View, Stored Procedure, dan Trigger di MySQL

Private Training Web Development Padang

10/31/2014

www.phpmu.com

Membuat View di MySQL

View adalah perintah query yang disimpan pada database dengan suatu nama tertentu, sehingga bisa digunakan setiap saat untuk melihat data tanpa menuliskan ulang query tersebut.

Syntax dasar perintah untuk membuat view adalah sebagai berikut :

```
CREATE
  [OR REPLACE]
      VIEW view_name [(column_list)]
         AS select_statement
```

Kita menggunakan opsi OR REPLACE jika kita ingin mengganti view dengan nama yang sama dengan perintah tersebut. Jika tidak maka perintah CREATE VIEW akan menghasilkan error jika nama view yang ingin dibuat sudah ada sebelumnya.

Contoh Penggunaan

Kita akan membuat view dari relasi antara table "**ms_harga_harian**", "**ms_cabang**" dan "**ms_produk**" dari database minimarket dengan nama "**view_harga**". Perintahnya adalah sebagai berikut :

```
CREATE VIEW view_harga
AS
SELECT m1.kode_produk, m2.nama_produk,
m1.kode_cabang, m3.nama_cabang,
m1.tgl_berlaku, m1.harga_berlaku_cabang
FROM
ms_harga_harian m1 JOIN ms_produk m2 ON m1.kode_produk = m2.kode_produk
JOIN ms_cabang m3 ON m1.kode_cabang = m3.kode_cabang;
```

Eksekusi perintah berikut untuk memastikan view telah dibuat :

```
SELECT * FROM information_schema.views WHERE table_name = 'view_harga';
```

Terakhir, query view tersebut untuk melihat hasilnya :

```
SELECT * FROM view_harga;
```

Membuat Stored Procedure di MySQL

Stored procedure adalah salah satu objek *routine* yang tersimpan pada database MySQL dan dapat digunakan untuk menggantikan berbagai kumpulan perintah yang sering kita gunakan, seperti misalkan sejumlah row ke table lain dengan filter tertentu.

Stored procedure sangat berguna ketika kita tidak ingin user mengakses table secara langsung, atau dengan kata lain membatasi hak akses user dan mencatat operasi yang dilakukan. Dengan demikian resiko kebocoran dan kerusakan data dapat lebih diminalisir.

Pembuatan Stored Procedure

Kita dapat membuat trigger dengan perintah CREATE PROCEDURE. Berikut adalah syntax lengkapnya :

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

Keterangan :

- **sp_name** : nama stored procedure.
- **proc_parameter** : parameter input / output dari stored procedure tersebut (opsional).
- **characteristic** : menjelaskan karakteristik dari stored procedure (COMMENT, LANGUAGE SQL, dan lain-lain).
- **routine_body** : kumpulan perintah pada stored procedure tersebut.
- Jika **DEFINER** dispesifikasikan maka kita memutuskan trigger tersebut dijalankan hanya oleh user tertentu (dalam format penulisan **user@host**). Jika tidak dispesifikasikan, maka user yang melakukan perubahan (CURRENT_USER) adalah pilihan *default*.

Contoh Penggunaan

Berikut adalah contoh pembuatan dan penggunaan stored procedure untuk menghapus data berdasarkan "**kode produk**" untuk tiga table yaitu table "**ms_produk**", "**ms_harga_harian**", dan "**tr_penjualan**".

1. Buatlah satu stored procedure dengan nama HapusProduk, dengan satu argumen yaitu kode_produk_param bertipe teks (varchar) dengan perintah berikut :

```
DELIMITER |
CREATE PROCEDURE HapusProduk(IN kode_produk_param VARCHAR(12))
BEGIN
    DELETE FROM ms_produk WHERE kode_produk = kode_produk_param;
    DELETE FROM ms_harga_harian WHERE kode_produk = kode_produk_param;
    DELETE FROM tr_penjualan WHERE kode_produk = kode_produk_param;
END;
|
DELIMITER ;
```

2. Setelah selesai dieksekusi, pastikan stored procedure tersebut sudah terbentuk di database kita.

```
SELECT routine_name, routine_type, routine_schema
FROM information_schema.routines
Where routine_type = 'PROCEDURE'
AND routine_schema ='minimarket'
```

3. Sebelum kita mengeksekusi stored procedure tersebut. Kita coba lihat hasil query untuk produk "PROD-0000002" untuk ketiga table yang disebutkan di atas. Ini untuk memastikan adanya row untuk produk tersebut.

```
SELECT * FROM tr_penjualan WHERE kode_produk = 'PROD-0000002'
SELECT * FROM ms_harga_harian WHERE kode_produk = 'PROD-0000002'
SELECT * FROM ms_produk WHERE kode_produk = 'PROD-0000002'
```

4. Sekarang coba panggil stored procedure HapusProduk dengan parameter "PROD-0000002" dan tunggu beberapa saat sampai eksekusi selesai.

```
CALL HapusProduk('PROD-0000002');
```

5. Silahkan Cek lagi Kode Produk yang di hapus.
6. Selesai,..

Membuat Trigger di MySQL

Trigger adalah suatu objek database yang merupakan aksi atau prosedur yang dilakukan jika terjadi perubahan pada row data suatu table. Trigger tidak dapat menjadi bagian dari temporary table atau view.

