BAB XII
TRIGGER, PROCEDURE, FUNCTION & VIEW

Dengan *database*, data atau informasi dapat disimpan secara permanen.Informasi yang tadinya di dalam variabel, akan segera hilang bersamaan dengan selesainya eksekusi program aplikasi. Untuk itu diperlukan *database* untuk menyimpan informasi yang ingin dipertahankan saat eksekusi selesai.

Salah satu sistem *database* (DBMS) populer saat ini adalah MySQL. Terdapat beberapa alasan mengapa MySQL dipilih sebagai DBMS, diantaranya: *freeware*, didukung hampir semua bahasa pemrograman populer saat ini, *database* tercepat (metode *one-sweep multijoin*), dan komunitas yang besar.

Dalam implementasinya, sering *programmer* hanya memanfaatkan fitur *table*. Tahukah Anda, sejak MySQL versi 5.x telah tersedia fitur lain yang sangat membantu terutama untuk aplikasi terpusat (*client-server*) atau tersebar (*distributed*), yaitu: *Trigger*, *Stored Procedure*, *Stored Function*, dan *View*.

Pendahuluan

Setiap *database* mempunyai fasilitas yang memungkinkan aplikasi-aplikasi untuk menyimpan dan memanipulasi data. Selain itu, *database* juga memberikan fasilitas lain yang lebih spesifik yang dipakai untuk menjamin konsistensi hubungan antar tabel dan integritas data di dalam *database*. *Referential integrity* merupakan sebuah mekanisme untuk mencegah putusnya hubungan master/detail. Jika *user* mencoba menghapus sebuah *field* pada tabel *master* sehingga *record* di tabel *detail* menjadi yatim (tidak mempunyai induk), *referential integrity* akan mencegahnya.

Trigger, *Stored Procedure/Function*, dan *View* merupakan komponen dan fitur *database*, yang dengan keunikan fungsi masing-masing dapat dimanfaatkan untuk menjaga, mengelola, dan membantu kinerja *database engineer* dalam upaya
terjaminnya integritas sebuah *database.*

Persiapan Data

Sekarang masuk ke bahasan utama, yaitu implementasi. Untuk menerapkan TRIGGER, PROCEDURE, FUNCTION dan VIEW dibutuhkan suatu relasi, misalkan: *mahasiswa* dan *prodi*, sebagaimana yang diilustrasikan dengan perintah SQL di bawah ini.

Membuat *database* “akademik”

```
mysql> create database akademik;
```

Menggunakan *database*

```
mysql> use akademik;
```

Membuat tabel “mahasiswa”

```
mysql> create table mahasiswa(nim char(5), nama varchar(25), alamat varchar(50),
kode_prodi char(3), primary key(nim));
```

Membuat tabel “prodi”

```
mysql> create table prodi(kode_prodi char(3), nama_prodi varchar(25),jurusan
varchar(20), primary key(kode_prodi));
```

Membuat relasi antara tabel “mahasiswa” dengan “prodi”

```
mysql> alter table mahasiswa add foreign key(kode_prodi) references prodi(kode_prodi);
```

Menginputkan 5 data ke tabel “prodi”

```
mysql> insert into prodi values('P01','Eks Ilmu Komputer','Matematika'), ('P02','Ilmu
Komputer','Matematika'), ('P03','D3 Komsi','Matematika'), ('P04','D3
Rekmed','Matematika'), ('P05','D3 Ellins','Fisika');
```

Menginputkan 3 data ke tabel “mahasiswa”

```
mysql> insert into mahasiswa values('00543','Muhammad','Karangmalang A-50',
'P01'),('10043','Ahmad Sholihun','Karangmalang D-17',’P02'),
('10041','Sugiharti','Karangmalang A-23','P02');
```

Latihan 1.

1. Tampilkan data dari tabel “prodi”, screen shoot hasil uji coba!
2. Insert Nama kalian sebagai salah satu data mahasiswa, lalu tampilkan data dari tabel “mahasiswa”, screen shoot hasil uji coba!

TRIGGER

Pernyataan CREATE TRIGGER digunakan untuk membuat *trigger*, termasuk aksi apa yang dilakukan saat *trigger* diaktifkan. *Trigger* berisi program yang dihubungkan dengan suatu tabel atau *view* yang secara otomatis melakukan suatu aksi ketika suatu baris di dalam tabel atau *view* dikenai operasi INSERT, UPDATE atau DELETE.

