Modul 6
Function dan Trigger

1.1 Tujuan
a. Mahasiswa dapat mengenal Sintak SQL Lanjut
b. Mahasiswa dapat mengoperasikan Sintak SQL Lanjut

1.2 Materi
a. SQL
b. Tabel

1.3 Alat dan Bahan
a. sqlyog
b. XAMPP
c. phpmyadmin

1.4 Prosedur Praktikum
a. Peserta membaca dan mempelajari materi praktikum sebelumnya
b. Instruktur menerangkan dan menjelaskan teori dan cara kerja
c. Peserta mempraktikan materi percobaan
d. Peserta membuat penyelesaian terhadap soal latihan

1.5 Teori
Trigger

Menurut Wikipedia, **trigger** dalam database diartikan sebagai "*procedural code that is automatically executed in response to certain events on a particular table or view in a database*". Singkatnya, **trigger** merupakan sekumpulan perintah atau sintaks yang akan secara otomatis dijalankan jika terjadi operasi tertentu dalam tabel atau view. **Trigger** digunakan untuk memanggil satu atau beberapa perintah SQL secara otomatis sebelum atau sesudah terjadi proses INSERT, UPDATE atau DELETE dari suatu tabel. Sebagai contoh misalnya kita ingin menyimpan id pelanggan secara otomatis ke tabel '**log'** sebelum menghapus data di tabel **pelanggan**.

Di **MySQL**, Triggers mulai dikenal di versi MySQL 5.0, dan di versi saat ini (5.1.4) fungsionalitasnya sudah bertambah. Pada versi selanjutnya pihak pengembang MySQL berjanji akan lebih menguatkan (menambah) fitur trigger ini.

Trigger sering digunakan, antara lain untuk:

- Melakukan update data otomatis jika terjadi perubahan. Contohnya adalah dalam sistem penjualan, jika dientri barang baru maka stock akan bertambah secara otomatis.
- Trigger dapat digunakan untuk mengimplementasikan suatu sistem log. Setiap terjadi perubahan, secara otomatis akan menyimpan ke tabel log.
- Trigger dapat digunakan untuk melakukan validasi dan verifikasi data sebelum data tersebut disimpan.

1.1.1
1.1.2 Membuat Trigger Baru

Berikut ini bentuk umum perintah untuk membuat triggers:

```
CREATE TRIGGER name
[BEFORE|AFTER] [INSERT|UPDATE|DELETE]
ON tablename
FOR EACH ROW statement
```

Keterangan dari bentuk umum perintah membuat trigger:

- **name**, Nama trigger mengikuti peraturan penamaan variabel / identifier dalam MySQL
- **[BEFORE | AFTER]** digunakan untuk menentukan kapan proses secara otomatis akan dieksekusi, sebelum atau sesudah proses.
- **[INSERT | UPDATE | DELETE]** digunakan untuk menentukan event (proses) yang dijadikan trigger (pemicu) untuk menjalankan perintah-perintah di dalam triggers.
- **tablename**, merupakan nama tabel dimana trigger berada.
- **statement**, merupakan sekumpulan perintah atau query yang akan secara otomatis dijalankan jika event / proses yang didefinisikan sebelumnya aktif.

Statement atau perintah dalam trigger dapat berupa satu perintah saja, dan dapat juga beberapa perintah sekaligus. Jika terdapat beberapa perintah dalam trigger, maka gunakan perintah **BEGIN** dan **END** untuk mengawali dan mengakhiri perintah. Di dalam statement trigger, kita dapat mengakses record tabel sebelum atau sesudah proses dengan menggunakan **NEW** dan **OLD. NEW** digunakan untuk mengambil record yang akan diproses (insert atau update), sedangkan **OLD** digunakan untuk mengakses record yang sudah diproses (update atau delete). Berikut ini contoh trigger yang akan mencatat aktivitas ke tabel **log** setiap terjadi proses insert ke tabel pelanggan:

```
DELIMITER $$
CREATE TRIGGER penjualan.before_insert BEFORE INSERT ON penjualan.pelanggan
FOR EACH ROW BEGIN
INSERT INTO `log` (description, `datetime`, user_id)
VALUES (CONCAT('Insert data ke tabel pelanggan id_plg = ', NEW.id_pelanggan), now(),
user());
END;
$$ DELIMITER ;
```

1.1.3

1.1.4 Menghapus Trigger

Untuk menghapus trigger, dapat menggunakan perintah **DROP TRIGGER** dengan diikuti dengan nama tabel dan nama triggernya. Berikut ini bentuk umum dan contoh perintah untuk menghapus trigger.

