
Open Source Campus Agreement

Modul Pelatihan

SQL DENGAN POSTGRES

oleh
Owo Sugiana <sugiana@rab.co.id>

Editor:
I Made Wiryana <mwiryana@rvs.uni-bielefeld.de>

Hak cipta buku ini tetap pada penulis. Tetapi buku ini bebas untuk diperbanyak, dikutip baik sebagian atau seluruhnya ataupun disebar luaskan dalam bentuk elektronik ataupun non-elektronik. Baik untuk tujuan komersial maupun non komersial. Selama penyebutan nama asli pengarang, penerbit, pemberi sponsor serta proyek Open Source Campus Agreement (OSCA) tetap dilakukan.

2001

SQL dengan Postgres

Owo Sugiana <sugiana@rab.co.id>

Editor : I Made Wiryana <mwiryana@rvs.uni-bielefeld.de>

2001

Daftar Isi

Kata Pengantar	vii
Tentang penulis	ix
Pernyataan	xi
1 Pendahuluan	1
2 Instalasi	3
2.1 Instalasi dengan RPM	3
2.2 Kompilasi dari Source	4
3 Mengenal Database	7
3.1 Konektivitas	7
3.2 Membuat Database	8
3.3 Tabel	8
3.4 Membuat Laporan	13
3.5 View	15
3.6 Temporary Table	16
4 Function dan Trigger	17
4.1 PL/pgSQL sebagai Procedural Language	18
5 Select	21
5.1 Meng-COPY tabel	21
5.2 Sub SELECT sebagai Field	21
5.3 Query = Himpunan	22
6 Table Rule	25
6.1 Insert Rule	25
6.2 Update Rule	26
6.3 Delete Rule	26
6.4 Rule vs Trigger	26
7 Transaksi	29
8 Data dari Database Lain	31
8.1 Perintah COPY	31
8.2 Format Lain	31
9 Pojok Admin	35

9.1	User	35
9.2	Grup	36
9.3	Perlindungan Konektivitas	36
9.4	Perlindungan Data	37
9.5	Backup	37
9.6	Cleaning-up	38
10	Aplikasi client	39
10.1	Pgaccess	39
10.2	kpsql	46
11	Tip dan Trik	49
11.1	Format Tanggal	49
11.2	Query Tanpa Tabel	50
11.3	Mengubah String Menjadi Tanggal	51
11.4	Memisahkan date dan time Pada datetime	51
11.5	Penambahan dan Pengurangan Untuk date	51
12	Sekilas Object Oriented dalam Postgres	53

Daftar Gambar

10.1	pgaccess	39
10.2	Menu login ke host	39
10.3	Menu tabel	40
10.4	Menu mengubah isi tabel	40
10.5	Konfirmasi untuk menghapus record	40
10.6	Proses sorting tabel	41
10.7	Penggunaan WHERE	41
10.8	Query builder	42
10.9	Visual designer	42
10.10	Bekerja dengan visual designer	43
10.11	Melink tabel	43
10.12	Operasi drag & drop	43
10.13	Menjalankan query	43
10.14	Pemakaian Query Builder	44
10.15	Report Builder	44
10.16	Menu Report Source	45
10.17	Report field	45
10.18	Menghapus obyek	46
10.19	Meletakkan obyek	46
10.20	Melihat hasil report	47
10.21	kpsql	47
10.22	Login di kpsql	47
10.23	Perintah SQL di kpsql	48

Kata Pengantar

Database merupakan aspek yang sangat penting dalam teknologi informasi. Aplikasi canggih yang mendukung sistem besar perlu didukung oleh database server yang handal, berkinerja tinggi, serta mudah perawatan dan pengembangan. Beberapa pihak bahkan menambahkan satu kriteria lagi, yaitu ketersediaan *source code* untuk lebih menjamin kelangsungan sistem. Postgres memang bukan satu-satunya *database server* yang menawarkan *feature* di atas. Tapi Postgres sudah sejak awal memberikannya. Para perancangannya begitu komitmen terhadap perkembangan Postgres dan kini ia telah digunakan secara meluas untuk berbagai macam aplikasi di banyak platform.

Berangkat dari apa yang dialami Postgres, penulis mencoba untuk menerapkannya sebagai *database server* untuk berbagai aplikasi Linux yang dibuat. Dalam perjalanan penggunaannya, penulis senantiasa membuat catatan-catatan kecil yang berkaitan dengan instalasi, konfigurasi, atau penemuan-penemuan lainnya. Catatan ini kemudian dikompilasi menjadi sebuah artikel kecil, pada awalnya. Seiring perjalanan waktu dalam pengembangan sistem di berbagai proyek database, artikel kecil itu kini telah membesar dan menjadi sebuah buku kecil. Dari catatan, artikel, dan kini buku sebenarnya hanya sebuah dokumentasi yang telah menjadi rujukan bagi penulis untuk hal-hal sama yang pernah ditemui. Kalau ternyata tulisan ini telah menjadi pengisi di tengah lengangnya tulisan tentang Postgres di Indonesia, maka itu sudah merupakan nilai tambah tersendiri yang semoga - bermanfaat bagi para programmer negeri ini, dan masyarakat luas pada umumnya.

Penulisan berawal dari sebuah proyek aplikasi web yang menggunakan PHP sebagai bahasa pemrogramannya, terutama pada bagian instalasi dan administrasi sistem. Untuk bagian SQL dasar, sebenarnya merupakan “konversi” dari tulisan sebelumnya yang membahas database InterBase. Sehingga dokumentasi ini sempat terdiri atas dua judul: Pengenalan Postgres dan SQL dengan Postgres. Proyek lainnya yang berbasis web tetap menggunakan Postgres dan PHP. Diikuti dengan pembuatan *billing system* untuk warnet-kos di RAB¹ yang menggunakan Python sebagai *programming language*. Namun sampai di situ dokumentasi belum mengalami peningkatan kuantitas dan kualitas tulisan dikarenakan proyek-proyek tersebut “mencukupkan” diri pada SQL dasar. Lompatan yang cukup besar terjadi pada pengembangan sistem informasi di RS Pertamina Jaya². Sistem besar ini tidak bisa lagi mencukupkan diri pada “SQL umum”. Banyak *feature* unik di Postgres yang dapat mempercepat proses penyelesaian masalah pemrograman yang dihadapi.

Terima kasih pada RAB yang telah membiayai seluruh operasional penulis, dan juga rekan-rekan dari NCS³ serta KPLI Jakarta⁴ yang mendukung penggunaan Postgres. Tidak lupa kepada para dosen saya di Universitas Gunadarma - khususnya pada I Made Wiryana - yang telah memberikan arahan dan menopang pengembangan pengetahuan penulis di bidang database dan teknologi informasi pada umumnya. Juga kepada rekan Avinanta Tarigan yang senantiasa *online* dalam memberikan saran teknikal yang cukup strategis perihal instalasi dan *setting-up* server.

Penulis

Owo Sugiana

¹<http://www.rab.co.id>

²<http://www.rspj.co.id>

³<http://www.nurulfikri.com>

⁴<http://jakarta.linux.or.id>

Tentang penulis



Owo Sugiana SKom, dilahirkan di Jakarta 19 Oktober 1973. Menamatkan S1 Teknik Informatika di Universitas Gunadarma pada tahun 1999. Mengenal komputer saat duduk di kelas 5 SD (Apple II), dan kini mengambil spesialisasi pada pemrograman aplikasi database. Saat duduk di tingkat III, penulis memperoleh proyek pertamanya untuk pengembangan aplikasi database penjualan tiket pesawat di PT Priaventure. Pernah bekerja di sebuah perusahaan software PT Kemang Hasta Mitra selama 2 tahun pada masa kuliahnya. Selepas lulus - bersama keluarga - mengembangkan bisnis perangkat lunak melalui IT Division CV Reksadana AB (RAB). Tema proyek yang digarapnya antara lain: aplikasi database untuk perkantoran, web programming, intranet dan internet server.

Penulis juga telah menulis beberapa program bernuansa *open source*. Pada tahun 2000 penulis dan keluarga mendirikan PT RAB Linux Indonesia (nama yang terdaftar pada Departemen Kehakiman) yang tetap bergerak di bidang teknologi informasi untuk platform Linux. Saat ini ia menduduki jabatan direktur di perusahaan barunya ini. Di samping itu penulis aktif sebagai pengurus di KPLI Jakarta dan kini menduduki jabatan di Bidang Pengembangan Bisnis dan Kemitraan, serta ikut aktif dalam mengisi berbagai seminar yang mengatasmamakan lembaga nirlaba ini.



I Made Wiryana SSi SKom MSc menamatkan S1 di jurusan Fisika FMIPA Universitas Indonesia pada bidang instrumentasi dan fisika terapan. Dengan beasiswa dari STMIK Gunadarma juga menamatkan S1 Teknik Informatika di STMIK Gunadarma. Melanjutkan studi S2 di Computer Science Department Edith Cowan University - Perh dengan beasiswa ADCSS dan STMIK Gunadarma pada bidang fuzzy system dan artificial neural network untuk pengolahan suara. Menangani perancangan dan implementasi jaringan Internet di Universitas Gunadarma dengan memanfaatkan sistem Open Source sebagai basisnya. Pernah mewakili IPKIN dalam kelompok kerja Standardisasi Profesi TI untuk Asia Pasifik (SEARCC). Saat ini dengan beasiswa dari DAAD melanjutkan studi doktoral

di RVS Arbeitsgruppe Universität Bielefeld Jerman di bawah bimbingan Prof. Peter B Ladkin PhD. Aktif menjadi koordinator beberapa proyek penterjemahan program Open Source seperti KDE, SuSE, Abiword dan juga sebagai advisor pada Trustix Merdeka, distribusi Linux Indonesia yang pertama. Terkadang menyumbangkan tulisannya sebagai kolumnis pada media online DETIK.COM dan SATUNET. Juga kontributor pada KOMPUTEK, MIKRODATA, ELEKTRO dan INFOLINUX. Kontribusi ke komunitas Open Source sering dilakukan bersama-sama kelompok Tim PANDU. Star pengajar tetap Universitas Gunadarma.

Pernyataan

Dokumentasi ini dibuat pada saat penulis tengah menyelesaikan sejumlah proyek yang menggunakan Postgres. Tujuan awalnya hanya sekedar alat pengingat kala menemukan suatu kasus yang sama sehingga tidak perlu repot membuka “dokumentasi besar” yang disertakan dalam setiap paket Postgres. Lompatan yang cukup besar terjadi pada pengembangan sistem informasi di RS Pertamina Jaya⁵. Sistem besar ini tidak bisa lagi mencukupkan diri pada “SQL umum”. Banyak *feature* unik di Postgres yang dapat mempercepat proses penyelesaian masalah pemrograman yang dihadapi.

Penulisan buku ini disponsori oleh :

PT RAB Linux Indonesia

Jl. At Taufik IV/43 RT 7 RW 17

Kemanggisan Pulo

Jakarta 11480

Indonesia

Telp: 021-923-0770

Fax: 021-5367-2465

URL : <http://www.rab.co.id>

Email : info@rab.co.id

Proses pengeditan, pemformatan dan tata letak dilakukan oleh editor secara tidak langsung disponsori oleh :

• **Deutscher Akademischer Austauschdienst (DAAD)**

Kennedyalle 50

D-53175 Bonn - Jerman.

URL : <http://www.daad.de>

• **Universitas Gunadarma**

Jl Margonda Raya No 100. Depok - Jakarta

URL : <http://www.gunadarma.ac.id>

Beberapa merk dagang yang disebutkan pada buku ini merupakan merk dagang terdaftar dari perusahaan tersebut, kecuali bila disebutkan lain.

⁵<http://www.rspj.co.id>

Bab 1

Pendahuluan

PostgreSQL atau sering disebut Postgres merupakan salah satu dari sejumlah database besar yang menawarkan skalabilitas, keluwesan, dan kinerja yang tinggi. Penggunaannya begitu meluas di berbagai platform dan didukung oleh banyak bahasa pemrograman. Bagi masyarakat TI (teknologi informasi) di Indonesia, Postgres sudah digunakan untuk berbagai aplikasi seperti web, *billing system*, dan sistem informasi besar lainnya.

Ada banyak hal unik yang bisa kita temui dari database yang satu ini. Niatan awal para *programmer*-nya adalah membuat suatu database yang kaya akan *feature* dengan keluwesan yang tinggi. Prioritas ini sempat membuat Postgres dianggap sebagai database SQL yang tidak sesuai dengan standar ANSI-SQL92 sebagaimana yang lazim ditemui pada database berskala besar. Namun kini - secara perlahan tapi pasti - Postgres telah menjawab tantangan tersebut. ANSI-SQL92 memang sebuah standar, dan Postgres menawarkan standar yang lebih baik.

Dibalik masalah teknis tersebut, Postgres tersedia dalam bentuk *source code* dan dapat di-*download* tanpa pembebanan biaya. Tidak heran kalau Linux Award sempat menobatkan Postgres sebagai database pilihan yang diikuti Oracle sebagai *runner-up*-nya.

SQL di Postgres tidaklah seperti yang kita temui pada RDBMS umumnya. Perbedaan penting antara Postgres dengan sistem relasional standar adalah arsitektur Postgres yang memungkinkan user untuk mendefinisikan sendiri SQL-nya, terutama pada pembuatan *function* atau biasa disebut sebagai *stored procedure*. Hal ini dimungkinkan karena informasi yang disimpan oleh Postgres bukan hanya tabel dan kolom, melainkan tipe, fungsi, metode akses, dan banyak lagi yang terkait dengan tabel dan kolom tersebut. Semuanya terhimpun dalam bentuk *class* yang bisa **diubah** user. Arsitektur yang menggunakan *class* ini lazim disebut sebagai *object oriented*. Karena Postgres bekerja dengan *class*¹, berarti Postgres lebih mudah dikembangkan di tingkat user, dan Anda bisa mendefinisikan sebuah tabel sebagai turunan dari tabel lain.