Sintak :

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

Keterangan :

- ***[DEFINER = { user | CURRENT_USER }]***: Definisi *user* yang sedang aktif, sifatnya opsional.
- ***trigger_name***: Nama *trigger*.
- ***trigger_time***: waktu menjalankan *trigger*. Ini dapat berupa BEFORE atau AFTER.
- BEFORE: Membuat *trigger* diaktifkan sebelum dihubungkan dengan suatu operasi.
- AFTER: Membuat *trigger* diaktifkan setelah dihubungkan dengan suatu operasi.
- ***trigger_event***: berupa kejadian yang akan dijalankan *trigger*.
- ***trigger_event*** dapat berupa salah satu dari berikut:
- INSERT : *trigger* diaktifkan ketika sebuah *record* baru disisipkan ke dalam tabel. Contoh: statemen INSERT, LOAD DATA, dan REPLACE.
- UPDATE : *trigger* diaktifkan ketika sebuah *record* dimodifikasi. Contoh: statemen UPDATE.
- DELETE : *trigger* diaktifkan ketika sebuah *record* dihapus. Contoh: statemen DELETE dan REPLACE.

Catatan : *trigger_event* **tidak** merepresentasikan statemen SQL yang diaktifkan *trigger* sebagai suatu operasi tabel. Sebagai contoh, *trigger* `BEFORE` untuk `INSERT` akan diaktifkan tidak hanya oleh statemen `INSERT` tetapi juga statemen `LOAD DATA`.

- ***tbl_name :*** Nama tabel yang berasosiasi dengan *trigger*.
- ***trigger_stmt*** **:** Statemen (tunggal atau jamak) yang akan dijalankan ketika *trigger* aktif.

Contoh yang akan dibahas adalah mencatat kejadian-kejadian yang terjadi beserta waktunya pada tabel mahasiswa, dan catatan-catatan tadi disimpan dalam table yang lain, misal **log_mhs**. Misalkan struktur tabel **log_mhs** adalah sebagai berikut.

```
mysql> describe log_mhs;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| kejadian | varchar(25) | YES  |     | NULL    |       |
| waktu    | datetime    | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

Contoh 1:

```
mysql> create trigger ins_mhs after insert on mahasiswa
-> for each row insert into log_mhs values('Tambah data',now());
mysql> insert into mahasiswa values('00631','Hanif','Kalasan','P01');
mysql> select * from log_mhs;
```

Dari contoh diatas dapat dilihat bahwa ketika satu *record* pada tabel **mahasiswa** disisipkan (*insert*), maka secara otomatis tabel **log_mhs** akan disisipkan satu *record*, yaitu kejadian `'Tambah data'` dan waktu saat *record* pada tabel **mahasiswa** disisipkan.

Latihan 2.

3. Cobalah query diatas, screen shoot hasil uji coba!

Contoh 2 :

```
mysql> create trigger updt_mhs after update on mahasiswa
-> for each row insert into log_mhs values('Ubah data',now());
mysql> update mahasiswa set nama='Moh. Riyan' where nim='00543';
mysql> select * from mahasiswa;
mysql> select * from log_mhs;
```

Dari contoh diatas dapat dilihat bahwa ketika satu *record* pada tabel **mahasiswa** diperbaharui (*update*), maka secara otomatis tabel **log_mhs** akan disisipkan satu *record*, yaitu kejadian `'Ubah data'` dan waktu saat *record* pada tabel **mahasiswa** diperbaharui.

Latihan 3.

4. Cobalah query diatas, screen shoot hasil uji coba!

Contoh 3 :

```
mysql> create trigger del_mhs after delete on mahasiswa
-> for each row insert into log_mhs values('Hapus data',now());
mysql> delete from mahasiswa where nim='00631';
mysql> select * from log_mhs;
```

Dari contoh diatas dapat dilihat bahwa ketika satu *record* pada tabel **mahasiswa** dihapus (*delete*), maka secara otomatis tabel **log_mhs** akan disisipkan satu *record*, yaitu kejadian `'Hapus data'` dan waktu saat record pada tabel **mahasiswa** dihapus.

Dalam implementasinya untuk pekerjaan sehari-hari, pembuatan *trigger* dan tabel *log*, digunakan untuk mencatat kejadian suatu tabel yang dianggap rawan serangan *cracker*. Dengan struktur *trigger* yang baik sesuai kebutuhan, *administrator* dapat melakukan pelacakan dan *recovery* data dengan cepat karena mengetahui *record* mana saja yang "diserang". Atau, dihubungkan dengan program aplikasi (*user interface*) agar mengaktifkan alarm, jika terdapat operasi *database* pada waktu yang tidak seharusnya (misalkan malam hari).

Menampilkan daftar trigger yang telah dibuat:

