MODUL 6
TRIGGER

A. TUJUAN

- ✓ Memahami konsep dasar trigger di dalam basis data.
- ✓ Memahami implementasi trigger sebagai bentuk respon atas suatu kejadian.
- ✓ Mampu menyelesaikan kasus-kasus manipulasi data yang kompleks dengan memanfaatkan trigger.

B. PETUNJUK

- Awali setiap aktivitas dengan doa, semoga berkah dan mendapat kemudahan.
- Pahami tujuan, dasar teori, dan latihan-latihan praktikum dengan baik dan benar.
- Kerjakan tugas-tugas praktikum dengan baik, sabar, dan jujur.
- Tanyakan kepada asisten/dosen apabila ada hal-hal yang kurang jelas.

C. DASAR TEORI

1. Trigger

Trigger dapat didefinisikan sebagai himpunan kode (prosedural) yang dieksekusi secara otomatis sebagai respon atas suatu kejadian berkaitan dengan tabel basis data. Kejadian (*event*) yang dapat membangkitkan trigger umumnya berupa pernyataan **INSERT**, **UPDATE**, dan **DELETE**.

Berdasarkan ruang lingkupnya, trigger diklasifikasikan menjadi dua jenis : *row trigger* dan *statement trigger*. Trigger baris (*row)* mendefinisikan aksi untuk setiap baris tabel : trigger pernyataan hanya berlaku untuk setiap pernyataan **INSERT**, **UPDATE**, dan **DELETE**.

Dari sisi perilaku (*behavior)* eksekusi, trigger dapat dibedakan menjadi beberapa jenis; namun umumnya ada dua jenis: trigger **BEFORE** dan **AFTER**. Sesuai penamaannya, jenis-jenis ini merepresentasikan waktu eksekusi trigger-misalnya sebelum ataukah sesudah pernyataan-pernyataan yang berkorespondensi.

Adakalanya trigger dipandang sebagai bentuk spesifik dari stored procedure (terkait pendefinisian *body*). Bagaimanapun, trigger akan dipanggil (secara otomatis) ketika *event* terjadi, sedangkan stored procedure harus dipanggil secara eksplisit.

2. Trigger MySQL

MySQL mendukung fitur trigger—termasuk juga stored procedure dan view-sejak versi 5.0.2. Sebagaimana objek-objek lainnya, trigger diciptakan menggunakan pernyataan **CREATE**.

Sintaks pendefinisian *trigger* diperlihatkan sebagai berikut :

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

MySQL tidak mengizinkan multiple trigger dengan waktu aksi dan event sama per tabel. Misalkan di tabel A sudah didefinisikan trigger **AFTER INSERT**, maka kita tidak boleh mendefinisikan trigger **AFTER INSERT** lagi; namun **AFTER EDIT**, **AFTER DELETE**, atau **BEFORE** (**INSERT**, **EDIT**, dan **DELETE**) bisa diterima.

D. LATIHAN

Dalam latihan ini digunakan dua buah tabel bernama *'barang'* dan *'pembelian'* dengan struktur seperti berikut :

```
CREATE TABLE barang (
id_brg varchar (5) NOT NULL,
nama_brg varchar (10) NOT NULL,
stok int (5) NOT NULL,
PRIMARY KEY (id_brg)
) ENGINE = MyISAM;

CREATE TABLE pembelian (
id_pem int (5) NOT NULL,
id_brg varchar (5) NOT NULL,
jml_beli int (5) NOT NULL,
PRIMARY KEY (id_pem)
) ENGINE = MyISAM;
```

Data yang digunakan adalah sebagai berikut:

Tabel barang

id_brg	nama_brg	stok
A10	Mouse	10
A11	Keyboard	15
A12	DVD-RW	10

Tabel pembelian

id_pem	id_brg	jml_beli
1	A10	5

1. Menggunakan Trigger

Operasi-operasi berkenaan dengan pendefinisan trigger tidak berbeda dengan objek-objek database lainnya.

1. Ketikkan pernyataan pembuatan trigger berikut di editor teks.

```
DELIMITER //

CREATE TRIGGER inkremenStok
  BEFORE INSERT ON barang

  FOR EACH ROW BEGIN

    SET NEW.stok = NEW.stok + 1;

END //
```

Pernyataan di atas memiliki arti untuk menambah nilai stok dengan nilai satu setiap kali ada penambahan data baru.