Sebagai perbandingan bahwa sistem database konvensional hanya dapat diperluas dengan mengubah *source code*-nya, atau menggunakan modul tambahan yang ditulis khusus oleh vendor, maka dengan Postgres memungkinkan user untuk membuat sendiri *object file* atau *shared library* yang dapat diterapkan untuk mendefinisikan tipe data, fungsi, bahkan bahasa yang baru.

Dengan demikian Postgres memiliki dua kekuatan besar: *source code* dan arsitektur yang luwes, tentunya di samping *feature* penting lainnya seperti dokumentasi yang lengkap, dsb. Disamping itu Postgres juga didukung oleh banyak antarmuka² ke berbagai bahasa pemrograman seperti C++, Java, Perl, PHP, Python, dan Tcl. ODBC dan JDBC juga tersedia yang membuat Postgres lebih terbuka dan dapat diterapkan secara meluas.

¹Bahkan Postgres menganggap tabel sebagai suatu class

²interface

Bab 2

Instalasi

Kita akan membahas dua cara instalasi: menggunakan paket yang sudah dikompilasi (biasanya dengan rpm¹) atau mengkompilasi *source-code*-nya.

2.1 Instalasi dengan RPM

Penulis menggunakan RedHat 6.1 yang menyediakan **Postgres 6.5.2** dan SuSE 6.3 dengan **Postgres 6.5.1**. Tulisan ini mencoba membahas keduanya yang memiliki perbedaan dalam hal inialisasi awal dan keragaman tools dimana SuSE 6.5.1 memiliki nilai lebih.

Berikut beberapa paket yang ada di RedHat 6.1:

- postgresql-6.5.2-1.i386.rpm
- postgresql-python-6.5.2-1.i386.rpm
- postgresql-devel-6.5.2-1.i386.rpm
- postgresql-server-6.5.2-1.i386.rpm
- postgresql-tcl-6.5.2-1.i386.rpm

Biasanya awalan nama file tidak berubah untuk versi yang lainnya. Perbedaan hanya pada nomor versinya saja, seperti untuk versi 6.5.2 ditulis postgresql-6.5.2-1-i386.rpm.

Sedangkan pada SuSE 6.3 Anda dapat menginstall dengan menggunakan YaST, masuk menu Memilih/install paket / Mengubah/membuat konfigurasi / Program that don't need X. Selanjutnya pilih paket berikut :

- postgres
- pg_datab
- pg_ifa
- pg_iface
- pgaccess

Setelah itu jalankan Postgres server dengan cara (**RedHat**):

```
# /etc/rc.d/init.d/postgres start
```

Untuk SuSE 6. 3:

¹RedHat Package Manager, merupakan format yang kini banyak dipakai untuk kemudahan dalam instalasi suatu program.

```
# /etc/rc.d/init.d/postgresql start
```

Bila proses startup Postgres mengalami kegagalan, pastikan file `/tmp/.s.PGSQL.5432` dihapus terlebih dahulu, kemudian lakukan starting seperti di atas.

Untuk RedHat, jika postgres ingin dijalankan pada saat startup maka konfigurasi dengan :

```
# ntsysv
```

Di SuSE Anda tidak perlu melakukan hal serupa, karena secara otomatis ia akan membuat konfigurasi ini sehingga pada saat boot Postgres server langsung dijalankan.

2.2 Kompilasi dari Source

Salah satu cara untuk meng-*install* Postgres adalah dengan mengkompilasi *source code*-nya. Anda bisa mendapatkannya di <http://www.postgresql.org/>. Dengan mengkompilasi sendiri Anda akan lebih mengetahui apa yang Anda lakukan, disamping itu Anda bisa meng-upgrade dengan cepat karena Postgres yang diinstall diambil langsung dari sumbernya. Source yang tersedia sudah cukup lengkap, mulai dari dokumentasi, *library* PyGresql, dan *man pages*.

Tulisan ini akan membahas source Postgres versi 6.5.1.

Login sebagai root.²

```
# cd /usr/src
# mkdir pgsq
# chown postgres:postgres pgsq
# cd /usr/local
# mkdir pgsq
# chown postgres:postgres pgsq
```

Login dengan user postgres.

```
$ cd /usr/src/pgsq
$ tar xfv postgresql-6.5.1.tar.gz
$ mv postgresql-6.5.1/* .
$ rmdir postgresql-6.5.1
$ cd /usr/src/pgsq/src
$ ./configure
$ cd /usr/src/pgsq/doc
$ make install
$ cd /usr/src/pgsq/src
$ make all > make.log &
```

Untuk melihat proses kompilasi :

```
$ tail -f make.log
```

Selanjutnya install source yang telah dikompilasi, dan Anda harus sebagai root dalam hal ini.³

```
# cd /usr/src/pgsq/src
# make install > make.install.log &
```

Untuk melihat proses instalasi :

```
$ tail -f make.install.log
```

²Ok, kita buat kesepakatan : prompt # berarti root, sedangkan \$ berarti user biasa.

³Umumnya login sebagai root atau user biasa dibedakan dengan simbol prompt: '#' berarti root, dan '\$' berarti user biasa

Library yang terbentuk harus “diperkenalkan” ke seluruh user dengan menambahkan baris berikut ke file /etc/ld.so.conf.

```
/usr/local/pgsql/lib
```

Jalankan ldconfig agar perubahannya “terasa”.

```
# /sbin/ldconfig
```

Tambahkan baris ini ke file /etc/profile.

```
PATH=$PATH:/usr/local/pgsql/bin
LD_LIBRARY_PATH=/usr/local/pgsql/lib
MANPATH=$MANPATH:/usr/local/pgsql/man
PGLIB=/usr/local/pgsql/lib
PGDATA=/usr/local/pgsql/data
export PATH LD_LIBRARY_PATH MANPATH PGLIB PGDATA
```

Tahap berikutnya adalah membuat database sistem bernama template1. Loginlah sebagai user postgres.

```
$ initdb
```

Jalankan servernya.

```
$ nohup postmaster -i > pgserver.log 2>&1 &
```

Saatnya untuk mencoba.

```
$ psql -u template1
Username: postgres
Password: <ENTER>

template1=> SELECT NOW();
           now
-----
2000-09-08 11:34:50+07
(1 row)

template1=> \q
```

Konfigurasi untuk autostartup. Loginlah sebagai root.

```
# cp /usr/src/pgsql/contrib/linux/postgres.init.sh \
/etc/rc.d/init.d/postgres
# cd /etc/rc.d/rc5.d
# ln -s ../init.d/postgres S98postgres
```

Tambahkan option -i ke /etc/rc.d/init.d/postgres pada bagian PGOPTS agar bisa diakses *remotely*.

```
PGOPTS="-o -F -i -D/usr/local/pgsql/data"
```

Sekarang cobalah untuk mematikan Postgres dan menghidupkannya kembali dengan:

```
# /etc/rc.d/init.d/postgres stop
# /etc/rc.d/init.d/postgres start
```

Dengan berbagai kemungkinan dan alasan, bisa jadi server Postgres down (entah di-kill, crash, dsb). Untuk memastikan Postgres selalu *running*, tambahkan baris berikut ke file /etc/inittab dalam SATU BARIS.

```
pg:2345:respawn:/bin/su - postgres -c \  
"/usr/local/pgsql/bin/postmaster \  
-i -D/usr/local/pgsql/data \  
>> /usr/local/pgsql/server.log 2>&1 \  
</dev/null"
```

Menurut pembuatnya baris di atas dapat membuat Postgres hidup kembali manakala dia *down*. Namun demikian pembuatnya tidak tahu-menahu kalau script di atas menimbulkan efek samping lainnya. Jangan lupa menjalankan `init q` agar perubahannya terasa.

```
# init q
```

Sebelumnya Anda telah melakukan uji sederhana: terkoneksi ke database dan melakukan SELECT. Dalam source Postgres telah tersedia juga program test secara menyeluruh. Loginlah sebagai user `postgres`.

```
$ cd /usr/src/pgsql/src/test/regress  
$ make all runtest
```

Pesan kegagalan yang Anda peroleh pada saat test tipe data, bukan melulu ada masalah dengan Postgres-nya. Sebagai contoh untuk platform yang menggunakan prosesor Intel semacam Pentium II tidak akan diperoleh pesan kegagalan ini, karena sejak Postgres versi 6.5 program regress ini juga melakukan uji platform.

Kasus ini bisa terjadi pada uji tipe data `int8` (integer 8 byte). Anda akan mendapatkan pesan kegagalan manakala prosesor dan C compiler-nya tidak mendukung integer 64-bit (8 byte). Atau bisa jadi keduanya (prosesor dan C compiler) mampu, namun tidak dikonfigurasi untuk itu. Hal ini tidaklah perlu dirisaukan, kecuali Anda berniat untuk menggunakan tipe data `int8`.

Setelah test selesai dilakukan hapuslah database regression dan file lainnya yang sudah tidak digunakan.

```
$ destroydb regression  
$ cd /usr/src/pgsql/src/test/regress  
$ make clean
```

Bab 3

Mengenal Database

Database merupakan kumpulan dari seluruh objek database seperti tabel, view, trigger, fungsi, dan lain-lain. Postgres menyimpan suatu database dalam sebuah direktori, dan sebelum Anda membuatnya, pastikan Anda memperoleh hak untuk itu.

3.1 Konektivitas

Untuk menjalin konektivitas antara program client dengan server Postgres dibutuhkan beberapa informasi:

- Username
- Password
- Nama database
- Nama server (default: localhost), bisa juga berupa alamat IP
- Nomor port (default: 5432)

Postgres menyertakan program client yang sederhana, namanya `psql`. Pembuatan file database, atau memanipulasi tabel, semuanya bisa dilakukan di sini. Setiap perintah SQL harus diakhiri titik koma (;). Jika Anda terlanjur menekan <ENTER> sebelum mengakhirinya dengan titik koma, maka `psql` menganggap bahwa Anda belum selesai menuliskan perintah. Namun untuk perintah non-sql yang biasanya diawali back-slash (\) tidak perlu diakhiri titik koma. Untuk menyudahi `psql` ketikkan \q.

Sesaat setelah Postgres di-install maka didalamnya sudah diperoleh username postgres dan database `template1`. Anda bisa menggunakan keduanya untuk uji konektivitas secara lokal.

```
$ psql -u template1
Username: postgres
Password:
```

`template1` adalah *database system* dan merupakan *template* bagi seluruh database yang baru dibuat. Setelah itu Anda akan mendapat salam sambutan dan sebuah prompt.

```
Welcome to the POSTGRESQL interactive sql monitor:
Please read the file COPYRIGHT for copyright terms of POSTGRESQL
```

```
type \? for help on slash commands
type \q to quit
type \g or terminate with semicolon to execute query
You are currently connected to the database: template1
```

```
template1=>
```

Untuk menguji apakah server Postgres dapat diakses dari *host* lain, gunakan option `-h` disertai nama server atau IP address-nya. Misalkan IP address server 192.168.1.1, dan `psql` dijalankan di komputer lain ber-IP 192.168.1.2:

```
$ psql -h 192.168.1.1 -u template1
```

Secara default Postgres tidak mengizinkan akses dari *remote host*. Oleh karena itu Anda perlu melakukan perubahan pada file `pg_hba.conf`. Lengkapnya lihat Bab Pojok Admin.

3.2 Membuat Database

Setelah berhasil terkoneksi ke database sistem (`template1`) mulailah untuk membuat database. Anda memang dapat menggunakan `template1` sebagai ajang latihan, tapi hal tersebut tidak dianjurkan, karena `template1` merupakan *template* bagi database baru. Setiap objek (tabel, view, function, dan sebagainya) yang terdapat pada `template1` akan di-*copy* ke database baru.

```
template1=> CREATE DATABASE rab;
```

`psql` menyiapkan perintah `\c` untuk berpindah ke database lain.

```
template1=> \c rab
```

Atau jika dimulai dari console Linux:

```
$ psql -u rab
```

Setelah itu Anda akan mendapat salam sambutan dan sebuah prompt.

```
Welcome to the POSTGRESQL interactive sql monitor:
Please read the file COPYRIGHT for copyright terms of POSTGRESQL

type \? for help on slash commands
type \q to quit
type \g or terminate with semicolon to execute query
You are currently connected to the database: rab

rab=>
```

Selanjutnya penulis tidak menuliskan lagi prompt `rab=>` ini. Jika prompt diawali `#` atau `$` berarti merupakan bash prompt.

3.3 Tabel

Tabel merupakan wadah dimana data tersimpan. Setiap tabel memiliki field / kolom dan record / baris. Ada beberapa ketentuan dalam pembuatan tabel :

1. Harus memiliki field atau sekelompok field yang menyebabkan setiap record dalam tabel tersebut unik alias tidak ada yang sama, hal ini biasa disebut dengan primary key
2. Primary key tidak boleh null (hampa), jadi harus dideklarasikan sebagai not null. Jika Anda tidak menyebutkan `NULL / NOT NULL` maka Postgres secara default menganggapnya sebagai *nullable* (boleh kosong).

