BAB 14

Modul: PostgreSQL

PostgreSQL adalah Sistem Manajemen Basis Data (SMBD atau DBMS dalam bahasa Inggris). Dalam modul ini, Anda akan ditunjukkan bagaimana menggunakan PostgreSQL untuk membuat basisdata baru, serta mengakses fungsi-fungsi DBMS lainnya.

14.1 Pelajaran: Berkenalan dengan Basisdata

Sebelum menggunakan *PostgreSQL*, mari kita samakan latar belakang kita dengan mencakup teori basisdata umum. Anda tidak akan perlu mengetikkan kode yang ada di sini, kode-kode tersebut digunakan untuk ilustrasi saja.

Tujuan dari pelajaran ini: Untuk memahami konsep dasar basisdata.

14.1.1 Apa itu Basisdata?

> Sebuah basisdata merupakan kumpulan data yang terorganisir untuk satu atau lebih kegunaan, pada umumnya dalam bentuk digital. - *Wikipedia*
>
> Sebuah sistem manajemen basisdata (DBMS) terdiri dari perangkat lunak yang mengoperasikan basisdata, menyediakan penyimpanan, akses, keamanan, cadangan *(backup)* dan fasilitas lainnya. – *Wikipedia*

14.1.2 Tabel

> Dalam basisdata relasional dan basisdata berkas data *(flat file)*, tabel adalah seperangkat elemen data (nilai) yang disusun menggunakan model kolom vertikal (yang diidentifikasi dengan nama data-data tersebut) dan baris horisontal. Sebuah tabel memiliki kolom dengan jumlah tertentu, dengan jumlah baris bisa berapa saja. Setiap baris diidentifikasi oleh nilai-nilai yang muncul dalam subset kolom tertentu yang merupakan kunci kandidat *(candidate key)*. - *Wikipedia*

```
 id | name  | age
----+-------+-----
```

```
  1 | Tim   |  20
  2 | Horst |  88
(2 rows)
```

Dalam basisdata terminologi basisdata SQL (SQL Basisdata), sebuah tabel juga dikenal sebagai **relasi**

14.1.3 Kolom / *Field*

> Sebuah kolom adalah sekumpulan nilai-nilai data dari tipe sederhana tertentu, dalam hal ini satu untuk setiap baris pada tabel. Kolom memiliki struktur mengikuti baris-baris yang menyusunnya. Istilah *field* sering digunakan sebagai nama lain kolom, meskipun banyak yang menganggap lebih tepat menggunakan istilah medan/*field* (atau nilai *field*) untuk merujuk secara khusus ke item tunggal yang merupakan pertemuan antara sebuah baris dan sebuah kolom. – *Wikipedia*

Kolom:

```
| name  |
+-------+
| Tim   |
| Horst |
```

Field:

```
| Horst |
```

14.1.4 *Record*

Record adalah informasi yang disimpan dalam baris tabel. Setiap *record* mempunyai sebuah *field* untuk tiap kolom dalam tabel.

```
2 | Horst |  88  <-- one record
```

14.1.5 Tipe-tipe Data

> Tipe-tipe data membatasi jenis-jenis informasi yang dapat disimpan dalam sebuah kolom. –*Tim and Horst*

Ada beberapa jenis tipe data. Mari kita simak yang biasa digunakan:

- *String* – untuk menyimpan jenis data teks
- *Integer* – untuk menyimpan semua jenis data *number* (bilangan)
- *Real* – untuk menyimpan data-data bilangan desimal
- *Date* – untuk menyimpan tanggal ulang tahun agar tidak ada seorangpun yang lupa

- *Boolean* – untuk menyimpan nilai benar atau salah yang sederhana

Anda dapat memerintahkan basisdata untuk mempersilakan Anda tidak menyimpan nilai apapun dalam suatu *field*. Jika tidak ada apapun dalam sebuah *field*, maka isi dari *field* tersebut merujuk pada **nilai "null"**.

```
insert into person (age) values (40);

INSERT 0 1
test=# select * from person;
  id | name  | age
 ----+-------+-----
   1 | Tim   |  20
   2 | Horst |  88
   4 |       |  40  <-- null for name
(3 rows)
```

Ada banyak lagi tipe data yang dapat Anda gunakan. –lihat pedoman PostgreSQL![1]

14.1.6 Memodelkan sebuah Basisdata Alamat

Mari kita gunakan studi kasus yang sederhana untuk melihat bagaimana sebuah basisdata disusun. Kita akan membuat sebuah basisdata alamat. Informasi seperti apa yang harus kita simpan?

Tuliskan beberapa properti alamat dalam tempat yang disediakan:

[1] http://www.postgresql.org/docs/current/static/datatype.html

Beberapa properti yang menjelaskan sebuah alamat adalah kolom-kolom. Tipe informasi yang disimpan pada setiap kolom disebut tipe data. Pada bagian selanjutnya kita akan menganalisis tabel konseptual alamat kita di atas untuk menjadikannya lebih baik!

14.1.7 Teori Basisdata

Proses pembuatan basisdata mencakup pembuatan model dari dunia nyata; mengambil konsep dunia nyata dan merepresentasikannya ke dalam basisdata sebagai sekumpulan entitas.

14.1.8 Normalisasi

Salah satu gagasan utama mengapa basisdata diperlukan adalah untuk mencegah adanya duplikasi data/*redundansi*. Proses penghapusan *redundansi* dari basisdata disebut *Normalisasi*.

> Normalisasi adalah cara sistematis untuk memastikan bahwa struktur basisdata mendukung kebutuhan untuk melakukan query umum dan bebas dari karakteristik tertentu yang tidak diharapkan - *insertion, update, and deletion anomalies* – yang dapat menyebabkan hilangnya integritas data. –*Wikipedia*

Ada beberapa jenis "bentuk" normalisasi.