2. Eksekusi file trigger (sesuaikan path lokasi penyimpanan file).
3. Berikan perintah untuk menambah data baru, misalnya seperti berikut:

```
INSERT INTO barang
VALUES ('A13', 'Modem', 5);
```

4. Periksa hasilnya:

```
mysql> SELECT * FROM barang;
+------------+-------------+------+
| kodebarang | nama_barang | stok |
+------------+-------------+------+
| A10        | Mouse       |   10 |
| A11        | Keyboard    |   15 |
| A12        | DVD-RW      |   10 |
| A13        | Modem       |    6 |
+------------+-------------+------+
4 rows in set (0.00 sec)
```

Terlihat bahwa trigger sudah bekerja seperti yang diharapkan, di mana setiap penambahan baru akan menginkremen nilai stok.

Untuk mendapatkan informasi mengenai daftar trigger yang telah terdefinisi, gunakan perintah **SHOW TRIGGERS**(tambahkan \G untuk menampilkan data secara vertikal)

```
mysql> SHOW TRIGGERS\G
*************************** 1. row ***************************
             Trigger: inkremenStok
               Event: INSERT
               Table: barang
           Statement: BEGIN
SET NEW.stok = NEW.stok + 1;
END
              Timing: BEFORE
             Created: NULL
            sql_mode:
             Definer: root@localhost
character_set_client: cp850
collation_connection: cp850_general_ci
  Database Collation: latin1_swedish_ci
```

Sebagaimana objek-objek database lainnya, kita menghapus trigger dengan menggunakan perintah DROP

```
mysql> DROP TRIGGER inkremenStok;
Query OK, 0 rows affected (0.00 sec)
```

Untuk memastikan bahwa operasi penghapusan sudah berhasil, periksa kembali eksistensi trigger.

```
mysql> SHOW TRIGGERS\G
Empty set (0.02 sec)
```

2. Keyword OLD dan NEW

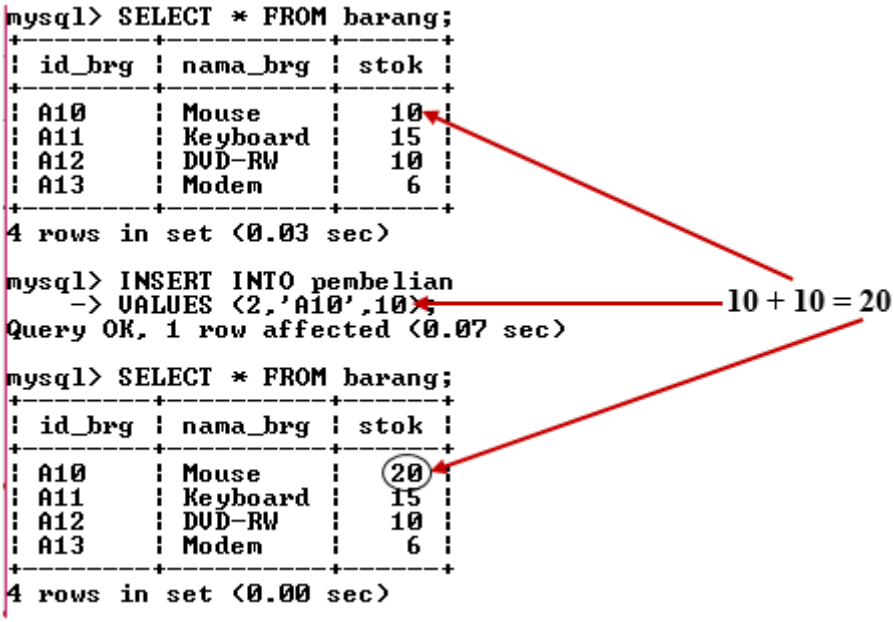
Untuk merujuk ke kolom-kolom tabel yang diasosiasikan dengan trigger, kita menggunakan keyword **OLD** dan **NEW**. Keyword OLD mengacu pada nilai lama, sedangkan **NEW** merepresentasikan nilai baru.

Di trigger **INSERT**, kita hanya dapat menggunakan keyword **NEW** karena tidak ada data lama.