Pembuatan tabel terkait erat dengan sistem yang akan dibuat. Kita dapat memulai dengan data kepegawaian. Struktur tabel Pegawai terdiri dari ID bertipe integer dan merupakan primary key serta NAMA yang bertipe 30 karakter.

```
CREATE TABLE pegawai (  
  id INTEGER NOT NULL,  
  nama VARCHAR(30),  
  PRIMARY KEY (id)  
);
```

Untuk melihat tabel yang telah dibuat:

```
\dt
```

Jika Anda ingin melihat definisi dari tabel pegawai:

```
\d pegawai
```

Untuk menghapusnya:

```
DROP TABLE pegawai;
```

3.3.1 Mengisi Tabel dengan Record

Pengisian tabel (menambah record) menggunakan perintah `INSERT`¹ dan field yang diisi juga tidak harus semuanya. Yang perlu diingat adalah:

- Jika suatu field tidak diisi, secara otomatis Postgres akan mengisinya dengan `NULL`²
- Suatu field yang didefinisikan sebagai `NOT NULL`, maka harus diisi. Kalau tidak, Postgres akan menampilkan pesan kesalahan, dan pengisian dibatalkan.

Misalkan tabel Pegawai akan diisi dengan nilai sebagai berikut :

Field ID berisi 1000

Field NAMA berisi "BAGUS KAMSENO"

maka perintahnya :

```
INSERT INTO pegawai (id, nama)  
VALUES (1000, 'Agus Kamseno');
```

Gunakanlah kutip tunggal untuk string.

3.3.2 Melihat Isi Tabel

Melihat isi tabel dapat menggunakan statement `SELECT`.

```
SELECT * FROM pegawai;
```

Penggunaan karakter `*` menunjukkan bahwa yang ditampilkan adalah seluruh isi field yang ada pada tabel Pegawai.

ID	NAMA
1000	Agus Kamseno

Jika hanya field NAMA saja yang ditampilkan :

```
SELECT nama FROM pegawai;
```

¹ `INSERT` merupakan salah satu dari apa yang disebut dengan **Data Manipulation Language**

² `NULL` hampa, bukan string kosong, apalagi spasi. `NULL` merupakan salah satu kata yang dicadangkan (*reserved word*).

Maka hasilnya :

NAMA
Agus Kamseno

Coba Anda masukkan beberapa record sehingga setelah di-SELECT hasilnya seperti di bawah ini :

ID	NAMA
1000	Agus Kamseno
1001	Indah Kusumadewi
1002	Budi Hajadi
1003	Nirwanawati

3.3.3 Mengubah Record

Record yang ada dalam suatu tabel dapat kita ubah dengan perintah UPDATE. Sebagai contoh, akan kita ubah seluruh nilai field NAMA pada tabel Pegawai dengan huruf besar.

```
UPDATE pegawai
SET nama = UPPER(nama);
```

3.3.4 Menghapus Record

Kita dapat menggunakan perintah DELETE untuk menghapus record, seperti contoh (berbahaya) di bawah ini:

```
DELETE FROM pegawai;
```

Hati-hatilah ! Contoh di atas dapat menghapus seluruh record pada tabel Pegawai.³

Sebagaimana SELECT dan UPDATE, DELETE juga bisa disertakan dengan WHERE. Misalkan akan dihapus record Pegawai yang bernomor id 1003.

```
DELETE FROM pegawai
WHERE id = 1003;
```

3.3.5 Memanipulasi Struktur Tabel

Meski tabel telah dibuat, bukan berarti kita tidak bisa menghapus, menambah, atau mengubah tipe field-fieldnya. Mengubah struktur tabel tidak perlu menghapus tabelnya terlebih dahulu, karena hal itu dapat menghilangkan data, tentu saja. Kita dapat menggunakan kata kunci ALTER TABLE untuk masalah ini. Katakanlah kita akan menambah field TGL_LAHIR yang bertipe DATE pada tabel Pegawai.

```
ALTER TABLE pegawai
ADD tgl_lahir DATE;4
```

Nah, sekarang kita bisa mengisi nilai tanggal ke dalam field TGL_LAHIR. Yang perlu Anda ketahui dalam pengisian field bertipe tanggal adalah formatnya yang berupa mm/dd/yyyy dimana mm adalah bulan, dd tanggal, dan yyyy tahun. Penulisannya juga harus diapit tanda kutip, bisa kutip tunggal maupun ganda. Contoh :

```
INSERT INTO pegawai (id, nama, tgl_lahir)
VALUES (1003, 'RIYANA', '10/19/1977');5
```

³Sebenarnya SQL memungkinkan kita untuk "mencoba dulu" akibat dari perintah yang di-execute. Penggunaan sintaks BEGIN TRANSACTION, ROLLBACK, dan COMMIT adalah merupakan standar SQL untuk proses yang menggunakan DML (Data Manipulation Language). Tema ini akan dibahas tersendiri.

⁴Pada saat pembuatan tulisan ini penambahan field tidak akan berpengaruh jika kita langsung memanfaatkan field baru tersebut. Lakukan reconnect dahulu supaya perubahannya berpengaruh : \connect pegawai

⁵Format mm/dd/yyyy untuk tanggal merupakan standar SQL. Namun Postgres mendapat kemampuan lebih dari programmer-nya, yaitu dapat menganalisa separator (boleh menggunakan '-') atau bisa mengenali bagian-tanggal untuk '19/10/1977' (19 dapat dikenali sebagai tanggal karena lebih besar dari 12).

Untuk Postgres versi 6.5.2 Anda akan mendapatkan pesan dibawah ini setelah menambah field baru dan melakukan pengisian data ke field tersebut.

```
ERROR: Relation 'pegawai' does not have attribute 'tgl_lahir'
```

yang perlu Anda lakukan adalah reconnect database :

```
\c rab
```

Ada pembuatan tentu ada penghapusan. Untuk menghapus suatu field dapat kita gunakan perintah DROP, seperti contoh berikut ini yang menghapus field TGL_LAHIR.

```
ALTER TABLE pegawai
DROP tgl_lahir;6
```

Sayang sekali bahwa versi Postgres yang penulis gunakan belum memungkinkan hal tersebut, sehingga muncul pesan di bawah ini.⁷

```
ERROR: ALTER TABLE/DROP COLUMN not yet implemented
```

Oh, ya, dari latihan di atas field TGL_LAHIR yang terisi hanya untuk ID Pegawai 1003, sehingga ketika Anda SELECT terhadap tabel Pegawai akan tampak seperti ini :

ID	NAMA	TGL_LAHIR
1000	AGUS KAMSENO	
1001	INDAH KUSUMADEWI	
1002	BUDI HAJADI	
1003	RIYANA	10-19-1977

Anda bisa mengubah isi field TGL_LAHIR untuk setiap pegawai. Perintah UPDATE yang sudah diperkenalkan sebelumnya belum mencukupi karena contoh UPDATE tersebut akan mengubah seluruh nilai field, padahal setiap orang tanggal lahirnya berbeda-beda, sehingga disini kita membutuhkan kondisi dengan menggunakan WHERE.

```
UPDATE pegawai
SET tgl_lahir = '8/8/1973'
WHERE id = 1000;
```

Sehingga hasil di atas seperti ini ketika di-SELECT :

ID	NAMA	TGL_LAHIR
1000	AGUS KAMSENO	8-8-1973
1001	INDAH KUSUMADEWI	
1002	BUDI HAJADI	
1003	RIYANA	10-19-1977

Nah, sekarang tinggal Anda melanjutkan dua pegawai lainnya yang TGL_LAHIR-nya masih NULL. Katakanlah menjadi seperti ini :

ID	NAMA	TGL_LAHIR
1000	AGUS KAMSENO	8-8-1973
1001	INDAH KUSUMADEWI	12-1-1974
1002	BUDI HAJADI	4-5-1975
1003	RIYANA	10-19-1977

⁶Awas, penghapusan suatu field juga berarti penghapusan data dalam field tersebut.

⁷Kekurangan tersebut tidaklah terlalu signifikan, karena hal itu bisa dilakukan dengan cara lain. Sebenarnya pada versi 6.5.2 ini terdapat kekurangan lain yang seharusnya telah menjadi standar SQL seperti ketiadaan foreign key, dsb. Prioritas utama para programmer Postgres adalah kelengkapan feature. Namun demikian mereka tetap mengagendakan feature-feature standar SQL untuk versi berikutnya. Sebagai tambahan, foreign key sudah disertakan pada versi 7.0.

3.3.6 Hubungan Antar Tabel (Relational)

Mari kita mulai dari contoh kasus. Sistem kepegawaian ini akan dilengkapi dengan data anak setiap pegawai, dan karena setiap pegawai bisa memiliki anak lebih dari satu, maka kita perlu membuat sebuah tabel tersendiri (kita namakan tabel Anak) yang berhubungan dengan tabel Pegawai. Pertanyaan selanjutnya, apa yang menjadi penghubung antara dua buah tabel? Jawabannya adalah field, namun field yang mana?

Yang jelas untuk menghubungkan dua buah tabel, keduanya harus memiliki satu atau beberapa field yang sama isinya. Jadi setidaknya salah satu tabel harus memiliki field yang merupakan kunci (primary key) dari tabel lain. Untuk kasus ini berarti tabel Anak harus mengandung field ID Pegawai yang sebenarnya merupakan primary key dari tabel Pegawai. Nah field ID Pegawai yang ada pada tabel Anak ini biasa disebut dengan **foreign key**.⁸

Tapi jangan lupa untuk memberi primary key pada tabel Anak ini.⁹ Primary key-nya tentu bukan hanya field ID Pegawai lagi, namun harus ditambah dengan field nama anak. Mengapa dipilih nama anak? Karena kita memang berasumsi (dan ini telah sangat umum dalam kehidupan manusia) bahwa dalam satu keluarga nama anak tidak mungkin sama, sehingga kita bisa menjadikan nama anak ini sebagai "anggota" primary key.

```
CREATE TABLE anak (
  id_pegawai INTEGER NOT NULL
    REFERENCES pegawai,
  nama VARCHAR(50) NOT NULL,
  tgl_lahir DATE,
  PRIMARY KEY (id_pegawai, nama)
);
```

Kata REFERENCES pegawai untuk field id_pegawai memastikan bahwa nilai field tersebut dipastikan terdapat pada field *primary key* milik tabel pegawai. Sekarang kita coba untuk mengisinya:

```
INSERT INTO anak (id_pegawai, nama, tgl_lahir)
VALUES (1000, 'FERAWATI HANSIN', '7/23/1998');
```

Coba Anda isikan beberapa data anak di tabel tersebut sehingga hasilnya seperti ini ketika di-SELECT:

ID_PEGAWAI	NAMA	TGL_LAHIR
1000	FERAWATI HANSIN	7-23-1998
1001	ANDINI SUCIATI	7-4-1997
1001	MUHAMMAD NAJIB	3-1-1999

Kalau field id_pegawai pada tabel anak diisi dengan nilai yang tidak terdapat pada *primary key* tabel pegawai:

```
INSERT INTO anak (id_pegawai, nama, tgl_lahir)
VALUES (9999, 'YANTI', '4/5/1996');
```

maka akan tampil pesan kesalahan berikut:

```
ERROR: <unnamed> referential integrity violation - key
referenced from anak not found in pegawai
```

Pesan kesalahan juga akan muncul jika record pegawai dihapus (DELETE), dimana nilai primary key-nya sedang digunakan oleh tabel anak. Untuk kasus ini, Postgres telah menyiapkan *option* pelengkap REFERENCES dimana ketika record pegawai dihapus maka record terkait pada tabel anak akan dihapus juga.

```
CREATE TABLE anak (
  id_pegawai INTEGER NOT NULL
    REFERENCES pegawai ON DELETE CASCADE,
```

⁸Foreign key belum diterapkan dalam Postgres 6.5.x, sedangkan versi 7 sudah.

⁹Lihat ketentuan pembuatan tabel pada pembahasan sebelumnya

```

nama VARCHAR(50) NOT NULL,
tgl_lahir DATE,
PRIMARY KEY (id_pegawai, nama)
);

```

Bahkan perubahan id pada tabel pegawai juga dapat mengubah nilai id_pegawai pada tabel anak:

```

CREATE TABLE anak (
id_pegawai INTEGER NOT NULL
REFERENCES pegawai
ON DELETE CASCADE,
ON UPDATE CASCADE,
nama VARCHAR(50) NOT NULL,
tgl_lahir DATE,
PRIMARY KEY (id_pegawai, nama)
);

```

3.4 Membuat Laporan

Yang dimaksud dengan laporan disini adalah kita membuat output yang datanya berasal dari tabel-tabel yang kita miliki (dalam hal ini tabel yang sudah kita buat adalah Pegawai dan Anak). Dari dua buah tabel tersebut sudah bisa kita buat beberapa jenis laporan :

1. Daftar pegawai, dan ini sudah pernah kita buat pada contoh sebelumnya
2. Daftar pegawai yang lahir di tanggal tertentu
3. Daftar anak beserta nama pegawai yang menjadi orang tuanya (tentu saja).
4. Daftar pegawai beserta anak-anaknya, baik pegawai yang sudah punya anak maupun belum.
5. Daftar pegawai yang belum memiliki anak.
6. Daftar pegawai yang anaknya lebih dari satu.

Jadi pada dasarnya laporan yang dihasilkan bisa banyak sekali kombinasi / kemungkinannya, tergantung permintaan pihak manajemen biasanya. Nah, mari kita urai satu-persatu.