Mari kita simak contoh sederhana berikut:

```
Table "public.people"
  Column  |          Type          |                Modifiers
----------+------------------------+-----------------------------------------
 id       | integer                | not null default
          |                        | nextval('people_id_seq'::regclass)
          |                        |
```

```
 name     | character varying(50)  |
 address  | character varying(200) | not null
 phone_no | character varying      |
Indexes:
 "people_pkey" PRIMARY KEY, btree (id)

select * from people;
id |     name      |           address           |  phone_no
 --+---------------+-----------------------------+-------------
 1 | Tim Sutton    | 3 Buirski Plein, Swellendam | 071 123 123
 2 | Horst Duester | 4 Avenue du Roix, Geneva    | 072 121 122
(2 rows)
```

Bayangkan Anda mempunyai banyak teman dengan nama jalan atau kota yang sama. Setiap saat datanya duplikat dan menghabiskan ruang *(space)*. Terlebih jika nama kota berubah, Anda harus melakukan banyak pekerjaan untuk memperbarui basisdata Anda.

Cobalah untuk mendesain ulang tabel people kita untuk mengurangi duplikasi:

Anda dapat membaca lebih banyak tentang normalisasi basisdata disini[2]

14.1.9 Indek-indek

> Sebuah indek basisdata adalah struktur data yang memiliki kegunaan untuk meningkatan kecepatan operasi pencarian kembali data dalam sebuah tabel pada basisdata. –*Wikipedia*

Bayangkan Anda sedang membaca sebuah buku dan perlu mencari penjelasan dari sebuah konsep, sedangkan buku tersebut tidak mempunyai indek! Anda harus membaca setiap halaman yang ada dan mencari ke seluruh halaman buku sampai Anda menemukan informasi yang Anda butuhkan. Indek pada halaman belakang buku membantu Anda untuk menuju dengan cepat ke halaman tertentu dimana informasi yang relevan tersebut berada.

```
create index person_name_idx on people (name);
```

sekarang pencarian nama akan lebih cepat:

```
Table "public.people"
  Column  |          Type          |                Modifiers
----------+------------------------+-----------------------------------------
 id       | integer                | not null default
          |                        | nextval('people_id_seq'::regclass)
          |                        |
 name     | character varying(50)  |
 address  | character varying(200) | not null
 phone_no | character varying      |
Indexes:
 "people_pkey" PRIMARY KEY, btree (id)
 "person_name_idx" btree (name)
```

14.1.10 *Sequence*

Sequence adalah sebuah *generator* bilangan unik. Biasanya sekuen digunakan untuk membuat *identifier* unik untuk satu kolom dalam sebuah tabel.

Pada contoh berikut, *id* adalah sequence –bilangan ditambahkan, setiap record ditambahkan ke dalam tabel:

```
id |     name     |           address            |  phone_no
---+--------------+------------------------------+------------
 1 | Tim Sutton   | 3 Buirski Plein, Swellendam | 071 123 123
 2 | Horst Duster | 4 Avenue du Roix, Geneva    | 072 121 122
```

[2] http://en.wikipedia.org/wiki/Database_normalization

14.1.11 Diagram Hubungan Entiti *(Diagram ER)*

Dalam sebuah basisdata yang ternormalisasi, Anda biasanya akan memiliki banyak relasi (tabel-tabel). Diagram hubungan entiti *(Entiy Relationship Diagram/ Diagram ER)* digunakan untuk mendesain keterkaitan/ketergantungan logis antara tabel-tabel yang ada. Masih ingat pada tabel *people* kita yang belum ternormalisasi?

```
test=# select * from people;
 id |     name     |           address           |  phone_no
----+--------------+-----------------------------+-------------
 1  | Tim Sutton   | 3 Buirski Plein, Swellendam | 071 123 123
 2  | Horst Duster | 4 Avenue du Roix, Geneva    | 072 121 122
(2 rows)
```

Dengan sedikit usaha kita dapat memisahnya menjadi dua tabel, menjadikan kita tidak perlu mengulang nama jalan untuk individu-individu yang tinggal di jalan yang sama:

```
test=# select * from streets;
 id |     name
----+--------------
 1  | Plein Street
(1 row)
```

dan

```
test=# select * from people;
 id |     name     | house_no | street_id |  phone_no
----+--------------+----------+-----------+-------------
  1 | Horst Duster |        4 |         1 | 072 121 122
(1 row)
```

Kita dapat menghubungkan dua tabel tersebut menggunakan 'keys' `streets.id` dan `people.streets_id`.

Jika dua tabel di atas kita gambarkan *Diagram ER* nya, maka akan tampak seperti berikut:

Diagram ER dapat membantu kita untuk mengekspresikan hubungan "satu ke banyak *(one to many)*" . Pada kasus ini, tanda panah di atas menunjukkan bahwa satu jalan dapat merupakan jalan dimana banyak orang tinggal di dalamnya.

Model *people* kita masih memiliki beberapa masalah terkait normalisasi - cobalah lihat apakah Anda dapat menormalisasikannya lebih jauh dan gambarkan pemikiran Anda ke dalam *Diagram ER*.

14.1.12 Konstrain, *Primary Key* dan *Foreign Key*

Sebuah konstrain basisdata digunakan untuk memastikan bahwa data dalam suatu tabel disimpan tepat sesuai dengan perspektif pembuat model. Misalnya konstrain pada kode pos, Anda bisa memastikan bahwa nomor tersebut berada antara angka `1000` dan `9999`.

Primary key adalah satu atau lebih nilai pada kolom untuk membuat sebuah record unik. Biasanya *primary key* disebut *id* dan berurutan.