Contoh berikut memperlihatkan penggunaan keyword **NEW** pada trigger **INSERT**.

```
DELIMITER //

CREATE TRIGGER updateStok
   AFTER INSERT ON pembelian

   FOR EACH ROW BEGIN

      UPDATE barang
      SET stok = stok + NEW.jml_beli
      WHERE id_brg = NEW.id_brg;

   END//
```

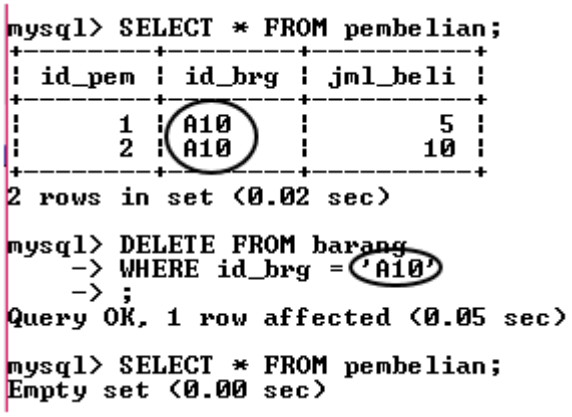
Pada contoh di atas, penambahan data pembelian akan mengakibatkan nilai stok barang berubah menyesuaikan banyaknya nilai jumlah pembelian.



Untuk kasus trigger **DELETE**, keyword yang bisa digunakan hanya **OLD**. Misalkan kita ingin mendefinisikan trigger untuk menghapus semua data pembelian manakala data barang yang sesuai diindikasikan melalui *primary key* dan *foreign key* dihapus.

```
DELIMITER //

CREATE TRIGGER deleteChild
AFTER DELETE ON barang

FOR EACH ROW BEGIN

DELETE FROM pembelian
WHERE id_brg = OLD.id_brg;

END//
```

Contoh penggunaan trigger delete Child diperlihatkan sebagai berikut:



Khusus untuk operasi **UPDATE**, kita bisa memanfaatkan keyword **NEW** maupun **OLD**.

```
DELIMITER //

CREATE TRIGGER updateStokEdit
AFTER UPDATE ON pembelian

FOR EACH ROW BEGIN

UPDATE barang
SET stok = stok + (NEW.jml_beli - OLD.jml_beli)
WHERE id_brg = NEW.id_brg;

END//
```

```
mysql> UPDATE pembelian
    -> SET jml_beli = 20
    -> WHERE id_pem = 3;
```

```
mysql> SELECT * FROM pembelian;
+--------+--------+----------+
| id_pem | id_brg | jml_beli |
+--------+--------+----------+
|      3 | A11    |       20 |
+--------+--------+----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM barang;
+--------+----------+------+
| id_brg | nama_brg | stok |
+--------+----------+------+
| A11    | Keyboard |   35 |
| A12    | DVD-RW   |   10 |
| A13    | Modem    |    6 |
+--------+----------+------+
3 rows in set (0.00 sec)
```

3. Trigger Kompleks

Keberadaan trigger secara nyata mampu mengatasi berbagai persoalan pelik, misalnya berkaitan dengan integritas atau audit data. Ini tentu sangat masuk akal, karena trigger juga bisa mengandung pernyataan-pernyataan yang kompleks termasuk percabangan, pengulangan, fungsi-fungsi agregat, bahkan kode pemanggilan prosedur. Sebagai ilustrasi sederhana, kita bisa mendefinisikan trigger untuk memeriksa operasi penambahan data **barang**. Skenarionya, jika data sudah ada, berikan status eksistensi barang; sebaliknya, data bisa langsung dimasukkan.

```
DELIMITER //
CREATE TRIGGER auditBarang
BEFORE INSERT ON barang
FOR EACH ROW BEGIN
IF NOT EXISTS (SELECT id_brg FROM barang WHERE id_brg = NEW.id_brg)
THEN
SET NEW.nama_brg = NEW.nama_brg, NEW.stok = NEW.stok;
ELSE
SET @status = CONCAT('id', NEW.id_brg, 'sudah ada');
END IF;
END//
```

Periksa dan analisis hasilnya !