Bentuk umum dan contoh menghapus trigger:

DROP TRIGGER tablename.triggername;

Contoh menghapus trigger bernama ‘before_insert’ yang ada di tabel pelanggan.

DROP TRIGGER penjualan.before_insert;

Function dan Stored Procedure

Function dan Stored Procedure merupakan fitur utama yang paling penting di MySQL 5. Function dan Stored Procedure merupakan suatu kumpulan perintah atau statement yang disimpan dan dieksekusi di server database MySQL. Dengan SP (Stored Procedure), kita dapat menyusun program sederhana berbasis sintaks SQL untuk menjalankan fungsi tertentu. Hal ini menjadikan aplikasi yang kita buat lebih efektif dan efisien.

Berikut ini beberapa keuntungan menggunakan Stored Procedure:

- **Lebih cepat**. Hal ini karena kumpulan perintah query dijalankan langsung diserver. Berbeda dengan jika dijalankan secara sekuensial di bahasa pemrograman, akan lebih lambat karena harus *“bolak-balik”* antara client dan server.
- **Menghilangkan duplikasi proses, pemeliharaan yang mudah.** Pada dasarnya operasi yang terjadi di suatu aplikasi terhadap database adalah sama. Secara umum, di alam aplikasi biasanya terdapat operasi untuk validasi data inputan, menambahkan record baru, mengubah record, menghapus record dan sebagainya. Dengan SP, mungkin kita dapat menghindari adanya duplikasi proses yang kurang lebih sama, sehingga pemeliharaannya juga jadi lebih mudah.
- **Meningkatkan keamanan database.** Dengan adanya SP, database akan lebih aman karena aplikasi yang memanggil SP tidak perlu mengetahui isi di dalamnya. Sebagai contoh, dalam proses menambahkan data (insert), kita membuat suatu SP khusus. Dengan demikian, saat client atau aplikasi akan menambahkan data (insert) maka tidak perlu tahu nama tabelnya, karena hanya cukup memanggil SP tersebut dengan mengirimkan parameter yang diinginkan.

Selanjutnya, Stored Procedure dari segi bentuk dan sifatnya terbagi menjadi 2 (dua), yaitu FUNCTION dan PROCEDURE. Perbedaan utama antara function dan procedure adalah terletak pada nilai yang dikembalikannya (di-return). Function memiliki suatu nilai yang dikembalikan (di-return), sedangkan procedure tidak. Umumnya suatu procedure hanya berisi suatu kumpulan proses yang tidak menghasilnya value, biasanya hanya menampilkan saja. *Sebagai catatan bahwa dalam modul ini jika terdapat istilah SP (Stored Procedure) maka yang dimaksud adalah Function dan Procedure.*

Hello W orld!

Sebagai contoh sederhana, kita akan membuat suatu SP yang akan menampilkan string “Hello World!” di layar hasil. Berikut ini perintah query untuk membuat SP tersebut:

DELIMITER $$

```
CREATE PROCEDURE hello()
BEGIN
SELECT "Hello World!";
END$$
DELIMITER ;
```

Untuk memanggil procedure tersebut, gunakanlah CALL. Berikut ini contoh pemanggilan procedure dan hasil tampilannya:

```
CALL hello();
```

Hasilnya sebagai berikut:

```
+-------------+
| Hello World! |
+-------------+
| Hello World! |
+-------------+
```

Membuat, Mengubah dan Menghapus SP

Membuat SP

Untuk membuat SP baru, berikut ini bentuk umumnya:

```
CREATE
[DEFINER = { user | CURRENT_USER }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body

CREATE
[DEFINER = { user | CURRENT_USER }]
FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body
```

Contoh 1. Procedure untuk menghitung jumlah pelanggan

```
DELIMITER $$
CREATE PROCEDURE jumlahPelanggan()
BEGIN
SELECT COUNT(*) FROM pelanggan;
END$$
DELIMITER ;
```

Cara pemanggilan dari procedure diatas adalah dengan menggunakan **CALL jumlahPelanggan()**. Hasilnya akan ditampilkan jumlah record dari table pelanggan.