Foreign Key digunakan untuk merujuk ke record yang unik pada tabel lain (menggunakan *primary key* tabel yang lain).

Dalam *Diagram ER*, keterkaitan antar tabel biasanya dinyatakan berdasarkan adanya hubungan *foreign key* ke *primary key*.

Jika kita melihat contoh tabel *people* kita, dari definisi tabel yang ada, disitu ditunjukkan bahwa kolom jalan adalah *foreign key* pada tabel *people* dan merujuk pada *primary key* pada tabel *streets*:

```
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      |
Indexes:
"people_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
"people_street_id_fkey" FOREIGN KEY (street_id) REFERENCES streets(id)
```

14.1.13 Transaksi *(Transaction)*

Ketika menambah, mengubah, atau menghapus data dalam sebuah basisdata, basisdata yang dipakai harus dalam kondisi baik ketika terjadi kesalahan. Kebanyakan sistem basisdata menyediakan fitur yang disebut dukungan transaksi *(transaction support)*. Transaksi memungkinkan Anda untuk membuat posisi *rollback* dimana Anda dapat kembali ke posisi semula, apabila modifikasi Anda dalam basisdata tidak berjalan seperti yang direncanakan.

Ambil skenario di mana Anda memiliki sistem akuntansi. Anda perlu untuk mentransfer dana dari satu *account* dan menambahkannya ke yang lain. Urutan langkah-langkah akan seperti ini:

- menghapus R20 dari *Joe*
- menambah R20 untuk *Anne*

Jika ada yang salah selama proses (misalnya mati listrik), transaksi akan dibatalkan.

14.1.14 Kesimpulan

Basisdata memungkinkan Anda untuk mengelola data dengan cara yang terstruktur menggunakan struktur kode sederhana.

14.1.15 Apa Selanjutnya?

Setelah kita melihat bagaimana basisdata bekerja secara teori, sekarang mari kita membuat basisdata baru untuk menerapkan teori yang sudah kita bahas.

14.2 Pelajaran: Mengimplementasikan Model Data

Sebelumnya kita sudah membahas teori, sekarang mari kita membuat basisdata baru. Basisdata ini akan digunakan untuk latihan-latihan kita pada pelajaran yang akan kita ikuti.

Tujuan untuk pelajaran ini: Menginstal perangkat lunak yang diperlukan dan menggunakannya untuk mengimplementasikan contoh basisdata kita.

14.2.1 Menginstal PostgreSQL

Bekerja dalam Ubuntu:

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

Anda seharusnya akan mendapatkan pesan seperti di bawah ini:

```
[sudo] password for timlinux:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
postgresql-client-9.1 postgresql-client-common postgresql-common
Suggested packages:
oidentd ident-server postgresql-doc-9.1
The following NEW packages will be installed:
postgresql-9.1 postgresql-client-9.1 postgresql-client-common postgresql-common
0 upgraded, 4 newly installed, 0 to remove and 5 not upgraded.
Need to get 5,012kB of archives.
After this operation, 19.0MB of additional disk space will be used.
Do you want to continue [Y/n]?
```

Tekan `Y` dan `Enter` dan tunggu proses *download* dan instalasi selesai.

14.2.2 Help

PostgreSQL memiliki dokumentasi online[3] yang sangat baik.

14.2.3 Membuat basisdata pengguna *(user)*

Bekerja dalam Ubuntu:

Setelah instalasi selesai, jalankan perintah ini untuk menjadi pengguna *postgres* dan kemudian membuat pengguna basisdata baru:

```
sudo su - postgres
```

Ketik *password* normal log in Anda ketika diminta (Anda harus memiliki hak *sudo*).

Sekarang, pada *pompt bash* di *postgres user*, buat basisdata *user*. Pastikan nama *user* sesuai nama *login* unik Anda: hal tersebut akan membuat hidup Anda lebih mudah, karena *postgres* secara otomatis akan melakukan pencocokan otentik terhadap Anda ketika Anda *login* sebagai *user* tersebut.

```
createuser -d -E -i -l -P -r -s timlinux
```

Masukkan *password* ketika diminta. Saya biasanya menggunakan *password* yang berbeda untuk unik *login* saya.

[3] http://www.postgresql.org/docs/9.1/static/index.html

Apa maksud dari pilihan-pilihan berikut?

```
-d, --createdb     role can create new databases
-E, --encrypted    encrypt stored password
-i, --inherit      role inherits privileges of roles it is a member of (default)
-l, --login        role can login (default)
-P, --pwprompt     assign a password to new role
-r, --createrole   role can create new roles
-s, --superuser    role will be superuser
```

Sekarang Anda harus meninggalkan halaman *postgres user's bash shell* itu dengan mengetik:

```
exit
```

14.2.4 Verifikasi akun baru

```
psql -l
```

Harus dilakukan kembali seperti ini:

```
timlinux@linfiniti:~$ psql -l
List of databases
Name      |  Owner   | Encoding | Collation  |   Ctype    |
----------+----------+----------+------------+------------+
postgres  | postgres | UTF8     | en_ZA.utf8 | en_ZA.utf8 |
template0 | postgres | UTF8     | en_ZA.utf8 | en_ZA.utf8 |
template1 | postgres | UTF8     | en_ZA.utf8 | en_ZA.utf8 |
(3 rows)
```

Ketik `q` untuk keluar.