E. TUGAS PRAKTIKUM

1. Modifikasi trigger **INSERT** pembelian untuk menambahkan fitur pemberian bonus. Skenarionya, jika pembelian lebih dari 150 dan kurang dari 250, bonus 15; pembelian lebih dari 250 dan kurang dari 350, bonus 25; pembelian lebih dari 350, bonus 50. Ingat, untuk aturan penyesuaian stok barang juga masih berlaku !
2. Buat tabel **pembayaran** dengan *field* **id_pem** dan **jumlah**, kemudian tambahkan *field* **harga** di tabel barang. Modifikasi trigger **INSERT** pembelian untuk menambahkan fitur penghitungan nota pembayaran. Misalkan harga satuan barang A adalah Rp. 2000, dan terjadi pembelian 15 barang, maka jumlah pembayaran tercatat Rp. 30000. Ingat, aturan penyesuaian stok barang dan juga pemberian bonus masih berlaku di sini.
3. Buat tabel **log_pembelian** dengan *field* **waktu** dan **operasi**. Selanjutnya, definisikan trigger di tabel pembelian untuk merekam operasi **INSERT**, **UPDATE**, dan **DELETE**, dan kemudian menyimpannya sebagai bukti transaksi di tabel **log_pembelian.** **Contoh hasil Operasi:**

```
INSERT: Menambah data bla bla bla...
UPDATE: Mengubah data ...  menjadi ... , ... menjadi ...
DELETE: Mengahapus data bla bla bla
```

> ***Note***
>
> *Untuk mencegah duplikasi pendefinisian trigger, terlebih dahulu hapus semua trigger di tabel pembelian*

F. TUGAS

1. Buat tabel **siswa** dengan *field* ***nis, nama,*** dan ***status*** serta tabel **daftarnilai** dengan *field* ***nis*** dan ***nilai.*** Selanjutnya, definisikan trigger pada tabel daftarnilai**,** sehingga saat melakukan **INSERT** dan **UPDATE** pada tabel daftarnilai, *field* **status** akan otomatis terisi "LULUS" atau "TIDAK LULUS" dengan ketentuan apabila **nilai < 75** dinyatakan tidak lulus.
2. Buatlah dua tabel **dosen_pa** (field : ***id_pa, nama_pa, jml_mhs***) dan **mahasiswa** (field: ***nim, nama, id_pa***). Kemudian buatlah trigger pada tabel mahasiswa untuk menghitung jumlah mahasiswa yang dibimbing oleh masing-masing dosen pa. Skenarionya, ketika terjadi **INSERT**,**UPDATE** dan **DELETE**pada tabel mahasiswa, field jml_mhs pada tabel dosen_PA akan terupdate secara otomatis.

Tabel dosen_pa:

id_pa	nama_pa	jml_mhs
001	Hamdan, S.Kom., M.T.	3
002	Munir S, S.Kom., M.T.	2
003	Junaedy, S.Kom., M.T.	1
004	Izmy, S.Kom., M.CS.	0
005	Dr. Sakti, S. T., M.Kom.	0

Tabel mahasiswa:

nim	nama	id_pa
52015001	Arif	001
52015002	Budi	003
52015003	Christin	002
52015004	Dedi	001
52015005	Ekawati	002
52015006	Firman	001

3. Terdapat 3 tabel **mhs, mk** dan **nilaimhs** dengan data seperti dibawah ini :
   Tabel mhs :

nim	nama	ip
52015001	Arif	3.25
52015002	Budi	0.0
52015003	Christin	0.0

Tabel mk :

kode_mk	nama_mk	sks
13520015	Praktikum Basis Data	1
13510008	Praktikum Basis Data	1
13520013	Sistem Basis Data	3
13500020	Jaringan Komputer	2
13500016	Sistem Operasi	3
13523006	Grafika Multimedia	3
13529006	Sistem Informasi	2

Tabel nilaimhs :

nim	kode_mk	nilai_abjad
52015001	13520015	A
52015001	13500020	B

Buatlah Trigger untuk tabel ***nilaimhs*** agar jika terdapat **INSERT**, **UPDATE** dan **DELETE** pada tabel ***nilaimhs***, maka data pada ***field ip*** di tabel **mhs** secara otomatis terupdate.

Rumus menghitung Ip (Indeks Prestasi):

$$Ip = \frac{jumlah\ Bobot\ mk\ yang\ diambil}{jumlah\ SKS\ mk\ yang\ diambil}$$

Dimana, **bobot =**

Jika Nilai Abjad **"A"** = sks x 4 Jika Nilai Abjad **"C"** = sks x 2

Jika Nilai Abjad **"B"** = sks x 3 Jika Nilai Abjad **"D"** = sks x 1