```
mysql> show triggers;
```

Latihan 4.

5. Cobalah query diatas, screen shoot hasil uji coba!

STORED PROCEDURE/FUNCTION

Untuk membuat *stored procedure/function* pada *database* digunakan pernyataan CREATE PROCEDURE atau CREATE FUNCTION.

2.1 PROCEDURE

Sintak :

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body
```

Keterangan :

- **`sp_name`:** Nama *routine* yang akan dibuat
- **`proc_parameter`:** Parameter *stored procedue*, terdiri dari :
- IN : parameter yang digunakan sebagai masukan.
- OUT : parameter yang digunakan sebagai keluaran
- INOUT : parameter yang digunakan sebagai masukan sekaligus keluaran.
- ***routine_body :*** terdiri dari statemen prosedur SQL yang valid.

Agar lebih jelas, perhatikan contoh penggunaannya berikut ini.

Contoh 1:

```
mysql> delimiter //
mysql> create procedure pMhsIlkom(OUT x varchar(25))
-> begin
-> select nama into x from mahasiswa where kode_prodi='P01';
-> end
-> //
```

```
mysql> call pMhsIlkom(@Nama);
-> select @Nama;
-> //
```

Dari contoh diatas terlihat bahwa parameter "x" (sebagai OUT) digunakan untuk menampung hasil dari perintah *routine_body*. Pernyataan "into x", inilah yang mengakibatkan "x" menyimpan informasi **nama** (sebagai kolom yang ter-*select*).

Untuk menjalankan *procedure* digunakan ststemen **call.** Pernyataan "call pMhsIlkom(@Nama)" menghasilkan informasi yang kemudian disimpan pada parameter "@Nama". Kemudian untuk menampilkan informasi ke layar digunakan pernyataan "select @Nama".

Latihan 5.

6. Cobalah query diatas, screen shoot hasil uji coba!

Contoh 2:

```
mysql> delimiter //
mysql> create procedure pMhs(out x varchar(25), out y varchar(3), in z
char(3))
-> begin
-> select nama,alamat into x,y from mahasiswa where kode_prodi=z;
-> end
-> //
mysql> call pMhs(@Nama,@Alamat,'P01');
mysql> select @Nama, @Alamat;
```

Dari contoh yang kedua ini terlihat bahwa parameter "z" (sebagai IN) digunakan sebagai jalur untuk masukan *routine* dan parameter "x" dan "y" digunakan untuk menampung hasil dari perintah *routine_body*. Pernyataan "into x, y", inilah yang mengakibatkan "x" dan "y" menyimpan informasi **nama** dan **alamat** (sebagai kolom yang ter-*select*).

Pernyataan "call pMhs(@Nama, @Alamat)" menghasilkan informasi yang kemudian disimpan pada parameter **@Nama** dan **@Alamat,** sedangkan parameter "z" digunakan untuk menampung string 'P01' yang kemudian digunakan untuk memproses *routine_body* . Kemudian untuk menampilkan informasi ke layar digunakan pernyataan "select @Nama, @Alamat".

Jika diperhatikan pada **contoh1** dan **contoh2**, dalam membuat *routine* selalu menggunakan **delimiter.** Hal ini digunakan untuk mengubah pernyataan *delimiter* dari ";" ke "//" ketika suatu *procedure* sedang didefinisikan. Sehingga sebelum *delimiter* ditutup, meskipun sudah ditekan **enter** masih dianggap satu-kesatuan perintah.

Jika menggunakan perintah **delimiter**, maka untuk menutupnya digunakan karakter *backslash* ('\') karena karakter ini merupakan karakter **escape** untuk MySQL.

Latihan 6.

7. Cobalah query diatas, screen shoot hasil uji coba!

2.2 FUNCTION

Secara *default*, *routine* (*procedure/function*) diasosiasikan dengan *database* yang sedang aktif. Untuk dapat mengasosiasikan *routine* secara eksplisit dengan *database* yang lain, buat *routine* dengan format: **`db_name.sp_name`**.