14.2.5 Membuat sebuah basisdata

Perintah `createdb` digunakan untuk membuat basisdata baru. Perintah tersebut harus dijalankan dari *bash shell prompt*.

```
createdb address
```

Anda dapat memverifikasi keberadaan basisdata baru Anda dengan menggunakan perintah ini:

```
psql -l
```

Yang mestinya akan memunculkan hasil seperti berikut:

```
List of databases
Name      |  Owner   | Encoding | Collation  |   Ctype    |   Access privileges
----------+----------+----------+------------+------------+-----------------------
address   | timlinux | UTF8     | en_ZA.utf8 | en_ZA.utf8 |
postgres  | postgres | UTF8     | en_ZA.utf8 | en_ZA.utf8 |
template0 | postgres | UTF8     | en_ZA.utf8 | en_ZA.utf8 | =c/postgres: postgre
```

```
template1 | postgres | UTF8     | en_ZA.utf8 | en_ZA.utf8 | =c/postgres: postgre
(4 rows)
```

Ketik `q` untuk keluar.

14.2.6 Memulai sesi *basisdata shell*

Anda dapat mengoneksikan basisdata Anda dengan mudah seperti berikut:

```
psql address
```

untuk keluar dari basisdata *shell psql*, ketik:

```
\q
```

Untuk *help* menggunakan *shell*, ketik:

```
\?
```

Untuk *help* menggunakan perintah sql, ketik:

```
\help
```

Untuk mendapatkan bantuan *(help)* dengan perintah tertentu, ketik (contoh) :

```
\help create table
```

Lihat juga *Psql cheat sheet* <*../_static/postgres/psql_cheatsheet.pdf*> –tersedia secara online disini[4].

14.2.7 Membuat Tabel dalam SQL

Mari kita memulai membuat beberapa tabel! Kita akan menggunakan Diagram ER kita sebagai panduan. Terlebih dahulu mari kita buat tabel *streets*:

```
create table streets (id serial not null primary key, name varchar(50));
```

`serial` dan `varchar` adalah **tipe-tipe data**. `serial` memerintahkan PostgreSQL agar memulai urutan *integer (AutoNumber)* untuk mengisi `id` secara otomatis untuk setiap record baru. `varchar(50)` memerintahkan PostgreSQL untuk membuat tipe data pada kolom berisi maksimal 50 karakter.

Anda akan melihat bahwa perintah diakhiri dengan `;` - (titik koma). Semua perintah SQL harus diakhiri dengan cara tersebut. Bila Anda menekan *enter*, psql akan melaporkan seperti berikut:

```
NOTICE:  CREATE TABLE will create implicit sequence "streets_id_seq" for
         serial column "streets.id"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "streets_pkey"
         for table "streets"
CREATE TABLE
```

[4] http://www.postgresonline.com/special_feature.php?sf_name=postgresql83_psql_cheatsheet&outputformat=html

Itu berarti tabel telah berhasil dibuat, dengan *primary key* `streets_pkey` menggunakan `streets.id`.

Catatan: Jika Anda menekan kembali tanpa memasukkan sebuah `;` , maka Anda akan mendapatkan prompt seperti ini: `address-#`. Hal ini dikarenakan PG mengharapkan Anda untuk mengetik lebih dari apa yang Anda masukkan. Masukkan `;` untuk menjalankan perintah Anda.

Untuk melihat skema tabel, Anda dapat melakukan hal di bawah ini:

```
\d streets
```

Yang harus menunjukkan sesuatu seperti berikut:

```
Table "public.streets"
 Column |         Type          |            Modifiers
--------+-----------------------+-----------------------------------
 id     | integer               | not null default
        |                       | nextval('streets_id_seq'::regclass)
 name   | character varying(50) |
Indexes:
  "streets_pkey" PRIMARY KEY, btree (id)
```

Untuk melihat isi tabel Anda, Anda dapat melakukan hal sebagai berikut:

```
select * from streets;
```

Yang harus menunjukkan sesuatu seperti berikut:

```
id | name
---+------
(0 rows)
```

Yang akan memberikan hasil seperti berikut!

Gunakan pendekatan yang ditunjukkan di atas untuk membuat tabel bernama *people*: Tambahkan kolom atau *field* seperti *house_no* (nomor telepon), *street_id* (id jalan/alamat rumah), *name* (nama), *phone_no* (nomor telepon), dll (tidak semua nama yang dimasukkan *valid*: ubah nama-nama tersebut menjadi *valid*).

Tuliskan SQL yang Anda buat di sini:

Solusi:

```
create table people (id serial not null primary key,
                     name varchar(50),
                     house_no int not null,
                     street_id int not null,
                     phone_no varchar null );
```

Skema untuk tabel (*enter* `\d people`) seperti berikut:

```
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     |
Indexes:
  "people_pkey" PRIMARY KEY, btree (id)
```

Catatan: Untuk tujuan *ilustrasi*, kami sengaja menghilangkan konstrain *fkey*.

14.2.8 Membuat Tombol-tombol di SQL

Solusi kita di atas masih bermasalah. Masalah dengan solusi kita di atas adalah bahwa basisdata tidak tahu bahwa antara *people* dan *streets* memiliki hubungan logis. Untuk mengungkapkan hubungan ini, kita harus mendefinisikan *foreign key* yang menunjuk ke *primary key* dari tabel *streets*.

There are two ways to do this:

- menambah kunci/*key* setelah tabel dibuat
- mendefinisikan kunci/*key* pada saat membuat tabel

Tabel kita telah selesai dibuat, jadi mari kita lakukan dengan cara yang pertama:

```
alter table people
  add constraint people_streets_fk foreign key (street_id) references streets(id
```

Yang menginformasikan kepada tabel `people` bahwa kolom `street_id` yang ada harus cocok secara tepat dengan kolom `id` pada tabel `streets`.