3.4.1 Daftar Pegawai

Ini sudah pernah kita lakukan dengan perintah SELECT :

```
SELECT id, nama, tgl_lahir FROM pegawai;
```

ID	NAMA	TGL_LAHIR
1000	AGUS KAMSENO	8-8-1973
1001	INDAH KUSUMADEWI	12-1-1974
1002	BUDI HAJADI	4-5-1975
1003	RIYANA	10-19-1977

Anda bisa menggunakan perintah ORDER BY untuk mengurutkan data berdasarkan field tertentu:

```
SELECT id, nama, tgl_lahir FROM pegawai ORDER BY nama;
```

atau dengan menyebutkan nomor kolom yang akan memberikan hasil yang sama:

```
SELECT id, nama, tgl_lahir FROM pegawai ORDER BY 2;
```

ID	NAMA	TGL_LAHIR
1000	AGUS KAMSENO	8-8-1973
1002	BUDI HAJADI	4-5-1975
1001	INDAH KUSUMADEWI	12-1-1974
1003	RIYANA	10-19-1977

Daftar Pegawai yang Lahir di Tanggal Tertentu

```
SELECT id, nama, tgl_lahir
FROM pegawai
WHERE tgl_lahir = '10/19/1977';
```

Beberapa operator logika yang umum seperti kurang dari (<), lebihdari (>), tidak sama dengan (!=), kurang dari atau sama dengan (<=), lebihdari atau sama dengan (>=) juga bisa dipakai.

```
SELECT id, nama, tgl_lahir
FROM pegawai
WHERE tgl_lahir > '1/1/1977';
```

Beberapa laporan bisa saja terdiri dari lebih dari satu ekspresi logika. Misalnya kita diminta membuat laporan daftar pegawai yang lahir di tahun 1974. Secara logis, dengan menggunakan sintaks yang telah kita pelajari sebelumnya, hal itu bisa diperoleh dengan kondisi : TGL LAHIR lebih-dari-atau-sama-dengan 1 Januari 1974 dan kurang-dari-atau-sama-dengan 31 Desember 1974.

```
SELECT id, nama, tgl_lahir
FROM pegawai
WHERE tgl_lahir >= '1/1/1974' AND tgl_lahir <= '12/31/1974';
```

Atau kita bisa memanfaatkan sintaks BETWEEN untuk menyederhanakan penulisan namun dengan hasil yang sama.

```
SELECT id, nama, tgl_lahir
FROM pegawai
WHERE tgl_lahir BETWEEN '1/1/1974' AND '12/31/1974';
```

Atau bisa juga menggunakan internal function date_part untuk mengambil nilai tahun dari field tgl_lahir.

```
select * from pegawai where
date_part('year', tgl_lahir) = 1977;
```

3.4.2 Daftar Anak Beserta Nama Pegawai

Kita akan menggabungkan dua buah tabel yang saling terkait dimana tabel Anak memiliki foreign key (baca: field penghubung) ke tabel Pegawai. Sehingga secara logis foreign key tabel Anak (ID_PEGAWAI) akan dihubungkan dengan primary key tabel Pegawai (ID).

```
SELECT pegawai.id, pegawai.nama, anak.nama
FROM pegawai, anak
WHERE pegawai.id = anak.id_pegawai;
```

ID	NAMA	NAMA
1000	AGUS KAMSENO	FERAWATI
1001	INDAH KUSUMADEWI	ANDINI SUCIATI
1001	INDAH KUSUMADEWI	MUHAMMAD NAJIB

Penyebutan nama tabel beserta titik sebelum nama field dimaksudkan untuk memastikan bahwa field yang ditampilkan diambil dari tabel yang diinginkan, karena bisa terjadi dua buah tabel memiliki nama field yang sama.

Penulisan perintah di atas sebenarnya masih bisa dipersingkat dengan membuat alias untuk setiap tabel:

```
SELECT p.id, p.nama, p.tgl_lahir, a.nama
FROM pegawai p, anak a
WHERE p.id = a.id_pegawai;
```

3.4.3 Daftar Pegawai yang Anaknya Lebih Dari Satu

Langkah pertama adalah kita buat daftar jumlah anak setiap pegawai. Ini bisa kita buat dengan memanfaatkan **agregate function**¹⁰ COUNT yang harus dikombinasikan dengan GROUP BY.

```
SELECT p.id, p.nama, COUNT(*)
FROM pegawai p, anak a
WHERE p.id = a.id_pegawai
GROUP BY p.id, p.nama;
```

ID	NAMA	COUNT
1000	AGUS KAMSENO	1
1001	INDAH KUSUMADEWI	2

Karena COUNT() adalah *agregate function* maka mengkondisikannya bukan dengan WHERE melainkan dengan HAVING.

```
SELECT p.id, p.nama, COUNT(*)
FROM pegawai p, anak a
WHERE p.id = a.id_pegawai
GROUP BY p.id, p.nama
HAVING COUNT(*) > 1;
```

3.5 View

View dapat digunakan untuk menyimpan perintah query:

```
CREATE VIEW v_anak AS
SELECT p.id, p.nama, p.tgl_lahir, a.nama AS nama_anak
FROM pegawai p, anak a
WHERE p.id = a.id_pegawai;
```

View dapat dianggap sebagai tabel sehingga dapat di-SELECT layaknya tabel:

```
SELECT * FROM v_anak;
```

Bahkan di Postgres view dapat di-INSERT, UPDATE, atau DELETE, yaitu dengan menerapkan *rule*. Lebih jelasnya lihat Bab Table Rule.

Anda perlu hati-hati dalam menstrukturisasi tabel yang digunakan view. Untuk contoh di atas view v_anak menggunakan tabel pegawai dan tabel anak. Bila suatu waktu Anda berniat menghapus dan membuat ulang tabel anak, maka pastikan - sebelum melakukannya - terlebih dahulu mem-backup view v_anak dengan menggunakan pg_dump:¹¹

```
$ pg_dump -t v_anak -f v_anak.sql -u rab
```

Contoh berikut merupakan kesalahan fatal yang dapat terjadi manakala tabel anak dihapus dan dibuat kembali - meski dengan nama dan struktur yang sama - namun view v_anak kehilangan "alamat memori":

¹⁰ **Agregate function** boleh juga dikatakan function yang "membuat kesimpulan" terhadap sekumpulan field dalam suatu perintah SELECT.

¹¹ Lihat Sub Bab Backup

```

DROP TABLE anak;

CREATE TABLE anak (
  id_pegawai INTEGER NOT NULL
    REFERENCES pegawai,
  nama VARCHAR(50) NOT NULL,
  tgl_lahir DATE,
  PRIMARY KEY (id_pegawai, nama);

SELECT * FROM v_anak;
ERROR:  relation_info: Relation 17833983 not found

```

3.6 Temporary Table

Temporary table adalah tabel dengan masa hidup singkat. Keberadaannya hanya ada dalam sebuah *database session*.¹² Bahkan dua *database session* dapat membuat *temporary table* dengan nama yang sama. Ketika *database session* ini berakhir maka *temporary table* secara otomatis terhapus. Tabel berikut mengilustrasikan hal tersebut:

User1	User2
CREATE TEMP TABLE test (id INT)	CREATE TEMP TABLE test(id INT)
INSERT INTO test VALUES(1)	INSERT INTO test VALUES(2)
SELECT id FROM test -> 1	SELECT id FROM test -> 2

Temporary table cocok untuk query yang kompleks dimana beberapa hasil query “dikumpulkan” dalam *temporary table* ini. Contoh menarik dapat ditemukan pada Bab Data dari Database Lain.

¹²*Database session*: sebuah konektivitas antara aplikasi client dengan Postgres. Dua program psql yang sedang terkoneksi ke Postgres dikatakan terjadi dua *database session*.

Bab 4

Function dan Trigger

Fungsi sertaan(*built-in*) dalam Postgres sangatlah banyak. Di `psql` kita dapat melihatnya dengan perintah:

```
\df
```

Contoh:

```
SELECT UPPER('halo');

 upper
-----
  HALO
(1 row)
```

Ada beberapa konsep menarik tentang fungsi ini:

- Bahasanya bisa didefinisikan sendiri dengan tersedianya parameter `LANGUAGE`, tanpa harus mengkompilasi ulang Postgres.
- Dua buah fungsi dapat memiliki nama yang sama namun parameter masukkan yang berbeda, baik tipe datanya dan/atau jumlahnya.

Selain memiliki *built-in function*, Postgres juga memiliki *built-in language* bernama `sql` bagi mereka yang ingin membuat fungsi sendiri.

```
CREATE FUNCTION tambah(INT, INT) RETURNS INT AS
'SELECTselect $1 + $2'
LANGUAGE 'sql';

SELECT tambah(5,6);

 tambah
-----
      11
(1 row)
```

Penjelasan poin kedua dari konsep di atas terlihat dalam contoh berikut:

```
CREATE FUNCTION tambah(INT, INT, INT) RETURNS INT AS
'SELECT $1 + $2 + $3'
LANGUAGE 'sql';

SELECT tambah(7,8,9);
```

```
tambah
-----
      24
(1 row)
```

Juga memungkinkan jumlah parameter yang sama namun tipe datanya berbeda:

```
CREATE FUNCTION tambah(FLOAT, FLOAT) RETURNS FLOAT AS
  'SELECT $1 + $2'
LANGUAGE 'sql';
```

```
SELECT tambah(8.9, 10.5);
```

```
tambah
-----
     19.4
(1 row)
```

Karena komposisi parameter suatu fungsi merupakan bagian dari “ID” fungsi itu sendiri, maka penghapusannya pun harus menyertakan parameter ini:

```
DROP FUNCTION tambah(FLOAT, FLOAT);
```

4.1 PL/pgSQL sebagai Procedural Language

Untuk fungsi yang lebih kompleks dapat menggunakan `plpgsql` sebagai language. *Language* ini terdapat dalam instalasi Postgres namun perlu didaftarkan dulu di setiap database yang akan menggunakannya.

Perintah berikut memberitahu suatu database dimana lokasi *shared object* untuk bahasa PL/pgSQL:

```
CREATE FUNCTION plpgsql_call_handler() RETURNS OPAQUE AS
  '/usr/local/pgsql/lib/plpgsql.so' LANGUAGE 'C';
```

Selanjutnya baris berikut mendefinisikan bahwa fungsi di atas akan digunakan manakala suatu fungsi menggunakan *language 'plpgsql'*.

```
CREATE TRUSTED PROCEDURAL LANGUAGE 'plpgsql' HANDLER
plpgsql_call_handler LANCOMPILER 'PL/pgSQL';
```

Anda harus sebagai *superuser* (`postgres`) untuk melakukan dua hal di atas.

Programmer PL/pgSQL (Jan Wieck) merancang bahasa ini dengan tujuan:

- untuk membuat *function* dan *trigger procedure*
- merupakan bahasa SQL terstruktur
- dapat mengolah algoritma yang rumit
- dapat mengenali semua tipe data, fungsi, dan operator yang kita definisikan sendiri (*user defined*)
- dapat digunakan oleh user lain (selain *superuser* postgres)
- mudah

4.1.1 Struktur PL/pgSQL

PL/pgSQL adalah bahasa dengan mekanisme blok dengan struktur berikut:

```
[<<label>>]
[DECLARE
  declaration]
BEGIN
  statements
END;
```

Dalam blok *statements* bisa terdiri dari beberapa subblok.

Comment (keterangan/dokumentasi) bisa berada dimana saja. Ada dua tipe *comment* : menggunakan *dash* '-' yang berarti memulai comment hingga akhir baris, atau menggunakan **/* yang menandakan *comment* dimulai hingga **/* ditemukan.

```
CREATE FUNCTION kali(FLOAT, FLOAT) RETURNS FLOAT AS '
BEGIN
  -- Fungsi perkalian dua bilangan
  /* Dibuat oleh:
     sugiana@rab.co.id
  */
  RETURN $1 * $2;
END;
' LANGUAGE 'plpgsql';

SELECT kali(3,4);

   kali
-----
      12
(1 row)
```

4.1.2 Function sebagai Trigger Procedures

PL/pgSQL dapat digunakan untuk *trigger procedures*. Ciri khas fungsi yang diperuntukkan untuk trigger adalah menghasilkan output bertipe OPAQUE. Fungsi untuk trigger ini memiliki beberapa variabel khusus yang terdeklarasi secara otomatis.

NEW

Bertipe RECORD, variabel yang berisi nilai-nilai baru suatu record pada saat INSERT atau UPDATE.

OLD

Bertipe RECORD, variabel yang berisi nilai-nilai lama suatu record pada saat UPDATE atau DELETE.

Berikut ini adalah contoh penggunaan fungsi sebagai *trigger procedure*. Trigger berikut memastikan isi field NAMA pada tabel PEGAWAI selalu huruf besar.

```
CREATE FUNCTION pegawai_ins_upd () RETURNS OPAQUE AS '
BEGIN
  NEW.nama := UPPER(NEW.nama);
  RETURN NEW;
END; '
LANGUAGE 'plpgsql';
```

Trigger sendiri digunakan untuk “menyisipkan” suatu fungsi pada saat suatu record di-INSERT, UPDATE, atau DELETE.

```
CREATE TRIGGER peg_ins_upd BEFORE INSERT OR UPDATE
ON pegawai FOR EACH ROW
EXECUTE PROCEDURE pegawai_ins_upd();
```

```
INSERT INTO pegawai (id,nama) VALUES (1012,'owo');
```

```
SELECT * FROM pegawai;
```

```
   id | nama
-----+-----
 1012 | OWO
(1 row)
```

Bab 5

Select

Bab berikut ingin mengulas secara lebih mendalam perintah SELECT.

