Stored Procedure Stored Function dan Trigger

Modul Praktikum Sistem Manajemen Basis Data 2025

Tim Asisten
Sistem Manajemen
Basis Data 2025

Stored Procedure

Stored procedure adalah sekelompok perintah SQL yang telah disimpan dalam database MySQL. Mereka sering digunakan untuk mengeksekusi tugas-tugas kompleks secara berkala atau untuk memudahkan pengelolaan database.

Dengan menggunakan stored procedure, kita dapat menyimpan serangkaian perintah SQL, kondisi logika, dan perulangan di dalam database. Ini memungkinkan kita dapat mengeksekusi serangkaian tugas atau perhitungan dengan cara yang terstruktur dan efisien.

Keuntungan Stored Procedure

1. Kinerja yang Lebih Baik

   Stored Procedure dapat meningkatkan kinerja aplikasi karena mereka dapat disimpan dalam cache server database. Ini berarti ketika dipanggil, mereka tidak perlu dikompilasi lagi, mengurangi overhead pemrosesan.

2. Keamanan

   Stored Procedure dapat membantu meningkatkan keamanan database karena pengguna hanya perlu izin untuk menjalankan prosedur, bukan mengakses tabel langsung. Ini memungkinkan pengelola database untuk mengontrol dengan lebih tepat siapa yang dapat melakukan operasi apa di dalam database.

3. Pemeliharaan yang Lebih Mudah

   Dengan Stored Procedure, logika aplikasi terpusat di dalam database. Ini berarti jika ada perubahan dalam logika aplikasi, kita hanya perlu memperbarui prosedur terkait di database, bukan kode aplikasi itu sendiri.

4. Penggunaan Ulang Kode

   Stored Procedure dapat digunakan kembali di berbagai bagian aplikasi yang berbeda. Ini mengurangi duplikasi kode dan memudahkan pemeliharaan.

Syntax Stored Procedure

1. Membuat Procedure

```
CREATE PROCEDURE nama_prosedur (parameter)
      query…;
```

2. Memanggil Procedure

```
CALL nama_prosedur(parameter);
```

3. Menghapus Procedure

```
DROP PROCEDURE IF EXISTS nama_prosedur;
```

4. Melihat Daftar Procedure

```
SHOW PROCEDURE STATUS WHERE Db = 'Nama_Database';
```

Contoh Stored Procedure (Tanpa Parameter)

```
CREATE PROCEDURE getMenu()
    SELECT m.item_id, m.name, m.price, m.description, m.image,
c.category_name
    FROM menu_items m
   JOIN categories c ON m.category_id = c.category_id
   ORDER BY m.item_id ASC;
```

Compound Procedure

Dalam MySQL juga terdapat Compound-Statement yang merupakan sebuah blok program yang dapat berisi blok program lainnya seperti deklarasi variabel, condition, dll.

Stored Procedure

```
CREATE PROCEDURE
nama_prosedur(parameter)
code ;
```

Compound Procedure

```
DELIMITER $$
CREATE PROCEDURE
nama_prosedur()
BEGIN
    Code;
    Code;
END$$
```

Delimiter adalah karakter pemisah yang berfungsi untuk memberi tahu kepada mysql akhir dari sebuah statement. Secara default delimiter

menggunakan karakter semicolon (;) artinya apabila ada tanda ; maka mysql akan mengartikan akhir dari statement. Karena stored procedure tersusun dari blok program yang terdiri dari beberapa statement query maka delimiter yang digunakan adalah tanda $$, sehingga apabila ditemukan tanda $$ maka mysql akan mengartikan akhir dari statement.

Contoh Compound Procedure:

```
/* membuat procedure */
DELIMITER $$
CREATE PROCEDURE sp_lokal_variabel()

BEGIN
   /* deklarasi lokal variabel */
   DECLARE a INT DEFAULT 10;
   DECLARE b, c INT;

   /* inisialisasi lokal variabel */
   SET a = a + 100;
   SET b = 2;
   SET c = a + b;
  BEGIN
      /* deklarasi lokal variabel pada nested block */
      DECLARE c INT;
      SET c = 5;

        /* akan menampilkan nilai variabel c pada nested block */
      SELECT a, b, c;
   END;


END$$
DELIMITER ;
/* memanggil prosedur */
CALL sp_lokal_variabel();
```

hasil eksekusi:

a	b	c
110	2	5

Flow Control Statements

1. IF statement	`IF condition THEN statement`
	`  [ELSEIF condition THEN statement] ...`
	`  [ELSE statement]`
	`END IF`

2. Case Statement	`  -- cara penggunaan pertama`
	` CASE case_value`
	`  WHEN when_value THEN statement_list`
	`  [WHEN when_value THEN statement_list] ...`
	`  [ELSE statement_list]`
	`END CASE`
	
	`-- cara penggunaan kedua`
	`CASE`
	`  WHEN search_condition THEN statement_list`
	`  [WHEN search_condition THEN statement_list] ...`
	`  [ELSE statement_list]`
	`END CASE`

3. While Statement	`WHILE search_condition DO`
	`  statement_list`
	`END WHILE`

Selain itu terdapat statement lain seperti ***Iterate, Leave, Loop, dan Repeat.***

Procedure Parameter

Terdapat 3 jenis parameter procedure, yaitu:

1. IN

   Parameter input (IN) digunakan untuk mengirim nilai ke dalam stored procedure. Nilai parameter diinisialisasi di luar stored procedure, dan nilai tersebut akan digunakan oleh stored procedure selama eksekusi prosedur.

```
DELIMITER $$
CREATE PROCEDURE nama_prosedur(IN [par_name][par_type])
```

```
BEGIN
    code
END $$
```