Beberapa contoh penggunaan trigger yang sangat berguna adalah jika kita ingin melakukan kalkulasi tertentu yang tidak perlu "diketahui" aplikasi luar, mencatat aktivitas operasi table misalkan untuk kepentingan change data capture (CDC), dan lain-lain.

Pembuatan Trigger

Kita dapat membuat trigger dengan perintah CREATE TRIGGER. Berikut adalah syntax lengkapnya :

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

Keterangan :

- **trigger_name** : nama trigger.
- **trigger_time** : kapan kita mengeksekusi trigger, apakah sebelum atau sesudah perubahan pada row data table. Jadi pilihannya adalah **AFTER** atau **BEFORE**.
- **trigger_event** : merupakan event atau peristiwa yang menyebabkan trigger dilakukan. Pilihan event tersebut adalah **INSERT**, **UPDATE**, **DELETE.**
- **tbl_name** : nama table.
- **trigger_body** : *statement-statement* perintah SQL yang akan dilakukan. Jika perintahnya lebih dari satu maka gunakan dalam blok statement **BEGIN … END**.
- Jika **DEFINER** dispesifikasikan maka kita memutuskan trigger tersebut dijalankan hanya oleh user tertentu (dalam format penulisan **user@host**). Jika tidak dispesifikasikan, maka user yang melakukan perubahan (CURRENT_USER) adalah pilihan *default*.

Referensi "OLD" dan "NEW"

Karena trigger digunakan pada saat terjadi perubahan row data, maka kita perlu referensi ke row sebelum dan sesudah perubahan. Untuk ini ada dua alias yang berfungsi untuk hal tersebut yaitu **OLD** dan **NEW**.

Sesuai namanya, OLD digunakan untuk referensi sebelum perubahan dan NEW untuk referensi sesudah perubahan.

Contoh Penggunaan : Trigger After Delete

Berikut adalah contoh penggunaan trigger untuk event setelah penghapusan (**AFTER DELETE**) pada table "**tr_penjualan**" - database minimarket. Langkah yang akan kita lakukan adalah sebagai berikut :

1. Kita akan membuat satu table audit dengan nama "**tr_penjualan_hapus**" yang berisi row-row yang dihapus dari table "**tr_penjualan**" dengan tambahan dua field, yaitu tanggal penghapusan (**tgl_perubahan**) dan user MySQL yang melakukan hal tersebut (**nama_user**).

   Berikut adalah perintahnya :

```
USE minimarket;
CREATE TABLE `tr_penjualan_hapus` LIKE `tr_penjualan`;
ALTER TABLE `tr_penjualan_hapus` ADD
      (
      `tgl_perubahan` DATETIME,
      `nama_user` VARCHAR(200)
      );
```

2. Tahap berikutnya adalah membuat trigger yang akan melakukan populasi data yang dihapus dari "tr_penjualan" ke table "tr_penjualan_hapus".

   Berikut adalah perintahnya :

```
DELIMITER |
CREATE TRIGGER hapus_tr_penjualan AFTER DELETE
   ON tr_penjualan FOR EACH ROW
BEGIN
 INSERT INTO tr_penjualan_hapus
      (      tgl_transaksi,
             kode_cabang,
             kode_kasir,
             kode_item,
             kode_produk,
             jumlah_pembelian,
             tgl_perubahan,
             nama_user
      )
 VALUES
      (      OLD.tgl_transaksi,
             OLD.kode_cabang,
             OLD.kode_kasir,
             OLD.kode_item,
             OLD.kode_produk,
             OLD.jumlah_pembelian,
             SYSDATE(),
             CURRENT_USER
      );
END;
|
DELIMITER ;
```

3. Setelah trigger di atas kita buat, sekarang saatnya kita melakukan pengujian. Coba hapus tiga row data dari table "**tr_penjualan**" dan lihat efeknya di table "**tr_penjualan_hapus**".

   Jalankan perintah berikut :

```
DELETE FROM tr_penjualan LIMIT 3;
SELECT * FROM tr_penjualan_hapus;
```

4. 3 row yang dihapus telah "pindah" ke table "tr_penjualan_hapus" dengan tambahan informasi waktu penghapusan dan user yang menghapus.

5. Selesai.