Cara yang lebih umum digunakan untuk membuat konstrain adalah mendefinisikannya ketika Anda membuat tabel (cara yang kedua):

```
create table people (id serial not null primary key,
                     name varchar(50),
                     house_no int not null,
                     street_id int references streets(id) not null,
                     phone_no varchar null);
```

setelah menambah konstrain, skema tabel kita akan tampak sebagi berikut:

```
\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     |
Indexes:
  "people_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
  "people_streets_fk" FOREIGN KEY (id) REFERENCES streets(id)
```

14.2.9 Membuat indek dalam SQL

Kita ingin mencari nama-nama orang secara cepat. Untuk mengatasi hal tersebut, kita dapat membuat sebuah indek pada nama kolom dari tabel *people*:

```
create index people_name_idx on people(name);

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     |
Indexes:
 "people_pkey" PRIMARY KEY, btree (id)
 "people_name_idx" btree (name)    <-- new index added!
Foreign-key constraints:
 "people_streets_fk" FOREIGN KEY (id) REFERENCES streets(id)
```

14.2.10 Menghapus Tabel di SQL

Jika Anda ingin menghapus tabel, Anda dapat menggunakan perintah `drop`:

```
drop table streets;
```

Pada contoh kita, hal tersebut tidak akan bekerja – Mengapa?

Perlu inspirasi dan pemikiran mendalam untuk mengetahui penyebabnya...

Kadang-kadang Anda tidak ingin memiliki tabel lagi. Mungkin Anda merasa tersakiti dengan semua teman Anda. Bagaimana Anda bisa menghapus mereka semua dengan satu langkah yang mudah? Hapus tabel tentu saja! Tentu saja, saat ini terlalu sayang untuk menghapus dalam satu langkah saja karena banyak latihan yang telah dilakukan dalam tabel kita, tetapi jika Anda benar-benar membenci teman Anda yang banyak, tidak ada yang menghentikan Anda untuk menghapus mereka selamanya:

```
drop table people;
```

Kali ini berjalana dengan baik! Mengapa? Apakah people kurang penting daripada *streets*?

Beberapa alasan mengapa Anda bisa mengeluarkan *people*:

Catatan: Jika Anda benar-benar melakukan perintah itu dan tabel `people` terhapus, sekarang menjadi saat yang tepat untuk membuatnya kembali, karena Anda Akan membutuhkannya di latihan berikutnya.

14.2.11 Catatan mengenai *PG Admin III*

Kami menunjukan kepada Anda bagaimana mengoperasikan perintah-perintah SQL melalui *pompt* psql karena cara itu sangat berguna untuk belajar secara benar tentang basisdata. Namun, ada cara yang lebih cepat dan mudah untuk melakukan banyak hal, sama seperti yang kita tunjukkan kepada Anda. Instal *PGAdminIII* dan Anda kemudian dapat membuat, menghapus, mengubah tabel dll menggunakan '*point* and klik' operasi melalui sebuah antarmuka/GUI.

Di dalam Ubuntu, Anda dapat menginstalnya seperti ini:

```
sudo apt-get install pgadmin3
```

14.2.12 Kesimpulan

Anda telah berlatih bagaimana membuat sebuah basisdata baru, memulainya dari nol.

14.2.13 Apa Selanjutnya?

Selanjutnya Anda akan belajar bagaimana menggunakan **DBMS** untuk menambahkan data baru.

14.3 Pelajaran: Menambahkan Data ke Model

Model yang kami buat sekarang perlu diisi dengan data yang sesuai dengan maksud pembuatannya.

Tujuan untuk pelajaran ini: Mempelajari cara memasukkan data baru ke dalam model basisdata.

14.3.1 Memasukkan pernyataan

Bagaimana cara Anda menambahkan data ke dalam tabel? Pernyataan sql `INSERT` memenuhi maksud : tersebut:

```
insert into streets (name) values ('High street');
```

Beberapa hal yang perlu diperhatikan:

- setelah nama tabel (`streets`), Anda sebutkan nama kolom yang akan Anda isi (dalam hal ini hanya kolom `name`).
- setelah kata kunci `values`, masukkan daftar nilai field.
- string harus dikutip dengan menggunakan tanda kutip tunggal.
- Anda perhatikan bahwa kita tidak memasukkan nilai untuk kolom `id` - ini dikarenakan nilai kolomid akan ditulis secara otomatis dan berurutan.

- jika Anda mengatur `id` secara manual, Anda dapat membuat masalah serius dengan integritas basisdata Anda.

Anda harus melihat `INSERT 0 1` jika telah berhasil.

Anda dapat melihat hasil pemakaian insert dengan memilih semua data dalam tabel:

```
select * from streets;
```

hasil:

```
select * from streets;
 id |    name
----+-------------
  1 | High street
(1 row)
```

Sekarang Anda coba:

Gunakan perintah `INSERT` untuk menambahkan jalan baru ke dalam tabel `streets`.

Tuliskan sql yang Anda gunakan disini:

14.3.2 Mengurutkan data tambahan sesuai dengan konstrain

Cobalah untuk menambahkan seseorang ke dalam tabel *people* dengan rincian sebagai berikut:

```
Name: Joe Smith
House Number: 55
Street: Main Street
Phone: 072 882 33 21
```

Ingat, kita mendefinisikan nomor telepon sebagai *string*.

Masalah masalah apa yang Anda temui?

Anda akan mendapati laporan kesalahan jika Anda mencoba untuk melakukan hal tersebut tanpa terlebih dahulu membuat sebuah record untuk *Main Street* pada tabel `streets`.

Apa kesalahan yang Anda peroleh?

You should have noticed that:

- Anda tidak dapat menambahkan jalan menggunakan namanya
- Anda tidak dapat menambahkan jalan menggunakan `id` jalan sebelum terlebih dahulu membuat record jalan pada tabel *streets*

Ingat bahwa kedua tabel kita dihubungkan melalui sepasang *Primary/ Foreign Key*. Ini berarti bahwa tidak ada orang yang *valid* yang dapat dimasukkan tanpa adanya record yang *valid* dan sesuai pada tabel *street*.