5.1 Meng-COPY tabel

SELECT dapat digunakan untuk meng-COPY suatu tabel ke tabel lain dengan nama yang berbeda (tentunya).

```
SELECT * INTO pegawai_lama FROM pegawai;
```

Struktur dan isi tabel PEGAWAI_LAMA sama dengan tabel PEGAWAI. Atau bisa juga di-copy ke *temporary table*.

```
SELECT * INTO TEMP pegawai_lama FROM pegawai;
```

5.2 Sub SELECT sebagai Field

Suatu field dalam sebuah query dapat berupa sub-select. Ikuti contoh berikut:

```
CREATE TABLE jabatan (  
  id SMALLINT NOT NULL PRIMARY KEY,  
  nama VARCHAR(30) NOT NULL);  
  
INSERT INTO jabatan VALUES (1,'Komisaris');  
INSERT INTO jabatan VALUES (2,'Direktur');  
INSERT INTO jabatan VALUES (3,'Bendahara');  
INSERT INTO jabatan VALUES (4,'EDP');  
INSERT INTO jabatan VALUES (5,'HRD');
```

Melanjutkan struktur tabel PEGAWAI sebelumnya, sekarang kita tambahkan field ID_JABATAN yang mereferensi ke tabel JABATAN.

```
ALTER TABLE pegawai  
  ADD id_jabatan smallint  
  REFERENCES jabatan;
```

Kemudian tambahkan dua record pegawai dengan dan tanpa pengisian field ID_JABATAN.

```
INSERT INTO pegawai (id,nama)  
  VALUES (1010,'Hermawan');  
INSERT INTO pegawai (id,nama,id_jabatan)  
  VALUES (1011,'Yulianti',3);
```

Dari dua tabel di atas akan dibuat suatu query yang menampilkan ID pegawai, nama, dan nama jabatannya. Bagi yang belum memiliki jabatan tetap akan dimunculkan, namun dengan nilai NULL.

id	nama	jabatan
1010	Hermawan	
1011	Yulianti	Bendahara

```
SELECT
  a.id, a.nama,
  (SELECT b.nama
   FROM jabatan b
   WHERE b.id = a.id_jabatan) AS jabatan
FROM pegawai a;
```

```
 id |  nama  | jabatan
----+-----+-----
1010 | Hermawan |
1011 | Yulianti | Bendahara
(2 rows)
```

Bahasa SQL sebenarnya memungkinkan mendapatkan hasil di atas dengan perintah LEFT JOIN. Namun sampai versi 7.0.2 ini Postgres belum menyertakannya.

```
SELECT a.id, a.nama, b.nama AS jabatan FROM pegawai A
LEFT JOIN jabatan b ON b.id = a.id_jabatan;
```

```
ERROR: OUTER JOIN is not yet supported
```

5.3 Query = Himpunan

Hasil query sebenarnya merupakan suatu himpunan sebagaimana yang sering kita temui dalam pelajaran matematika (ingat diagram Venn). Buatlah dua buah tabel berikut:

```
CREATE TABLE tabell (id INT);
INSERT INTO tabell SELECT 1;
INSERT INTO tabell SELECT 2;
INSERT INTO tabell SELECT 3;
```

```
CREATE TABLE tabel2 (id INT);
INSERT INTO tabel2 SELECT 1;
INSERT INTO tabel2 SELECT 2;
```

Gabungkan keduanya dengan UNION:

```
SELECT * FROM tabell
UNION
SELECT * FROM tabel2;
```

```
 id
----
 1
 2
 3
(3 rows)
```

Hasil UNION dua tabel tersebut dipastikan tidak ada komposisi record yang sama.¹ Jika Anda mengharapkan seluruh record di TABEL1 dan TABEL2 tampak dalam hasil query gunakan UNION ALL.

```
SELECT * FROM tabel1
UNION ALL
SELECT * FROM tabel2;

 id
----
  1
  2
  3
  1
  2
(5 rows)
```

Irisan (INTERSECT) keduanya seperti ini:

```
SELECT * FROM tabel1
INTERSECT
SELECT * FROM tabel2;

 id
----
  1
  2
(2 rows)
```

Untuk mendapatkan anggota himpunan yang tidak terdapat dalam himpunan lain digunakan EXCEPT:

```
SELECT * FROM tabel1
EXCEPT
SELECT * FROM tabel2;

 id
----
  3
(1 row)
```

Jika sebaliknya maka tidak ada hasil query apapun, karena seluruh isi TABEL2 terdapat dalam TABEL1.

```
SELECT * FROM tabel2
EXCEPT
SELECT * FROM tabel1;

 id
----
(0 rows)
```

¹Meski di TABEL1 dan TABEL2 terdapat dua nilai yang sama (1 dan 2) namun hasil query hanya menyebutkan sekali.

Bab 6

Table Rule

Suatu tabel dapat diubah sifatnya dengan melekatkan *rule*. View dalam Postgres sebenarnya sebuah tabel yang telah diberi `SELECT RULE`.

```
CREATE VIEW v_peg AS SELECT * FROM pegawai;
```

Pada Linux console lakukan *dump*:

```
$ pg_dump -t v_peg -u rab
Username: postgres
Password:

\connect - postgres
CREATE TABLE "v_peg" (
    "id" int4,
    "nama" character varying(30),
    "id_jabatan" int2
);
CREATE RULE "_RETv_peg" AS ON SELECT TO v_peg DO INSTEAD
SELECT pegawai.id, pegawai.nama, pegawai.id_jabatan
FROM pegawai;
```

Proses *dump* di atas terlihat bahwa `v_peg` adalah tabel yang dilekatkan padanya `SELECT RULE` bernama `_RETv_peg`.

6.1 Insert Rule

Sekarang kita mencoba melekatkan `INSERT RULE` pada view `v_peg` dimana bila perintah `INSERT` dikenakan pada `v_peg` maka tabel `pegawai` yang di-`INSERT`.

```
CREATE RULE v_peg_ins AS ON INSERT TO v_peg DO
INSERT INTO pegawai (id,nama)
VALUES (NEW.id, NEW.nama);
```

Perhatikan `NEW.id` dan `NEW.nama` di atas. Keduanya mewakili nilai baru yang dimasukkan ke view `v_peg`. Contoh `INSERT` berikut memperjelas rule di atas:

```
INSERT INTO v_peg (id, nama) VALUES (1012, 'Wawan');
```

Perintah di atas memberikan nilai pada `NEW.id = 1012` dan `NEW.nama = 'Wawan'`. Untuk membuktikan `INSERT RULE` bekerja dengan baik, lakukan `SELECT` terhadap tabel `pegawai`.

```
SELECT * FROM pegawai;
```

id	nama	id_jabatan
1010	Hermawan	
1011	Yulianti	3
1012	Wawan	

(3 rows)

6.2 Update Rule

UPDATE RULE juga bisa diterapkan. Sedikit berbeda dengan INSERT RULE, UPDATE RULE selain memiliki NEW juga terdapat OLD, yaitu nilai lama sebelum UPDATE terjadi.

```
CREATE RULE v_peg_upd AS ON UPDATE TO v_peg DO
UPDATE pegawai
SET nama = NEW.nama
WHERE id = OLD.id;
```

Rule di atas hanya “menyempatkan” mengubah field nama untuk tabel pegawai.

```
UPDATE v_peg SET nama = 'Wawan Jati' WHERE id = 1012;
```

```
SELECT * FROM pegawai;
```

id	nama	id_jabatan
1010	Hermawan	
1011	Yulianti	3
1012	Wawan Jati	

(3 rows)

6.3 Delete Rule

DELETE RULE juga serupa, hanya saja ia tidak memiliki NEW.

```
CREATE RULE v_peg_del AS ON DELETE TO v_peg DO
DELETE FROM pegawai WHERE id = OLD.id;
```

```
DELETE FROM v_peg WHERE id = 1012;
```

```
SELECT * FROM pegawai;
```

id	nama	id_jabatan
1010	Hermawan	
1011	Yulianti	3

(2 rows)

6.4 Rule vs Trigger

Cara kerja Trigger dan Rule berbeda. Trigger memanggil suatu fungsi pada saat setiap record mengalami perubahan. Sedangkan rule mengubah query atau menambahkannya. Trigger cocok untuk memeriksa nilai dalam suatu record

sebelum record tersebut disimpan dalam tabel. Sedangkan rule lebih tepat digunakan untuk hal yang menyangkut tabel lain.

Melalui sebuah pengamatan rule ternyata lebih dahulu terjadi ketimbang trigger. Mari kita buat sebuah eksperimen. Terdapat dua tabel a dan b.

```
CREATE TABLE a (id INTEGER);
CREATE TABLE b (id INTEGER);

CREATE RULE a_ins AS ON INSERT TO a
DO INSERT INTO b (id) VALUES (NEW.id);

CREATE FUNCTION a_ins () RETURNS OPAQUE AS '
BEGIN
    INSERT INTO b (id) VALUES (NEW.id + 1);
    RETURN NEW;
END;'
LANGUAGE 'plpgsql';

CREATE TRIGGER a_ins BEFORE INSERT ON a
FOR EACH ROW EXECUTE PROCEDURE a_ins();

INSERT INTO a VALUES (1);

SELECT oid, * FROM b;
```

oid	id
17265688	1
17265689	2

(2 rows)

Bab 7

Transaksi

Postgres menyediakan modus transaksi (*transaction mode*) sebagaimana standar SQL92. *Transaction* dapat digunakan untuk:

- Melihat perubahan sementara (tidak permanen)
- Mengubah data dengan beberapa statement pada saat bersamaan

Transaction mode diawali BEGIN dan diakhiri dengan COMMIT (perubahan disetujui) dan ROLLBACK (perubahan dibatalkan).

```
CREATE TABLE test (id INT);

INSERT INTO test SELECT 1;

BEGIN;

DELETE FROM test;

SELECT * FROM test;
  id
----
(0 rows)

ROLLBACK;

SELECT * FROM test;

  id
----
  1
(1 row)
```

ROLLBACK juga terjadi (secara otomatis) manakala:

- terdapat statement yang gagal
- database session terputus

Contoh nyata bisa kita ambil dalam dunia perbankan. Terdapat sebuah tabel nasabah dengan definisi sebagai berikut:

```
CREATE TABLE nasabah (
  id INT NOT NULL PRIMARY KEY,
  nama VARCHAR(30) NOT NULL,
  saldo FLOAT NOT NULL);
```

Field saldo di atas nilainya selalu berubah, tergantung transaksi yang terjadi pada tabel transaksi berikut:

```
CREATE TABLE transaksi (  
    waktu DATETIME NOT NULL DEFAULT NOW(),  
    id_nasabah INT REFERENCES nasabah,  
    jumlah FLOAT NOT NULL);
```

Pertama kali seorang calon nasabah mendaftarkan diri, maka dia harus menyetorkan uang sebesar Rp 10.000,-.

```
INSERT INTO nasabah (id, nama, saldo)  
VALUES (1412, 'Fernandoz', 10000);
```

Keluar-masuknya uang tetap harus tercatat dalam tabel transaksi:

```
INSERT INTO transaksi (id_nasabah, jumlah)  
VALUES (1412, 10000);
```

Selama kedua perintah INSERT di atas berlangsung dengan baik maka nilai saldo tetap konsisten sesuai dengan akumulasi jumlah pada tabel transaksi. Namun manakala sesaat setelah tabel nasabah dimasukkan data (INSERT yang pertama) terjadi sebuah “kecelakaan” - misalnya sistem *down* - maka nilai saldo tidak dapat dipertanggungjawabkan bila sistem telah normal kembali, yaitu sebuah saldo dengan akumulasi jumlah yang tidak sesuai pada tabel transaksi.

Penggunaan *transaction* dapat mengatasi hal ini:

```
BEGIN;  
  
INSERT INTO nasabah (id, nama, saldo)  
VALUES (1412, 'Fernandoz', 10000);  
  
INSERT INTO transaksi (id_nasabah, jumlah)  
VALUES (1412, 10000);  
  
COMMIT;
```

Sepanjang belum di-COMMIT, maka seluruh aktivitas setelah perintah BEGIN dianggap tidak ada. Oleh karena itu jika terjadi kegagalan sesaat setelah INSERT yang pertama maka transaksi otomatis di-ROLLBACK, artinya INSERT tersebut dianggap tidak terjadi. Dengan demikian konsistensi data tetap terjaga.

Bab 8

Data dari Database Lain

Postgres memiliki `pg_dump` untuk mengubah objek database (table, view, function, dsb) beserta isi tabel menjadi script SQL. Namun sebaliknya, tidak semua database memiliki tools ini, dan seringkali kita dihadapkan untuk memindahkan data dari suatu database ke Postgres.

8.1 Perintah COPY

Postgres memiliki perintah SQL bernama `COPY` untuk memindahkan isi file ke suatu tabel. Layaknya tabel, file ini harus memiliki format baris dan kolom dimana antar kolom dipisahkan oleh karakter khusus yang secara default karakter `TAB`.

Misalkan terdapat sebuah file bernama `mhs.txt` dengan isi sebagai berikut:

```
Pribadi Endro,18-7-1975
Deni Mahmud,28-12-1976
Andre Grananda,1-4-1975
```

Field pertama berisi nama dan yang kedua berisi tanggal lahir, keduanya dipisahkan dengan koma. Untuk memasukkannya dalam database Postgres perlu disiapkan tabel berikut:

```
CREATE TABLE mhs(
  nama VARCHAR(30),
  tgl_lahir DATE);
```

Selanjutnya Anda perlu login ke database sebagai user Postgres karena user yang melakukan `COPY` perlu membuka file `mhs.txt` tadi dan ini hanya bisa dilakukan oleh user Linux.¹ Pastikan user postgres dapat membuka file `mhs.txt`, misalnya diletakkan di direktori `/tmp`.

```
SET DATESTYLE TO 'european';
COPY mhs FROM '/tmp/mhs.txt';
```

Baris pertama di atas untuk memastikan bahwa format tanggal yang digunakan adalah `DD/MM/YYYY`.