Berikut ini bentuk lain dari contoh diatas:

```
DELIMITER $$
CREATE PROCEDURE jumlahPelanggan2(OUT hasil AS INT)
BEGIN
SELECT COUNT(*) INTO hasil FROM pelanggan;
END$$
DELIMITER ;
```

Pada bentuk procedure yang kedua di atas (**jumlahPelanggan2**), kita menyimpan hasil dari procedure ke dalam satu variabel bernama **hasil** yang bertipe **INT.** Perbedaan dari kedua bentuk di atas adalah, pada bentuk kedua, kita dapat memanggil procedure dengan SELECT, sedangkan pada yang pertama tidak bisa. Berikut ini contoh pemanggilan untuk procedure yang kedua:

```
mysql> CALL jumlahPelanggan2(@jumlah);
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @jumlah AS `Jumlah Pelanggan`;
+-----------------+
| Jumlah Pelanggan |
+-----------------+
| 5 |
+-----------------+
1 row in set (0.02 sec)
```

Contoh 2. Procedure untuk menghitung jumlah item barang yang pernah dibeli oleh satu pelanggan.

```
DELIMITER $$
CREATE PROCEDURE
jumlahItemBarang (pelanggan VARCHAR(5))
BEGIN
SELECT SUM(detil_pesan.jumlah)
FROM pesan, detil_pesan
WHERE pesan.id_pesan=detil_pesan.id_pesan
AND pesan.id_pelanggan=pelanggan;
END$$
DELIMITER ;
```

Contoh 3. Function untuk menghitung jumlah produk yang tersedia (stock) untuk satu produk tertentu.

```
DELIMITER $$
CREATE FUNCTION jumlahStockBarang(produk VARCHAR(5))
RETURNS INT
BEGIN
DECLARE jumlah INT;
SELECT COUNT(*) INTO jumlah FROM produk
WHERE id_produk=produk;
RETURN jumlah;
END$$
DELIMITER ;
```

Untuk memanggil suatu function, kita tidak menggunakan CALL, tetapi langsung dapat memanggil dengan SELECT. Berikut ini contoh pemanggilan untuk fungsi di atas.

```
SELECT jumlahStockBarang('B0001');
```

Dan berikut ini hasilnya:

```
+--------------------------+
| jumlahStockBarang('B0001') |
+--------------------------+
| 1 |
+--------------------------+
```

Mengubah SP

Untuk mengubah SP yang sudah ada, berikut ini bentuk umumnya:

ALTER {**PROCEDURE** | **FUNCTION**} *sp_name*
[*characteristic* ...]

Menghapus SP

Untuk menghapus SP yang sudah ada, berikut ini bentuk umumnya:

DROP {**PROCEDURE** | **FUNCTION**} [IF EXISTS] *sp_name*

Sintaks Dasar dalam SP

SP dapat dikatakan sebagai bahasa pemrograman yang berada di dalam database. Oleh karena itu, tentunya terdapat sintaks-sintaks tertentu berhubungan dengan SP tersebut, misalnya bagaimana untuk mendeklarasikan variabel, penyeleksian kondisi, perulangan dsb. Pada bagian ini akan diuraikan beberapa sintaks dasar SP yang didukung oleh MySQL.

Variabel

Variabel digunakan untuk menyimpan suatu nilai secara temporer (sementara) di memory. Variabel akan hilang saat sudah tidak digunakan lagi. Variabel dalam MySQL sebelum dapat digunakan, pertama kali harus dideklarasikan terlebih dahulu. Berikut ini bentuk umum pendeklarasian suatu variabel di MySQL:

DECLARE variable_name **DATATYPE** [**DEFAULT** value];

Contohnya:

```
DECLARE jumlah INT;
DECLARE kode VARCHAR(5);
DECLARE tgl_lahir DATE DEFAULT '1982-10-20';
```