Berikut adalah bagaimana kita membuat teman kita:

```
insert into streets (name) values('Main Street');
insert into people (name,house_no, street_id, phone_no)
  values ('Joe Smith',55,2,'072 882 33 21');
```

Jika Anda melihat tabel *streets* sekali lagi (menggunakan pernyataan **SELECT** sebelumnya), Anda akan melihat bahwa `id` untuk entri `Main Street` adalah `2`. Itulah mengapa kita hanya bisa memasukkan nilai numeris `2` di atas. Meskipun kita tidak melihat `Main Street` ditampilkan sepenuhnya dalam entri di atas, basisdata akan dapat menghubungkannya menggunakan nilai `street_id` yaitu `2`.

14.3.3 Memilih data

Kami telah menunjukkan kepada Anda sintaks untuk memilih record. Mari kita melihat beberapa contoh berikut:

```
select name from streets;

select * from streets;

select * from streets where name='Main Street';
```

Dalam sesi selanjutnya kami akan membahas lebih detail tentang cara memilih dan menyaring data.

14.3.4 Update data

Apakah Anda ingin membuat perubahan pada beberapa data yang ada? Misalnya nama jalan perlu dirubah:

```
update streets set name='New Main Street' where name='Main Street';
```

Berhati-hatilah pada saat menggunakan pernyataan update tersebut - jika lebih dari satu record sesuai dengan kata kunci `WHERE` Anda, semua record yang sesuai akan diperbarui!

Solusi yang lebih baik adalah dengan menggunakan *primary key* dari tabel untuk menunjuk pasti kepada record yang akan diubah:

```
update streets set name='New Main Street' where id=2;
```

hal di atas seharusnya akan memberikan hasil `UPDATE 1`.

Catatan: Kriteria pernyataan `WHERE` adalah sensitif terhadap kapital (case sensitive) sehingga `Main Street` <> `Main street`

14.3.5 Menghapus Data

Beberapa kali Anda gagal dalam menjalin persahabatan dengan orang-orang. Kedengarannya seperti tanda untuk menggunakan perintah `DELETE`!

```
delete from people where name = 'Joe Smith';
```

sekarang, kita lihat tabel *people*:

```
address=# select * from people;
  id | name | house_no | street_id | phone_no
 ----+------+----------+-----------+----------
(0 rows)
```

Latihan: Gunakan keterampilan yang telah Anda pelajari sebelumnya untuk menambahkan beberapa teman baru ke basisdata Anda:

```
       name        | house_no | street_id |   phone_no
-------------------+----------+-----------+---------------
 Joe Bloggs        |        3 |         2 | 072 887 23 45
 IP Knightly       |       55 |         3 | 072 837 33 35
 Rusty Bedsprings  |       33 |         1 | 072 832 31 38
 QGIS Geek         |       83 |         1 | 072 932 31 32
```

14.3.6 Kesimpulan

Sekarang Anda telah mengetahui cara untuk menambahkan data baru ke dalam model basisdata yang telah Anda buat sebelumnya. Harap diingat bahwa jika Anda ingin menambahkan jenis data baru (misalnya tipe data baru), Anda mungkin perlu melakukan modifikasi dan / atau membuat model baru untuk dapat menampung data baru tersebut.

14.3.7 Apa Selanjutnya?

Setelah Anda dapat menambahkan beberapa data, selanjutnya Anda akan belajar bagaimana menggunakan query untuk mengakses data ini melalui berbagai cara.

14.4 Pelajaran: Queri

Ketika Anda mengetik perintah `SELECT ...` yang umum disebut sebagai queri, Anda menginterogasi basisdata untuk mendapatkan informasi.

Tujuan dari pelajaran ini: Mempelajari bagaimana membuat queri untuk menghasilkan informasi yang bermanfaat.

14.4.1 Tindak Lanjut dari pelajaran sebelumnya

Mari kita mengecek apakah Anda telah menambah beberapa orang ke dalam basisdata Anda dengan sukses:

```
insert into people (name,house_no, street_id, phone_no)
            values ('Joe Bloggs',3,1,'072 887 23 45');
insert into people (name,house_no, street_id, phone_no)
            values ('IP Knightly',55,1,'072 837 33 35');
insert into people (name,house_no, street_id, phone_no)
            values ('Rusty Bedsprings',33,1,'072 832 31 38');
insert into people (name,house_no, street_id, phone_no)
            values ('QGIS Geek',83,1,'072 932 31 32');
```

14.4.2 Mengurutkan hasil *(ordering)*

Mari kita melihat daftar orang-orang berdasarkan nomor rumah mereka:

```
select name, house_no from people order by house_no;
```

Hasil:

```
       name       | house_no
------------------+----------
 Joe Bloggs       |        3
 Rusty Bedsprings |       33
 IP Knightly      |       55
 QGIS Geek        |       83
(4 rows)
```

Anda dapat mengurutkannya dengan lebih dari satu kolom:

```
select name, house_no from people order by name, house_no;
```

Hasil:

```
       name       | house_no
------------------+----------
 IP Knightly      |       55
 Joe Bloggs       |        3
 QGIS Geek        |       83
 Rusty Bedsprings |       33
(4 rows)
```

14.4.3 Menyaring hasil *(filtering)*

Sering kali Anda tidak akan ingin melihat satu per satu semua record dalam basisdata- terutama jika terdapat ribuan record dan Anda hanya tertarik melihat satu atau dua saja. Jangan takut, Anda dapat menyaring hasil!