8.2 Format Lain

Ada kalanya dalam beberapa kasus kita menemui file teks dengan format berikut:

NAMA	TGL_LAHIR
Pribadi Endro	18/07/1975

¹Ingat bahwa user Postgres tidak berarti juga user Linux.

```
Deni Mahmud      28/12/1976
Andre Grananda   01/04/1975
```

```
(3 row(s) affected)
```

File di atas adalah contoh “pola” hasil query yang diberikan oleh utility `isqlw` yang diperoleh dari MS SQL dimana:

- dua baris pertama dan tiga baris terakhir tidak digunakan
- pemisah antar kolom tidak ada, sebagai gantinya digunakan lebar kolom

Sekarang kita buat suatu skenario perpindahan data yang memaksimalkan SQL:

- Buat *temporary table* bernama `x` yang hanya berisi sebuah field bertipe `text`

```
CREATE TEMP TABLE x(
  y TEXT);
```

- Gunakan perintah `COPY` untuk memasukkan isi file ke dalam tabel `x`. Jadi `mhs.txt` dianggap sebagai sebuah tabel dengan sebuah field.

```
COPY x FROM '/tmp/mhs.rpt';
```

```
SELECT * FROM x;
      y
```

```
-----
nama          tgl_lahir
-----
Pribadi Endro 18/07/1975
Deni Mahmud   28/12/1976
Andre Grananda 01/04/1975
```

```
(3 row(s) affected)
```

```
(8 rows)
```

- Buang dua baris pertama dan Anda perlu mengetahui `OID` dari dua record pertama. Tidak perlu lakukan *cut & paste*, tapi cukup simpan dalam *temporary table* yang lain (misalnya bernama `p`), dan masukkan dalam kondisi `DELETE`.

```
SELECT oid, * FROM x ORDER BY OID LIMIT 2;
```

```
oid |
-----+-----
17409882 | nama          tgl_lahir
17409883 | -----
```

```
(2 rows)
```

```
SELECT OID AS oid INTO TEMP p FROM x ORDER BY OID LIMIT 2;
DELETE FROM x WHERE OID IN (SELECT oid FROM p);
```

- Buang tiga baris terakhir.

```
SELECT oid, * FROM x ORDER BY OID DESC LIMIT 3;
```

```
oid |
-----+-----
y
```

```
-----+-----
17409889 |
17409888 | (3 row(s) affected)
17409887 |
(3 rows)
```

```
SELECT OID AS _oid INTO TEMP q FROM x ORDER BY OID DESC LIMIT 3;
DELETE FROM x WHERE OID IN (SELECT _oid FROM q);
```

```
SELECT * FROM x;
```

```

                y
-----+-----
Pribadi Endro   18/07/1975
Deni Mahmud     28/12/1976
Andre Grananda  01/04/1975
(3 rows)
```

- Lakukan query dengan fungsi SUBSTR() untuk memilah field tadi sesuai dengan lebar kolom, fungsi TRIM() untuk menghilangkan spasi, dan fungsi DATE() untuk mengkonversi TEXT menjadi DATE. Hasil query-nya dimasukkan dalam *temporary table* bernama z.

```
SET DATESTYLE TO 'european';
```

```
SELECT TRIM(SUBSTR(y,1,17)) AS nama,
DATE(TRIM(SUBSTR(y,18,10))) AS tgl_lahir
INTO TEMP z FROM x;
```

- Masukkan isi tabel z ke dalam tabel mhs.

```
INSERT INTO mhs (nama,tgl_lahir)
SELECT nama, tgl_lahir FROM z;
```


Bab 9

Pojok Admin

Topik ini membahas hal-hal yang berkaitan dengan pekerjaan seorang administrator seperti menambah *user*, *user group*, *security*, penyelamatan data, atau optimasi.

9.1 User

Pemberian hak kepada user lain dapat diberikan oleh user postgres.

```
$ psql -u templatel
Username: postgres
Password:

templatel=# CREATE USER sugiana WITH PASSWORD 'a' CREATEDB;
```

Gunakan \h untuk keterangan lebih lengkap perintah CREATE USER ini.

```
templatel=# \h CREATE USER
Command:      CREATE USER
Description:  Creates a new database user
Syntax:
CREATE USER username
    [ WITH
    [ SYSID uid ]
    [ PASSWORD 'password' ] ]
    [ CREATEDB      | NOCREATEDB ] [ CREATEUSER | NOCREATEUSER ]
    [ IN GROUP      groupname [, ...] ]
    [ VALID UNTIL   'abstime' ]
```

Berikut ini ada beberapa contoh perintah yang berkaitan dengan user:

- Buat user tanto dan masukkan dalam grup dokter:

```
CREATE USER tanto PASSWORD 'a' IN GROUP dokter;
```

- Buat user kopra dan masukkan dalam grup apotek serta tu_medis:

```
CREATE USER kopra PASSWORD 'b' IN GROUP apotek, tu_medis;
```

- Setiap user diperkenankan mengubah password dengan cara:

```
ALTER USER kopra WITH PASSWORD 'c';
```

Perubahan password dengan perintah SQL di atas hanya berfungsi jika file konfigurasi `pg_hba.conf` terkandung option password.

9.2 Grup

User group adalah kumpulan user PostgreSQL. Pengelompokan ini berguna untuk kemudahan pemberian otoritas (GRANT/REVOKE).

```
CREATE GROUP dokter;
```

Bisa juga memasukkan user dalam suatu grup dengan cara seperti ini:

```
ALTER GROUP dokter ADD USER tanto, kopra, kosud, dewi;
```

Menghapus user dari suatu grup:

```
ALTER GROUP dokter DROP USER dewi;
```

9.3 Perlindungan Konektivitas

Postgres meletakkan kebijaksanaan *security*-nya di file *pg_hba.conf*.¹Formatnya adalah:

```
host DBNAME IP_ADDRESS ADDRESS_MASK USERAUTH [AUTH_ARGUMENT]
```

Secara default, Postgres membiarkan setiap user untuk login tanpa diharuskan mengisikan password:

```
local all trust
host all 127.0.0.1 255.255.255.255 trust
```

Agar Postgres dapat diakses dari *remote host* dengan IP 192.168.1.x maka Anda perlu menambahkan baris berikut ini pada *pg_hba.conf* :

```
host template1 192.168.1.0 255.255.255.0 ident sameuser
```

Baris di atas berarti client manapun dengan IP Address 192.168.1.x dapat mengakses *template1* tanpa proses otorisasi (option *sameuser*).

Gunakan option password dimana proses otorisasi akan diterapkan.

```
host all 192.168.1.0 255.255.255.0 password
```

9.3.1 Plain Text Password

Untuk *security* yang lebih ketat lagi gunakan *plain text* untuk menyimpan password:

```
host all 192.168.1.0 255.255.255.0 password kunci
```

Contoh di atas menunjukkan bahwa user dari host dengan alamat IP 192.168.1.x boleh terkoneksi ke server Postgres dengan kewajiban mengisikan username dan password yang terdapat dalam file *kunci* yang terletak di *home directory* user Postgres.

```
$ cd /usr/local/pgsql/data
$ pg_passwd kunci
Username: owo
New password:
Re-enter new password:
```

Karena direktori dan file-nya milik user postgres maka setiap perubahan password hanya dapat dilakukan oleh user postgres ini. Password yang tertulis dalam file *kunci* akan di-encrypt agar lebih terjaga kerahasiannya.

¹Biasanya file ini terletak dalam *home directory* postgres. Namun sebenarnya Postgres melihat variabel PGDATA dalam PATH. Anda bisa mengetahui nilai dari variabel ini dengan perintah *export* pada shell. Jika Anda mengikuti proses instalasi dengan mengkompilasi maka direktorinya ada di */usr/local/pgsql/data*.

9.4 Perlindungan Data

Sistem perlindungan terhadap data dilakukan pada level tabel dan bukan pada aplikasi, yaitu dengan cara pemberian hak (GRANT) atau pencabutan hak (REVOKE) pada perintah SQL. Sehingga perubahan hak tidak akan mempengaruhi kode program. Selain superuser postgres, yang berhak memberikan GRANT / REVOKE adalah user yang membuat database bersangkutan.

Ketika tabel dibuat, hanya pemiliknya yang dapat menggunakan. Jika user lain diperbolehkan juga, maka pembuatnya harus memberikan GRANT. Hak akses yang dapat diberikan adalah SELECT, INSERT, UPDATE, DELETE, dan RULE.

```
CREATE TABLE test_izin (id INT);
```

Selama tabel tersebut belum diberikan hak apapun pada user lain, maka hanya pembuatnya yang dapat melakukan perubahan terhadap tabel tersebut.

```
GRANT SELECT ON test_izin TO anjas;
```

Namun sekali suatu tabel tercatat dapat diakses oleh user lain (pemberian GRANT), maka pembuatnya sendiri harus mendapat GRANT pula. Bila di-GRANT-kan ke grup:

```
GRANT ALL ON test_izin TO GROUP dokter;
```

ALL berarti seluruh hak akses diperbolehkan. Untuk mencabutnya, gunakan REVOKE:

```
REVOKE ALL ON test_izin FROM GROUP dokter;
```

Gunakan user PUBLIC jika Anda bermaksud meng-GRANT / REVOKE suatu tabel bagi seluruh user.

```
GRANT SELECT ON test_izin TO PUBLIC;
```

9.5 Backup

Seluruh objek (tabel, isi tabel, view, stored procedures, dll) dalam suatu database dapat “dikeluarkan” (baca: di-*backup*) dalam bentuk SQL script.

```
$ pg_dump -u -f rab.sql rab
```

`rab.sql` adalah nama *output file*, sedangkan `rab` adalah nama database. Untuk restore dapat menggunakan `psql`:

```
$ psql -u -f rab.sql rab
```

Atau bisa juga lewat prompt `psql` dengan perintah `\i`:

```
$ psql -u rab
Username: sugiana
Password:
```

```
rab=> \i rab.sql
```

Sebelum *restore* tentunya Anda harus memastikan bahwa database `rab` kosong, setidaknya tidak berisi objek yang akan di-*restore*. Mengosongkan database tidak harus dengan menghapus (DROP) tabel satu per satu, melainkan bisa dengan menghapus database dan membuatnya kembali.

Jika aplikasi Anda memanfaatkan field OID yang dimiliki Postgres gunakan option `-o` agar nilai-nilai OID tidak berubah:

```
$ pg_dump -o -u -f rab.sql rab
```

9.5.1 Dump Lewat Direktori

Perpindahan database dapat juga dilakukan dengan operasi file dan direktori. Hal seperti ini kadang dipakai manakala sistem *crash* dan kita belum sempat melakukan proses *dump*. Postgres menyimpan setiap database ke dalam suatu direktori. Setiap objeknya (seperti tabel) disimpan dalam sebuah file. Secara default seluruh direktori tersebut tersimpan dalam `/var/lib/pgsql/base` untuk RedHat dan `/var/lib/pgsql/data` untuk SuSE, atau jika Postgres hasil kompilasi dari *source* biasanya diletakkan di `/usr/local/pgsql/data`. Jadi jika Anda ingin mem-*backup* seluruh database maka Anda dapat melakukannya dengan utility kompresi biasa seperti `tar`.

```
$ cd /usr/local/pgsql
$ tar cfz backup.tgz data
```

Setelah server Postgres pulih, Anda tinggal meng-ekstrak-nya ke tempat semula :

```
$ cd /usr/local/pgsql
$ tar xfz backup.tgz
```

Perlu Anda ketahui bahwa Postgres menyimpan informasi seluruh nama database dalam database `template1` pada tabel `pg_database`. Jadi meski Anda telah meng-copy direktori database ke tempat yang sudah ditentukan namun belum mendaftarkannya ke tabel tersebut maka database tersebut tetap saja belum bisa diakses. Perlu sedikit tindakan pengakalan untuk mendaftarkannya.

```
INSERT INTO pg_database
VALUES('rab',100,'rab');
```

Tentu saja tindakan di atas tidak perlu dilakukan jika direktori yang Anda restore termasuk direktori `template1` dari sistem sebelumnya.

9.5.2 Seluruh Database

Sebuah sistem Postgres bisa terdiri dari beberapa database. Pada saat upgrade Anda tentu perlu mem-*backup* seluruh database tersebut. Untunglah postgres telah menyiapkan utility `pg_dumpall` untuk masalah ini:

```
$ pg_dumpall > db.out
```

Setelah diupgrade, lakukan restore.

```
$ psql -e template1 < db.out
```

9.6 Cleaning-up

Aktivitas query biasanya meninggalkan “sampah” dalam direktori database. Postgres menyediakan perintah `VACUUM` untuk membuang *garbage* yang tak perlu tersebut.

```
VACUUM;
```

Atau gunakan `VACUUM ANALYZE` untuk mempercepat hasil query karena perintah ini membuat statistik yang akan dimanfaatkan oleh *query optimizer*.

```
VACUUM ANALYZE;
```

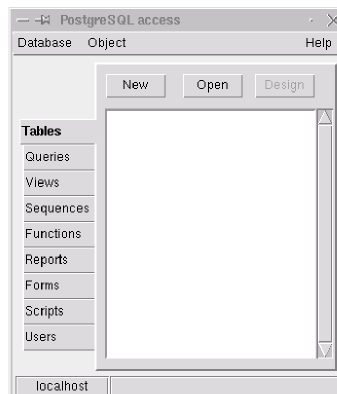
Bab 10

Aplikasi client

Selain `psql`, paket Postgres juga menyertakan *tools* lain seperti `pgaccess`. Bahkan pihak ketiga juga mengambil langkah serupa untuk meramaikan aplikasi Postgres ini.

