* dengan *People*

Untuk latihan ini Anda harus melakukan berikut ini:

Hapus semua data dari tabel *People* Anda. Tambah kolom *foreign key* pada tabel *people* yang mempunyai referensi *primary key* dari tabel *City*. Gunakan QGIS untuk memilih beberapa kota. Gunakan SQL untuk memasukkan beberapa record orang baru, pastikan bahwa masing-masing memiliki jalan yang terkait dan kota.

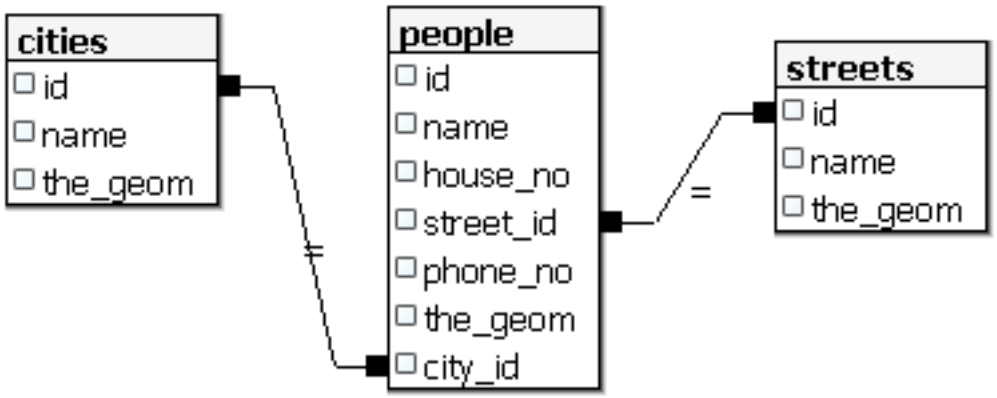
Skema *People* terupdate Anda harus terlihat seperti ini:

```
\d people
Table "public.people"
   Column   |         Type          |                        Modifiers
 -----------+-----------------------+-----------------------------------------
  id        | integer               | not null
            |                       | default nextval('people_id_seq'::regcl
  name      | character varying(50) |
  house_no  | integer               | not null
  street_id | integer               | not null
  phone_no  | character varying     |
  the_geom  | geometry              |
  city_id   | integer               | not null
Indexes:
  "people_pkey" PRIMARY KEY, btree (id)
  "people_name_idx" btree (name)
Check constraints:
  "people_geom_point_chk" CHECK (st_geometrytype(the_geom) =
                       'ST_Point'::text OR the_geom IS NULL)
Foreign-key constraints:
  "people_city_id_fkey" FOREIGN KEY (city_id) REFERENCES cities(id)
  "people_street_id_fkey" FOREIGN KEY (street_id) REFERENCES streets(id)
```

Periksa hasil Anda.

15.5.4 Melihat skema kita

Sekarang skema basisdata kita seharusnya terlihat seperti ini:



15.5.5 Akses sub obyek

Dengan fungsi SFS-model, Anda memiliki beragam pilihan untuk mengakses sub obyek geometri SFS. Ketika Anda ingin memilih *vertex* pertama setiap geometri poligon dalam tabel *myPolygonTable*, Anda harus melakukan ini dengan cara ini:

- Merubah batas poligon menjadi *linestring*:

```
select st_boundary(geometry) from myPolygonTable;
```

- pilih *vertex* pertama dari linestring yang dihasilkan:

```
select st_startpoint(myGeometry)
from (
  select st_boundary(geometry) as myGeometry
  from myPolygonTable) as foo;
```

15.5.6 Pengolahan data

PostGIS mendukung semua fungsi standar OGC SFS/MM. Semua fungsi ini dimulai dengan `ST_`.

15.5.7 Clipping

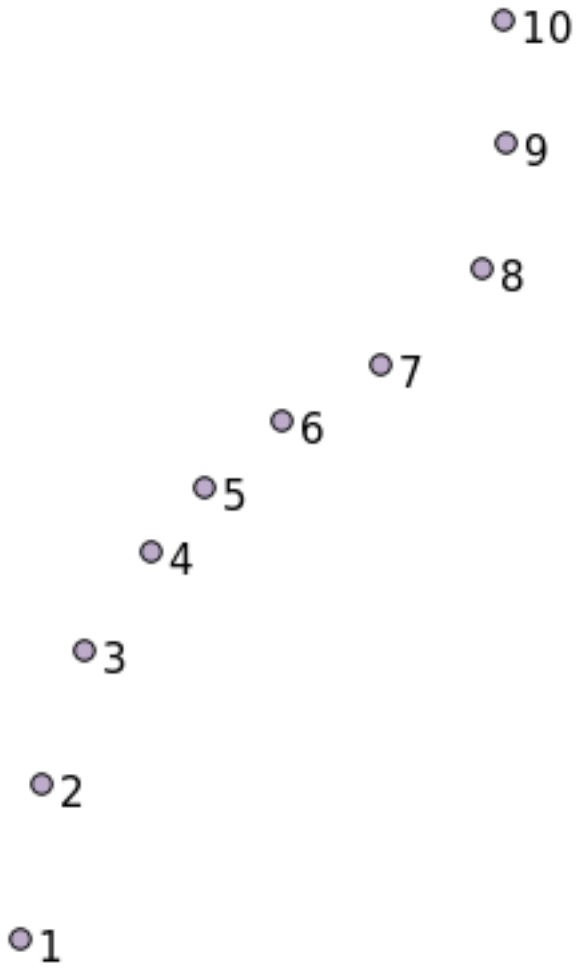
Untuk memotong sebuah sub bagian dari data Anda dapat menggunakan fungsi `ST_INTERSECT()` . untuk menghindari geometri kosong, gunakan:

```
where not st_isempty(st_intersection(a.the_geom, b.the_geom))
```

```
select st_intersection(a.the_geom, b.the_geom), b.*
from clip as a, road_lines as b
where not st_isempty(st_intersection(st_setsrid(a.the_geom,32734),
  b.the_geom));
```