Berikut adalah contoh dari penyaringan/*filter numerik*:

```
address=# select name, house_no from people where house_no < 50;
       name       | house_no
------------------+----------
 Joe Bloggs       |        3
 Rusty Bedsprings |       33
(2 rows)
```

Anda dapat mengkombinasikan penyaringan (didefinisikan menggunakan perintah `WHERE`) dengan pengurutan (didefinisikan menggunakan perintah `ORDER BY`)

```
address=# select name, house_no from people where house_no < 50 order by
address-# house_no;
       name       | house_no
------------------+----------
 Joe Bloggs       |        3
 Rusty Bedsprings |       33
(2 rows)
```

Anda juga dapat menyaring hasil berdasarkan data teks :

```
address=# select name, house_no from people where name like '%i%';
       name       | house_no
------------------+----------
 IP Knightly      |       55
 Rusty Bedsprings |       33
(2 rows)
```

Di sini kita menggunakan perintah `LIKE` untuk menemukan semua nama dengan `i` di dalamnya. Jika Anda ingin mencari serangkaian huruf *string* yang sesuai dengan kebutuhan pencarian, Anda dapat melakukan sesuatu pencarian yang tidak sensitif terhadap kapital *(case insencitive)* menggunakan perintah `ILIKE`:

```
address=# select name, house_no from people where name ilike '%k%';
    name     | house_no
-------------+----------
 IP Knightly |       55
 QGIS Geek   |       83
(2 rows)
```

Hal tersebut di atas mempertemukan setiap orang degan `k` atau `K` dalam nama mereka. Dengan menggunakan perintah `ILIKE`, Anda memperoleh:

```
address=# select name, house_no from people where name like '%k%';
    name    | house_no
 -----------+----------
  QGIS Geek |       83
  (1 row)
```

14.4.4 Penggabungan *(Join)*

Bagaimana jika Anda ingin melihat rincian orang tersebut berikut nama jalan (bukan sebagai *id*)? Untuk melakukan itu, Anda perlu menggabungkan *(join)* dua tabel bersama-sama dalam sebuah queri tunggal. Mari kita melihat sebuah contoh:

```
select people.name, house_no, streets.name
from people,streets
where people.street_id=streets.id;
```

Catatan: Dengan *join*, Anda perlu selalu menyatakan dua tabel darimana informasi tersebut berasal, dalam hal ini *people* dan *streets*. Anda juga perlu menentukan dua kunci yang harus cocok *(foreign key & primary key)*. Jika Anda tidak menentukan itu, Anda akan mendapatkan daftar semua kemungkinan kombinasi dari *people* dan *streets*, tapi Anda sulit untuk mengetahui siapa sebenarnya yang tinggal di jalan tertentu!

Hasil yang benar akan terlihat seperti:

```
       name       | house_no |    name
------------------+----------+-------------
```

```
 Joe Bloggs       |        3 | High street
 IP Knightly      |       55 | High street
 Rusty Bedsprings |       33 | High street
 QGIS Geek        |       83 | High street
(4 rows)
```

Kami akan mengulas kembali *join* saat kita membuat query yang lebih kompleks pada latihan berikutnya. Yang perlu dicatat, *join* menyediakan cara sederhana untuk menggabungkan informasi dari dua atau lebih tabel.

14.4.5 *Subselect*

Pertama, mari kita lakukan sedikit perubahan dengan data kita:

```
insert into streets (name) values('QGIS Road');
insert into streets (name) values('OGR Corner');
insert into streets (name) values('Goodle Square');
update people set street_id = 2 where id=2;
update people set street_id = 3 where id=3;
```

Mari kita lihat dengan cepat data kita setelah perubahan-perubahan tersebut – kita menggunakan kembali queri kita dari sesi sebelumnya:

```
select people.name, house_no, streets.name
from people,streets
where people.street_id=streets.id;
```

Hasil:

```
       name       | house_no |      name
------------------+----------+-----------------
 Rusty Bedsprings |       33 | High street
 QGIS Geek        |       83 | High street
 Joe Bloggs       |        3 | New Main Street
 IP Knightly      |       55 | QGIS Road
(4 rows)
```

Sekarang, mari kita tunjukkan *subselection* pada data ini. Kita ingin menunjukkan orang-orang yang hanya tinggal di `street_id` number `1`.

```
select people.name
from people, (
    select *
    from streets
    where id=1
  ) as streets_subset
where people.street_id = streets_subset.id;
```

Hasil:

```
       name
------------------
 Rusty Bedsprings
 QGIS Geek
(2 rows)
```

Ini adalah contoh yang dibuat-buat dan dalam situasi di atas , mungkin berlebihan. Namun bila Anda harus melakukan penyaringan berdasarkan berdasarkan pilihan, *subselect* akan benar-benar membantu!

14.4.6 *Aggregate Queri*

Salah satu fitur canggih dari sebuah database adalah kemampuannya untuk meringkas data dalam tabel tersebut. Ringkasan ini disebut query agregat *(aggregate queries)*. Berikut adalah contohnya:

```
select count(*) from people;
```

Hasil:

```
 count
-------
     4
(1 row)
```

Jika kita menginginkan ringkasan hitungan dari nama jalan, kita dapat melakukan hal berikut:

```
select count(name), street_id
from people
group by street_id;
```

Hasil:

```
 count | street_id
-------+-----------
     1 |         2
     1 |         3
     2 |         1
(3 rows)
```

Catatan: Dikarenakan tidak ada perintah `ORDER BY` (pada contoh di atas), penyajian hasil pada komputer Anda mungkin tidak akan disajikan sama urutannya seperti contoh disini.