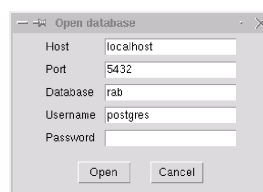
10.1 Pgaccess

Postgres telah menyertakan tool dengan antarmuka grafis bernama `pgaccess` untuk memanipulasi tabel dan objek lain postgres lainnya seperti view, function, dan sebagainya.



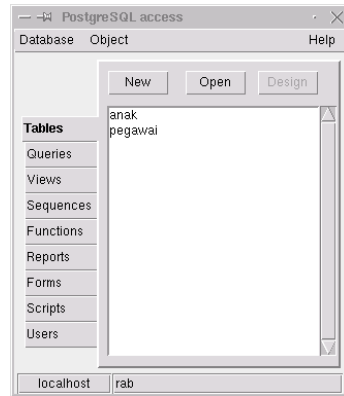
Gambar 10.1: pgaccess

Database *rab* yang sudah kita buat tadi bisa dilihat dengan memasuki menu Database | Open .



Gambar 10.2: Menu login ke host

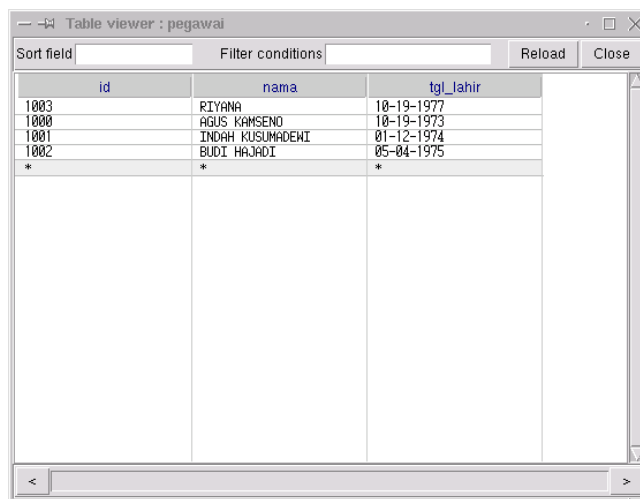
Klik Open setelah Anda mengisikan nama database, username, dan password. Pgaccess akan menampilkan tabel yang ada dalam database tersebut.



Gambar 10.3: Menu tabel

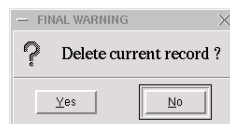
10.1.1 Table

Klik dua kali pada tabel `pegawai` dan kini dengan mudahnya Anda dapat menambah, mengubah, dan menghapus record tanpa harus tahu perintah SQL-nya.



Gambar 10.4: Menu mengubah isi tabel

Record yang diberi karakter bintang (*) digunakan untuk menambah record (`INSERT`) dan setelah Anda mengisi nilai-nilainya, record tersebut dapat disimpan dengan klik sekali pada tombol `Reload`. Sedangkan penghapusan record (`DELETE`) kita gunakan tombol `Del` pada keyboard di record yang akan dihapus. `Pgaccess` akan memberikan konfirmasi terlebih dahulu sebelum record tersebut benar-benar akan dihapus.



Gambar 10.5: Konfirmasi untuk menghapus record

Pengurutan (*sort*) berdasarkan field tertentu juga dapat dilakukan dengan mengisi nama field pada `Sort Field`. Misalkan tabel `Pegawai` ini akan diurut berdasarkan field `NAMA`, maka ketikkan "nama" pada `Sort Field` dan tekan `Enter`.

Filter layaknya `WHERE` yang berguna untuk menampilkan record sesuai dengan kondisi yang diberikan.

id	nama	tgl_lahir
1000	AGUS KAMSENO	10-19-1973
1002	BUDI HAJADI	05-04-1975
1001	INDAH KUSUMADEWI	01-12-1974
1003	RIYANA	10-19-1977
*	*	*

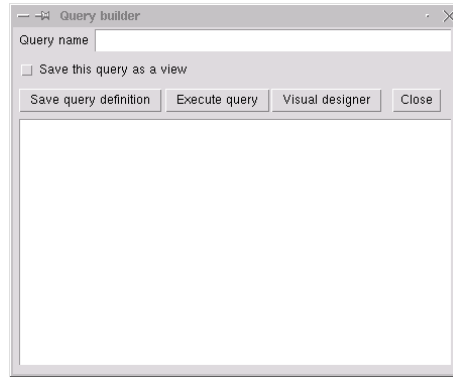
Gambar 10.6: Proses sorting tabel

id	nama	tgl_lahir
1002	BUDI HAJADI	05-04-1975
1001	INDAH KUSUMADEWI	01-12-1974
1003	RIYANA	10-19-1977
*	*	*

Gambar 10.7: Penggunaan WHERE

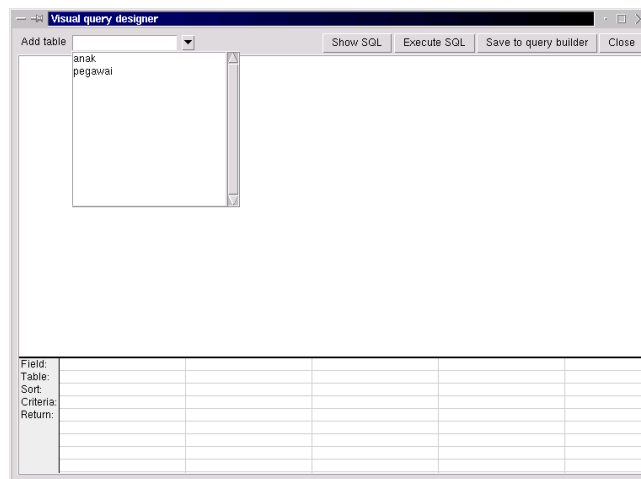
10.1.2 Query

Query-query sebelumnya yang ditulis kini dapat dibuat dengan Query Builder dengan penggunaan mouse yang optimal yaitu menerapkan mekanisme *drag & drop*. Klik Close terlebih dahulu pada Table Viewer, dan pada menu utama klik Queries | New untuk membuat query baru.



Gambar 10.8: Query builder

Klik Visual Designer dan Anda diberikan form yang memiliki *look & feel* yang mungkin Anda pernah temukan di produk lain.



Gambar 10.9: Visual designer

Sekarang kita coba membuat query yang pernah disebutkan pada contoh terdahulu yaitu menampilkan nama pegawai beserta anaknya dengan *script* sebagai berikut:

```
SELECT pegawai.id, pegawai.nama, anak.nama
FROM pegawai, anak
WHERE pegawai.id = anak.id_pegawai;
```

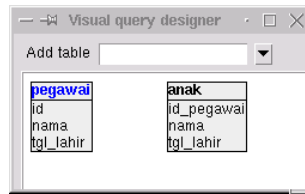
Langkah-langkah yang perlu Anda lakukan adalah:

Pilih tabel yang akan dimasukkan dalam query dengan memilihnya pada *drop-down-list*.

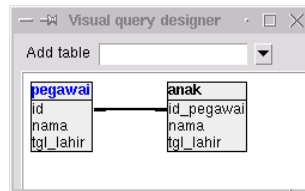
Klik field ID pada tabel Pegawai, tahan, sambil menggeser mouse ke arah field ID_PEGAWAI di tabel Anak. Mekanisme seperti ini biasa disebut *drag & drop*.

Selanjutnya di bagian paling bawah terdapat beberapa kolom yang berfungsi untuk mendefinisikan field-field yang akan ditampilkan. Anda tidak perlu mengetikkan lagi nama field tersebut, akan tetapi cukup *drag & drop* dari tabel Pegawai dan Anak yang ada di atas.

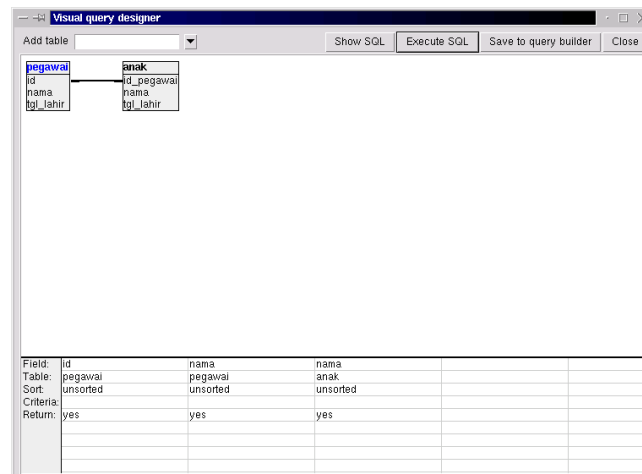
Untuk melihat hasil query klik Execute query.



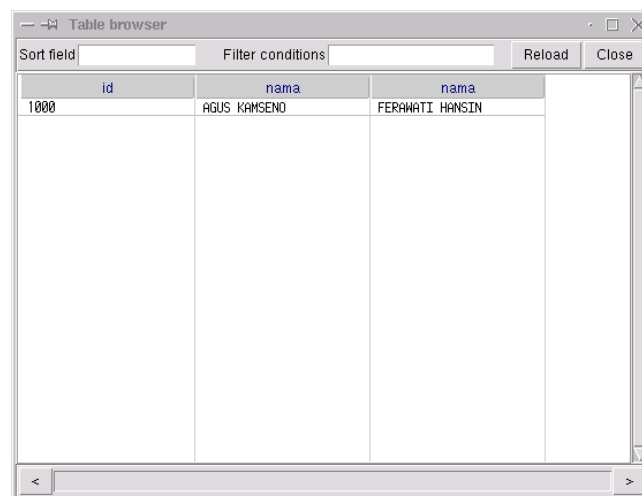
Gambar 10.10: Bekerja dengan visual designer



Gambar 10.11: Melink tabel

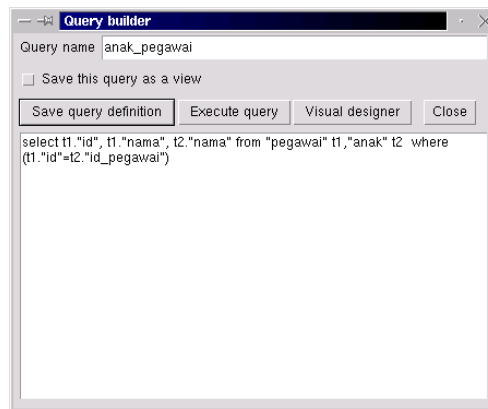


Gambar 10.12: Operasi drag & drop



Gambar 10.13: Menjalankan query

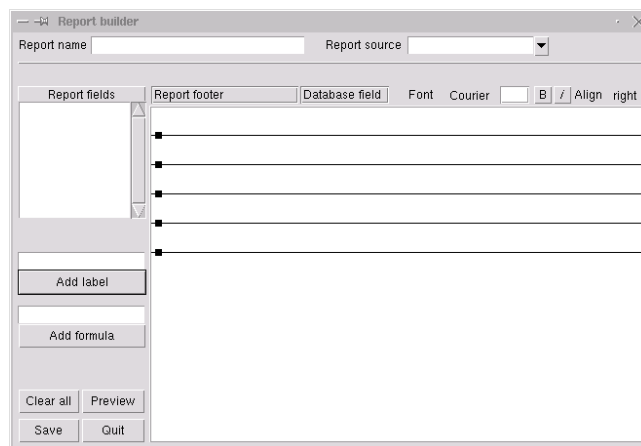
Simpan *script* yang dihasilkan dengan klik pada Save to query builder yang dilanjutkan dengan Close untuk menutup *form* tersebut. Anda akan melihat *script* yang dihasilkan dari proses *drag & drop* tadi. Berikan nama untuk *script* ini dengan “anak_pegawai” lalu klik Save query definition untuk menyimpannya.



Gambar 10.14: Pemakaian Query Builder

10.1.3 Report

Tidak lengkap rasanya kalau informasi tidak bisa dicetak. Pgaccess telah menyiapkan *Report Builder*

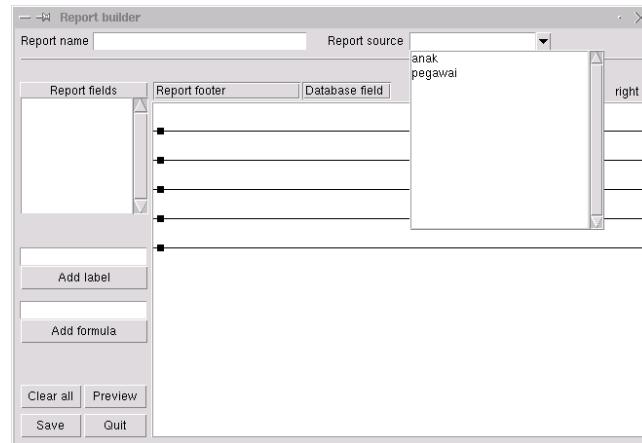


Gambar 10.15: Report Builder

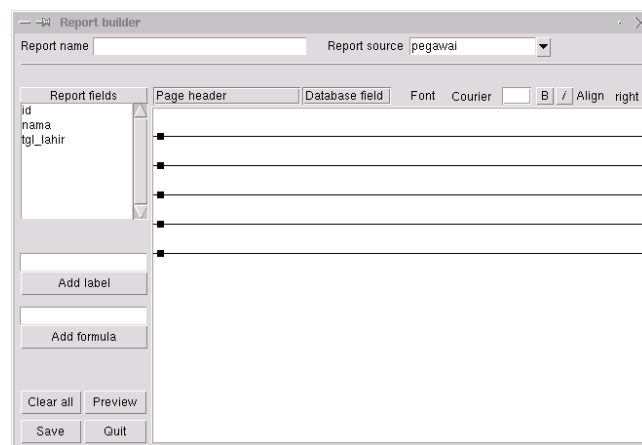
Tampak pada gambar di atas terdapat lima bidang yang dipisahkan oleh lima garis. Berturut-turut dari atas:

- **Report header:** dicetak sekali di setiap laporan, yaitu hanya pada halaman pertama, cocok untuk judul laporan
- **Page header:** dicetak di setiap halaman, biasanya digunakan untuk label kolom dan nomor halaman
- **Detail record:** dicetak di setiap record. Bidang ini untuk meletakkan item data yang ada di setiap record.
- **Page footer:** sama dengan page header, hanya posisinya ada di paling bawah di setiap halaman.
- **Report footer:** sama dengan report header, hanya posisinya ada di akhir setiap laporan. Sesuai untuk *summary*

Sekarang mulailah untuk memilih sumber data yang bisa Anda pilih pada *Report source*. Kita bisa gunakan tabel Pegawai sebagai sumber datanya.