Setelah dideklarasikan, suatu variabel dapat diisi dengan suatu nilai sesuai dengan tipe data yang didefinisikan saat pendeklarasian. Untuk mengisikan nilai ke dalam suatu variabel, digunakan perintah SET. Format umumnya sebagai berikut:

SET *variable_name = expression|value;*

Contohnya:

```
SET jumlah = 10;
SET kode = (SELECT id_pelanggan FROM pelanggan LIMIT 1);
SET tgl_lahir = now();
```

Berikut ini contoh function **hitungUmur()** untuk menghitung umur seseorang saat ini berdasarkan tahun kelahiran yang diberikan.

```
DELIMITER $$
CREATE FUNCTION hitungUmur (lahir DATE)
RETURNS INT
BEGIN
DECLARE thn_sekarang, thn_lahir INT;
SET thn_sekarang = YEAR(now());
SET thn_lahir = YEAR (lahir);
RETURN thn_sekarang - thn_lahir;
END$$
DELIMITER ;
```

Penyeleksian Kondisi

Dengan adanya fasilitas penyeleksian kondisi, kita dapat mengatur alur proses yang terjadi dalam database kita. Di MySQL, penyeleksian kondisi terdiri dari IF, IF...ELSE dan CASE. Berikut ini bentuk umum ketiga perintah tersebut:

```
IF kondisi THEN
perintah-jika-benar;
END IF;
IF kondisi THEN
perintah-jika-benar;
ELSE
perintah-jika-salah;
END IF;
CASE expression
WHEN value THEN
statements
[WHEN value THEN
statements ...]
 [ELSE
statements]
END CASE;
```

Berikut ini contoh penggunaan perintah IF dalam fungsi **cekPelanggan**() dimana fungsi ini memeriksa apakah pelanggan sudah pernah melakukan transaksi pemesanan barang. Jika sudah pernah, tampilkan pesan berapa kali melakukan pemesanan, jika belum tampilkan pesan belum pernah memesan.

```
DELIMITER $$
CREATE FUNCTION cekPelanggan (pelanggan varchar(5))
RETURNS VARCHAR (100)
BEGIN
DECLARE jumlah INT;
SELECT COUNT(id_pesan) INTO jumlah FROM pesan
WHERE id_pelanggan=pelanggan;
IF (jumlah > 0) THEN
RETURN CONCAT("Anda sudah bertransaksi sebanyak ",
jumlah, " kali");
ELSE
RETURN "Anda belum pernah melakukan transaksi";
END IF;
END$$
DELIMITER ;
```

Dan berikut ini contoh penggunaan perintah CASE dalam fungsi **getDiskon**() dimana fungsi ini menentukan diskon berdasarkan jumlah pesanan yang dilakukan.

```
DELIMITER $$
CREATE FUNCTION getDiskon(jumlah INT) RETURNS int(11)
BEGIN
DECLARE diskon INT;
CASE
WHEN (jumlah >= 100) THEN
SET diskon = 10;
WHEN (jumlah >= 50 AND jumlah < 100) THEN
SET diskon = 5;
WHEN (jumlah >= 20 AND jumlah < 50) THEN
SET diskon = 3;
ELSE SET diskon = 0;
END CASE;
RETURN diskon;
END$$
DELIMITER ;
```

Perulangan

Selain penyeleksian kondisi, MySQL juga mendukung adanya perulangan dalam querynya. Perulangan biasanya digunakan untuk mengulang proses atau perintah yang sama. Dengan perulangan, perintah akan lebih efisien dan singkat. Berikut ini bentuk-bentuk perintah perulangan:

```
[label:] LOOP
statements
END LOOP [label];

[label:] REPEAT
statements
UNTIL expression
END REPEAT [label]

[label:] WHILE expression DO
statements
END WHILE [label]
```

Contoh perulangan dengan LOOP

```
SET i=1;
ulang: WHILE i<=10 DO
IF MOD(i,2)<>0 THEN
SELECT CONCAT(i," adalah bilangan ganjil");
END IF;
SET i=i+1;
END WHILE ulang;
```

1.6 Latihan

1. Coba dulu semua contoh dalam modul 6, sampai bisa running.
2. Buatlah function untuk soal no 2 pada modul 5.