Latihan:

> Ringkaslah orang-orang berdasarkan nama jalan dan tunjukkan nama jalan yang sebenarnya dan bukan *street_id*

Solusi:

```
select count(people.name), streets.name
from people, streets
where people.street_id=streets.id
group by streets.name;
```

Hasil:

```
 count |      name
-------+-----------------
     1 | New Main Street
     2 | High street
     1 | QGIS Road
(3 rows)
```

Catatan: Anda perhatikan dari contoh di atas bahwa kita mempunyai *prefix* nama kolom dengan nama tabel (misal nama people.name dan *street.name*). pemberian *prefix* seperti ini perlu dilakukan agar terhindar dari ambiguitas nama kolom.

14.4.7 Kesimpulan

Anda telah melihat bagaimana *queries* digunakan untuk menghasilkan data dari basisdata Anda sehingga Anda dapat menggali informasi yang berguna.

14.4.8 Apa Selanjutnya?

Anda akan melihat bagaiman membuat tampilan dari *queri* yang telah Anda tulis.

14.5 Pelajaran: *Views*

Ketika Anda menulis queri, Anda perlu menghabiskan banyak waktu dan usaha untuk merumuskannya. Dengan *views*, Anda dapat menyimpan definisi query sql dalam 'tabel virtual' yang selanjutnya dapat digunakan kembali.

Tujuan untuk pelajaran ini: Menyimpan query sebagai sebuah Tampilan *(View)*.

14.5.1 Membuat sebuah *View*

Anda dapat memperlakukan sebuah *view* seperti tabel, tetapi datanya bersumber dari queri. Mari kita membuat tampilan sederhana berdasarkan hal di atas.

```
create view roads_count_v as
  select count(people.name), streets.name
  from people, streets where people.street_id=streets.id
  group by people.street_id, streets.name;
```

Seperti yang Anda lihat satu-satunya perubahan adalah create `create view roads_count_v as` sebagai bagian awal. Yang menyenangkan adalah kita sekarang dapat memilih data dari *view* kita:

```
select * from roads_count_v;
```

Hasil:

```
 count |      name
-------+-----------------
     1 | New Main Street
     2 | High street
     1 | QGIS Road
(3 rows)
```

14.5.2 Memodifikasi sebuah *View*

Sebuah tampilan/*view* tidak tetap, dan tidak memiliki 'data sesungguhnya'. Ini berarti Anda dapat secara mudah mengubahnya tanpa mempengaruhi setiap data dalam basisdata Anda.

```
CREATE OR REPLACE VIEW roads_count_v AS
  SELECT count(people.name), streets.name
  FROM people, streets WHERE people.street_id=streets.id
  GROUP BY people.street_id, streets.name
  ORDER BY streets.name;
```

(Contoh ini juga menunjukkan praktek konvensi baik penggunaan UPPER CASE untuk semua katakunci pernyataan SQL)

Anda akan melihat bahwa kita telah menambahkan suatu perintah `ORDER BY` sehingga baris *view* kita diurutkan dengan baik:

```
 count |      name
-------+-----------------
     2 | High street
     1 | New Main Street
     1 | QGIS Road
(3 rows)
```

14.5.3 Menghapus sebuah *view*

Jika Anda sudah tidak lagi membutuhkan sebuah *view*, Anda dapat menghapusnya seperti berikut ini:

```
drop view roads_count_v;
```

14.5.4 Kesimpulan

Dengan menggunakan *view*, Anda dapat menyimpan sebuah queri dan mengakses hasilnya seolah-olah sebagai tabel.

14.5.5 Apa Selanjutnya?

Kadang-kadang, ketika mengubah data, Anda ingin perubahan tersebut akan berefek pada tempat lain dalam basisdata. Pelajaran selanjutnya akan menunjukkan bagaimana melakukan hal tersebut.

14.6 Pelajaran: *Rules*

> *Rules* memungkinkan "pohon queri" dari sebuah queri yang masuk untuk ditulis ulang. Penggunaan yang biasanya dilakukan adalah untuk menerapkan tampilan/*view*, termasuk *view* yang dapat diupdate. - *Wikipedia*

Tujuan untuk pelajaran ini: Mempelajari cara untuk membuat *rules* baru untuk basisdata.

14.6.1 *Materialised Views* (Peraturan berdasarkan *view*)

Misalnya saja Anda ingin mencatat setiap perubahan *phone_no* dalam tabel *people* Anda ke ke dalam tabel *people_log*. Jadi Anda perlu membuat tabel baru

```
create table people_log (name text, time timestamp default NOW());
```

Pada langkah selanjutnya buatlah sebuah *rule*, yang mencatat setiap perubahan dari *phone_no* dalam tabel *people* ke dalam tabel *people_log*:

```
create rule people_log as on update to people
  where NEW.phone_no <> OLD.phone_no
  do insert into people_log values (OLD.name);
```

untuk menguji *rule* bekerja, mari kita modifikasi nomor telepon:

```
update people set phone_no = '082 555 1234' where id = 2;
```

pastikan bahwa tabel telah terupdate secara benar:

```
 id |    name     | house_no | street_id |   phone_no
----+-------------+----------+-----------+--------------
  2 | Joe Bloggs |        3 |         2 | 082 555 1234
(1 row)
```

Sekarang, berkat *rule* yang kita buat, table *people log* tampak sebagai berikut:

```
    name     |            time
-------------+----------------------------
 Joe Bloggs | 2012-04-23 15:20:56.683382
(1 row)
```

Catatan: Nilai dari *field* `time` akan tergantung pada tanggal dan waktu saat ini.

14.6.2 Kesimpulan

Rule memungkinkan Anda untuk secara otomatis menambahkan atau mengubah data dalam basisdata Anda untuk memonitor perubahan pada bagian lain dari basisdata.

14.6.3 Apa Selanjutnya?

Modul berikutnya akan memperkenalkan Anda kepada *PostGIS*, dimana konsep-konsep basisdata yang sudah kita bahas akan dipakai dan diterapkan untuk data SIG.