Gambar 10.16: Menu Report Source

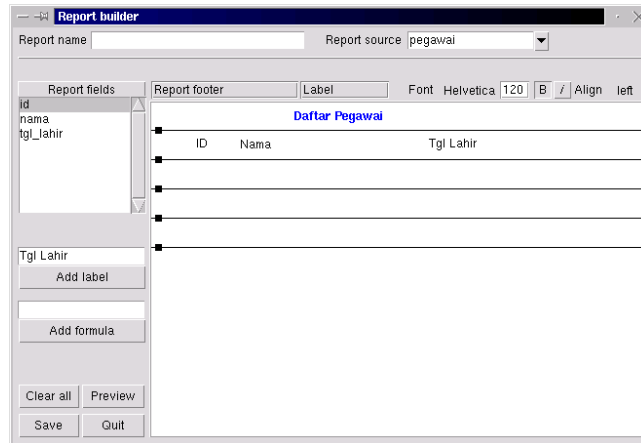


Gambar 10.17: Report field

Setelah tabel dipilih, akan tampak pada `Report fields` nama-nama field dari sumber data tersebut.

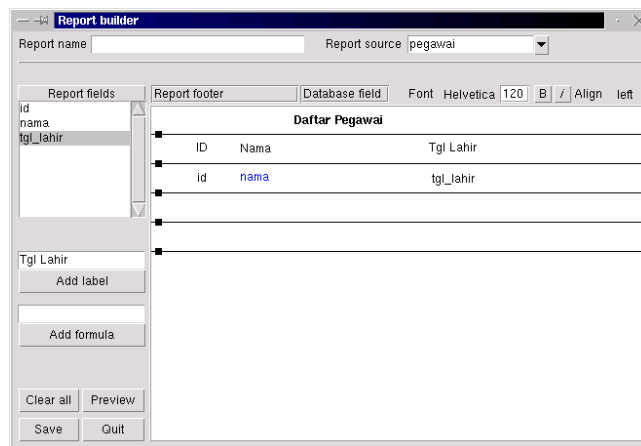
Isikan judul laporan dengan “Daftar Pegawai” pada tempat yang telah tersedia, yaitu di atas `Add label`. Selanjutnya klik tombol `Add label` agar judul tersebut berada pada bidang `Report header`. Jika ternyata `Pgaccess` meletakkan pada bidang lain, Anda bisa melakukan *drag & drop* pada judul tersebut ke bidang yang diinginkan. Hal yang sama dapat Anda lakukan dengan memberikan judul kolom seperti `ID`, `NAMA`, dan `TGL LAHIR`.

Penghapusan setiap objek label dapat dilakukan dengan klik terlebih dahulu pada objek yang dimaksud. Objek “aktif” diberi warna biru, lalu tekan `Del` pada keyboard.



Gambar 10.18: Menghapus obyek

Selanjutnya memasukkan objek field ke bidang `Detail record`. Anda cukup klik sekali pada `Report fields` dan secara otomatis objek field akan diletakkan.



Gambar 10.19: Meletakkan obyek

Selamat ! Anda kini sudah dapat melihat hasil karya Anda dengan klik pada `Preview`.

10.2 kpsql

Dalam distribusi SuSE 6.3 terdapat program `kpsql` yang berbasis KDE. Kelebihan dari `pgaccess` adalah adanya pewarnaan pada setiap sintaks SQL sehingga memudahkan pembacaan, dan fasilitas olah teks lainnya seperti `find & replace`, `cut & paste`, `save & load`, dan sebagainya.

Bagian atas tempat untuk menuliskan query sedangkan yang bawah merupakan hasil query.

Lakukan login terlebih dahulu sebelum melakukan aktivitas query.

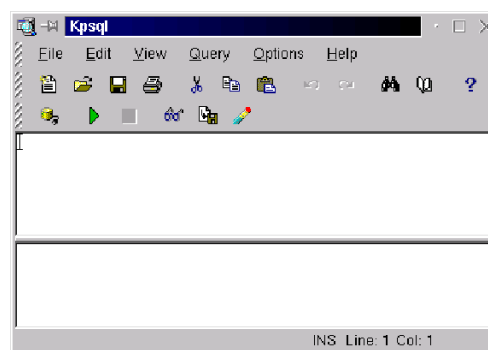


The screenshot shows a window titled "Report preview" containing a table with the following data:

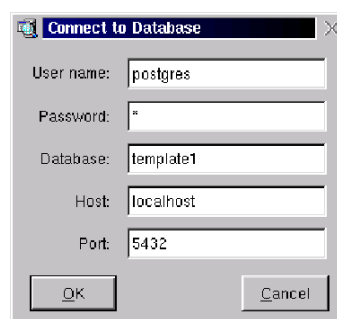
ID	Nama	Tgl Lahir
1003	RIYANA	10-19-1977
1000	AGUS KAMSENO	10-19-1973
1001	INDAH KUSUMADEWI	01-12-1974
1002	BUDI HAJADI	05-04-1975

At the bottom of the window, there are "Print" and "Close" buttons.

Gambar 10.20: Melihat hasil report



Gambar 10.21: kpsql



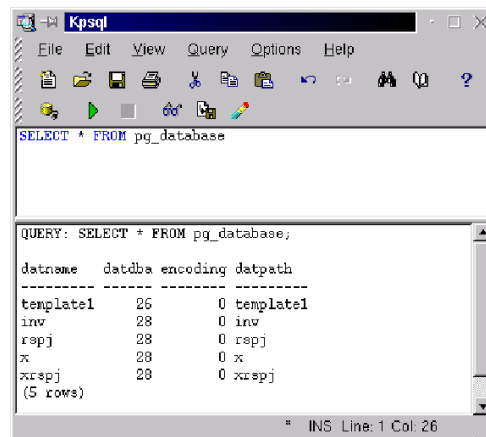
The screenshot shows a "Connect to Database" dialog box with the following fields:

- User name: postgres
- Password: *
- Database: template1
- Host: localhost
- Port: 5432

At the bottom, there are "OK" and "Cancel" buttons.

Gambar 10.22: Login di kpsql

Ada banyak shorcut yang bisa Anda gunakan. Misalnya setelah menuliskan perintah SQL tekan `Ctrl-E` untuk meng-execute-nya.



Gambar 10.23: Perintah SQL di kpsql

Bab 11

Tip dan Trik

Bab ini berisi kumpulan pengalaman penulis yang cukup bermanfaat dalam perjalanan pembuatan aplikasi database.

11.1 Format Tanggal

Dalam hal memasukkan record (INSERT) secara default format tanggal yang diperkenankan adalah MM/DD/YYYY:

```
CREATE TABLE test (tgl DATE);

INSERT INTO test VALUES ('8/17/1945');

SELECT * FROM test;

    tgl
-----
1945-08-17
```

Meski Postgres memiliki kemampuan mengenal tanggal manakala blok pertama lebih besar dari 12 maka dianggap tanggal:

```
INSERT INTO test VALUES ('17/8/1945');

SELECT * FROM test;

    tgl
-----
1945-08-17
1945-08-17
(2 rows)
```

namun kita perlu meyakinkan diri bahwa format yang dikehendaki sesuai dengan sistem Indonesia yaitu DD/MM/YYYY, sehingga '1/10/1945' pasti 1 Agustus 1945 dan bukan 10 Januari 1945. Gunakan perintah SET DATESTYLE sebelum melakukan aktivitas INSERT atau UPDATE, dan cukup dilakukan sekali dalam sebuah *database session*:

```
SET DATESTYLE TO 'european';

INSERT INTO test VALUES ('1/10/1945');

SELECT * FROM test;

    tgl
```

```

-----
1945-08-17
1945-08-17
1945-10-01
(3 rows)

```

Sebaiknya lakukan perintah `SET DATESTYLE` di atas sesaat setelah aplikasi Anda berhasil login ke database. Bisa jadi Anda belum merasa puas dengan tampilannya karena formatnya tetap saja tidak sesuai dengan nilai yang dimasukkan. Gunakan

```
SET DATESTYLE TO 'sql';
```

sebagai pelengkap

```
SET DATESTYLE to 'european';
```

Dengan demikian tampilan tanggal berformat DD/MM/YYYY:

```

SELECT * FROM test;

      tgl
-----
17/08/1945
17/08/1945
01/10/1945
(3 rows)

```

Selain lewat *set environment*, Postgres menyediakan fungsi `to_char` untuk masalah format tanggal ini:

```

SELECT to_char(tgl, 'dd-mm-yyyy') AS tgl FROM test;

      tgl
-----
17-08-1945
17-08-1945
01-10-1945
(3 rows)

```

11.2 Query Tanpa Tabel

Query dapat dilakukan tanpa menggunakan tabel. Biasanya digunakan untuk meng-*execute* suatu fungsi dan menampilkan hasilnya, seperti fungsi `timeofday()` berikut:

```

templatel=> SELECT timeofday();
timeofday
-----
Sun Apr 02 07:18:48.401226 2000 JAVT
(1 row)

```

Atau untuk melihat *current user*:

```

templatel=> select user;
getpgusername
-----
postgres
(1 row)

```


11.3 Mengubah String Menjadi Tanggal

Gunakan fungsi `text_datetime` untuk mengubah suatu string menjadi tipe tanggal dan jam (`datetime`). Format string masukan harus disesuaikan dengan format yang berlaku (lihat pembahasan Format Tanggal di atas).

```
template1=> SELECT text_datetime('31-12-2000');
text_datetime
-----
Sun 31 Dec 00:00:00 2000 JAVT
(1 row)
```

Jika Anda ingin menyertakan jam, pisahkan dengan spasi antara tanggal dan jam yang Anda masukkan:

```
template1=> SELECT text_datetime('31-12-2000 14:12:7');
text_datetime
-----
Sun 31 Dec 14:12:07 2000 JAVT
(1 row)
```

11.4 Memisahkan date dan time Pada datetime

Gunakan fungsi `date()` untuk mengubah tipe data `datetime` menjadi `date`. Kita dapat menggunakan fungsi `timeofday()` yang mengembalikan nilai bertipe `datetime` untuk contoh ini:

```
template1=> SELECT date(timeofday()), time(timeofday());
date|time
-----+-----
02-04-2000|07:21:48
(1 row)
```

11.5 Penambahan dan Pengurangan Untuk date

Tipe `date` dapat langsung ditambah atau dikurangi sebagaimana pada bilangan. Contoh berikut menggunakan fungsi `now()` untuk mengetahui tanggal kemarin, hari ini, dan esok:

```
template1=> select now()-1 as kemarin, date(now()) as kini, now()+1 as esok;
kemarin|      kini|      esok
-----+-----+-----
01-04-2000|02-04-2000|03-04-2000
(1 row)
```


Bab 12

Sekilas Object Oriented dalam Postgres

Seperti disebutkan di awal tulisan bahwa meski Postgres RDBMS, ia juga memiliki konsep object oriented. Salah satunya adalah tabel yang juga merupakan class dimana suatu tabel bisa menjadi turunan dari tabel lain.

```
create table kendaraan (roda int, jenis text);
insert into kendaraan values (2,'Sepeda Motor');
insert into kendaraan values (3,'Bajaj');
insert into kendaraan values (4,'Sedan');
insert into kendaraan values (4,'Niaga');
select * from kendaraan;
```

roda	jenis
2	Sepeda Motor
3	Bajaj
4	Sedan
4	Niaga

```
create table mobil (stir text) inherits kendaraan;
insert into kendaraan values (4,'Toyota Kijang','Kanan');
insert into kendaraan values (6,'Truk Renault','Kiri');
select * from mobil;
```

roda	jenis	stir
4	Toyota Kijang	Kanan
6	Truk Renault	Kiri

```
select * from kendaraan*;
```

roda	jenis
2	Sepeda Motor
3	Bajaj
4	Sedan
4	Niaga
4	Toyota Kijang
6	Truk Renault

Pemberian '*' menunjukkan bahwa "hal tersebut berlaku untuk seluruh keturunannya". Karena perintah `select` yang diberikan berarti "tampilkan seluruh record kendaraan berikut record lain dari tabel yang menjadi turunannya".

Daftar Pustaka

[1] PostgreSQL, *Programmer's Guide*

[2] Momjian, Bruce, *PostgreSQL: Introduction and Concepts*