15.5.8 Membangun geometri dari geometri lain

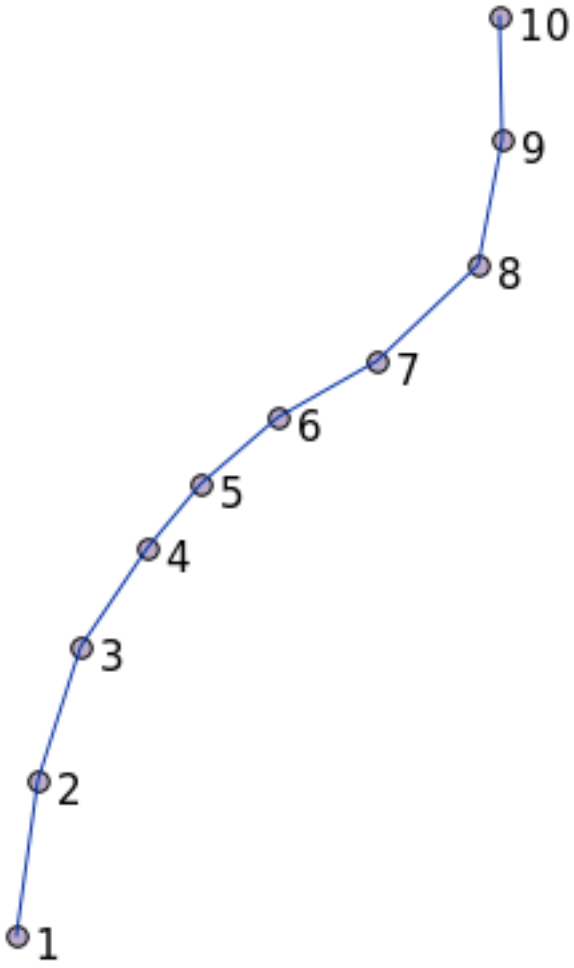
Dari tabel titik tertentu, Anda dapat menghasilkan sebuah linestring. Urutan poin didefinisikan dengan `id` mereka. Metode urutan lain bisa berupa *timestamp*, seperti yang Anda dapatkan pada saat Anda mengumpulkan data titik-titik jalan atau lintasan dengan alat GPS.



Untuk membuat *linestring* dari layer poin dengan nama 'points', Anda dapat menjalankan *command* berikut:

```
select ST_LineFromMultiPoint(st_collect(the_geom)), 1 as id
from (
  select the_geom
  from points
  order by id
) as foo;
```

Untuk melihat bagaimana ini bekerja tanpa membuat layer baru, Anda dapat juga menjalankan *command* pada layer 'people', meskipun tentu saja itu akan membuat sedikit tidak masuk akal pada kenyataannya apabila Anda melakukan hal ini.



15.5.9 Pembersihan Geometri

Anda dapat mendapatkan informasi lebih lanjut untuk topik ini di blog entri ini.

15.5.10 Perbedaan antar tabel

Untuk mendeteksi perbedaan antara dua tabel dengan struktur yang sama, Anda dapat menggunakan kata kunci PostgreSQL `EXCEPT`.

```
select * from table_a
except
select * from table_b;
```

Sebagai hasilnya, Anda akan mendapatkan semua catatan dari *table_a* yang tidak disimpan dalam *table_b*.

15.5.11 *Tablespace*

Anda dapat menentukan harus dimana *postgres* seharusnya menyimpan data pada disk dengan perintah *tablespace*.

```
CREATE TABLESPACE homespace LOCATION '/home/pg';
```

Pada saat Anda membuat basisdata, Anda dapat menentukan *tablespace* mana yang digunakan misalnya:

```
createdb --tablespace=homespace t4a
```

15.5.12 Kesimpulan

Anda telah belajar bagaimana caranya membuat geometri kompleks menggunakan pernyataan *(statements)* PostGIS. Perlu diingat bahwa sebagian besar materi tersebut secara tidak langsung dapat meningkatkan pengetahuan Anda ketika bekerja dengan basisdata yang memiliki dukungan fungsi geospasial pada antarmuka piranti lunak SIG. Pada umumnya Anda tidak harus mengetikkan pernyataan SQL secara manual, tetapi memahami secara umum tentang struktur SQL untuk membentuk geometri akan membantu Anda menggunakan piranti SIG, terutama jika Anda menemukan kesalahan yang secara kasat mata terlihat samar.