MySQL mengijinkan beberapa *routine* berisi statemen DDL, seperti CREATE dan DROP. MySQL juga mengijinkan beberapa *stored procedure* (tetapi tidak *stored function*) berisi statemen SQL *transaction*, seperti COMMIT. *Stored function* juga berisi beberapa statemen baik yang secara eksplisit atau implisit **commit** atau **rollback**.

Sintak :

```
CREATE FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body
proc_parameter:
[ IN | OUT | INOUT ] param_name type
func_parameter:
param_name type
type:
Any valid MySQL data type
characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
routine_body:
Valid SQL procedure statement or statements
```

Keterangan :

- **`sp_name:`** Nama *routine* yang akan dibuat
- **`proc_parameter:`** Spesifikasi parameter sebagai IN, OUT, atau INOUT valid hanya untuk PROCEDURE. (parameter FUNCTION selalu sebagai parameter IN)
- **`returns:`** Klausa RETURNS dispesifikan hanya untuk suatu FUNCTION. Klausa ini digunakan untuk mengembalikan tipe fungsi, dan *`routine_body`* harus berisi suatu statemen nilai RETURN.
- **`comment:`** Klausa COMMENT adalah suatu ekstensi MySQL, dan mungkin digunakan untuk mendeskripsikan *stored procedure*. Informasi ini ditampilkan dengan statemen SHOW CREATE PROCEDURE dan SHOW CREATE FUNCTION.

Contoh:

```
mysql> delimiter //
mysql> create function fcNamaMHS(x char(25)) returns char(40)
-> return concat('Nama : ', x);
-> //
Query OK, 0 rows affected (0.00 sec)
mysql> select fcNamaMHS('Sholihun');
```

Dari contoh diatas terlihat bahwa parameter “x” diperlakukan sebagai IN karena sebagaimana dijelaskan sebelumnya bahwa fungsi hanya bisa dilewatkan dengan parameter IN. Kemudian untuk pengembalian nilainya, digunakan tipe data dengan kisaran nilai tertentu (dalam hal ini char(40)) dengan diawali pernyataan **returns**.

Pernyataam “concat('Nama : ', x)” merupakan *`routine_body`* yang akan menghasilkan gabungan string “Nama :” dengan nilai dari parameter “x” yang didapat ketika fungsi ini

dieksekusi. Perintah yang digunakan untuk mengeksekusi fungsi adalah "select fcNamaMHS('Sholihun')".

Latihan 7.

8. Cobalah query diatas, screen shoot hasil uji coba!

Menampilkan status fungsi tertentu:

```
mysql> show function status like 'fcNamaMHS'\G
```

Berisi tentang *database* bersangkutan, tipe fungsi, definer dan lain-lain.

Latihan 8.

9. Cobalah query diatas, screen shoot hasil uji coba!

3. VIEW

Sintak :

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Keterangan :

- ***create*** **:** Statemen ini digunakan untuk membuat suatu *view* baru, atau mengganti suatu *view* yang telah ada (*exist*) jika klausa OR REPLACE diberikan.
- ***select_statement***: Suatu statemen SELECT yang menyediakan definisi dari *view*. Statemen ini dapat men-*select* dari tabel dasar atau *view* yang lain. Statemen ini membutuhkan CREATE VIEW *privilege* untuk *view*, dan beberapa *privilege* untuk setiap kolom terpilih oleh statemen SELECT.
- **[(*column_list*)]:** Daftar kolom yang akan dipilih.

View termasuk dalam komponen *database*. Secara *default*, suatu *view* baru dibuat ke dalam *database* yang diaktifkan. Untuk membuat secara eksplisit di dalam suatu *database* tertentu, maka buatlah nama *view* dengan format:

`db_name.view_name`.

Contoh yang akan diberikan adalah *view* untuk menyimpan informasi detail mahasiswa, dalam hal ini melibatkan 2 tabel, yaitu **mahasiswa** dan **prodi**.

Contoh:

```
mysql> create view vDetailMhs as
-> select m.nim, m.nama, m.alamat, p.nama_prodi, p.jurusan
-> from mahasiswa m, prodi p
-> where (m.kode_prodi=p.kode_prodi);
mysql> select * from vDetailMhs;
```

Dari contoh diatas dapat dijelaskan bahwa *view* tersebut berisi informasi mahasiswa (nim, nama, alamat) dan informasi prodi mahasiswa yang bersangkutan (nama_prodi dan jurusan). Implementasi *view* dalam program aplikasi adalah untuk memudahkan dalam mendesain laporan (*report*).

Latihan 9.

10. Cobalah query diatas, screen shoot hasil uji coba!