2. OUT

Parameter output (OUT) digunakan untuk mengembalikan nilai dari stored procedure ke pemanggil stored procedure. Nilai parameter diinisialisasi di dalam stored procedure, dan nilai tersebut akan dikembalikan ke pemanggil stored procedure setelah stored procedure dieksekusi.

```
DELIMITER $$
CREATE PROCEDURE nama_prosedur(OUT [par_name][par_type])
BEGIN
    code
END $$
```

3. INOUT

Parameter input output (INOUT) digunakan untuk mengirim dan mengembalikan nilai ke pemanggil stored procedure. Nilai parameter diinisialisasi di luar stored procedure, dan nilai tersebut akan digunakan oleh stored procedure selama eksekusi. Setelah stored procedure dieksekusi, nilai parameter akan diperbarui dan dikembalikan ke pemanggil stored procedure

```
DELIMITER $$
CREATE PROCEDURE nama_prosedur(INOUT [par_name][par_type])
BEGIN
    code
END $$
```

Contoh Stored Procedure (Dengan Parameter)

1. Menetapkan contoh Parameter INOUT

```
SET @Info = 'Hasil Pencarian';
```

2. Membuat Prosedur

```
DELIMITER $$

CREATE PROCEDURE SP_FilterMenuByMaxHarga(
    IN m_harga DECIMAL(10,2),              -- IN: input dari pengguna
    OUT total_menu INT,                    -- OUT: jumlah menu yang
cocok
    INOUT info_text VARCHAR(255)           -- INOUT: input string awal
+ output tambahan
)
BEGIN
    -- Hitung jumlah menu yang harganya di bawah m_harga
    SELECT COUNT(*) INTO total_menu
    FROM menu_items
    WHERE price <= m_harga;

    -- Ubah nilai info_text dengan tambahan informasi
    SET info_text = CONCAT(info_text, ' | Terdapat ', total_menu, '
menu di bawah Rp', m_harga);

    -- Tampilkan menu hasil filter
    SELECT item_id, name, price
    FROM menu_items
    WHERE price <= m_harga
    ORDER BY price DESC;
END $$

DELIMITER ;
```

3. Memanggil Prosedur

```
CALL SP_FilterMenuByMaxHarga(30000, @total, @info);
```

Output yang dihasilkan

```
MariaDB [black_beans]> CALL SP_FilterMenuByMaxHarga(30000, @total, @info);
+---------+------------------+----------+
| item_id | name             | price    |
+---------+------------------+----------+
|       5 | Green Tea        | 28000.00 |
|       3 | Americano        | 27000.00 |
|       6 | Black Tea        | 25000.00 |
|      10 | Cheese Danish    | 24000.00 |
|       9 | Chocolate Muffin | 22000.00 |
|       8 | Croissant        | 20000.00 |
+---------+------------------+----------+
6 rows in set (0.002 sec)

Query OK, 1 row affected (0.024 sec)

MariaDB [black_beans]> SELECT @total AS 'Jumlah Menu', @info AS 'Informasi';
+-------------+-----------------------------------------------------------+
| Jumlah Menu | Informasi                                                 |
+-------------+-----------------------------------------------------------+
|           6 | Hasil Pencarian | Terdapat 6 menu di bawah Rp30000.00 |
+-------------+-----------------------------------------------------------+
1 row in set (0.000 sec)
```

Stored Function

Sebuah stored function adalah sebuah prosedur yang disimpan di dalam database yang dapat menerima parameter, melakukan operasi tertentu, dan mengembalikan nilai. Stored function biasanya digunakan dalam SQL untuk melakukan perhitungan atau manipulasi data yang kompleks secara efisien di dalam database.

Berbeda dengan stored procedure, stored function selalu mengembalikan sebuah nilai dan dapat digunakan dalam pernyataan SQL seperti SELECT, INSERT, UPDATE, dan DELETE. Stored function juga dapat dipanggil dari dalam pernyataan SQL lainnya atau dari aplikasi yang menggunakan database.

Keuntungan Menggunakan Stored Function

1. Mudah Digunakan di Dalam Query

   Stored Function dapat digunakan langsung dalam pernyataan SQL, yang memungkinkan kita untuk memproses data di dalam database tanpa perlu memanggilnya dari aplikasi.

2. Kemampuan untuk Mengembalikan Nilai

   Stored Function dapat mengembalikan nilai tunggal, sehingga dapat digunakan dalam ekspresi SQL untuk menghitung atau memanipulasi data.

Syntax Stored Function

1. Membuat Function

```
CREATE FUNCTION func_name(parameter)
RETURNS return_type
code ;
```

2. Memanggil Function

```
SELECT func_name(parameter);
```

3. Mengubah Function

```
CREATE OR REPLACE FUNCTION func_name(parameter)
code ;
```

4. Menghapus Function

```
DROP FUNCTION IF EXISTS func_name;
```

Contoh Stored Function (Dengan Parameter)

```
DELIMITER $$
CREATE FUNCTION FN_CekHargaByNama(nama_menu VARCHAR(100))
RETURNS VARCHAR(50)
BEGIN
    DECLARE harga DECIMAL(10,2);
    DECLARE pesan VARCHAR(50);

    -- Ambil harga dari menu_items berdasarkan nama menu
    SELECT price INTO harga
    FROM menu_items
    WHERE name = nama_menu
```

```
    LIMIT 1;

    -- Cek status harga
    IF harga > 25000 THEN
        SET pesan = 'Mahal';
    ELSE
        SET pesan = 'Terjangkau';
    END IF;

    RETURN pesan;
END $$

DELIMITER ;
```

Hasil Output:

```
MariaDB [black_beans]> SELECT FN_CekHargaByNama('Americano');
+---------------------------------+
| FN_CekHargaByNama('Americano') |
+---------------------------------+
| Mahal                           |
+---------------------------------+
1 row in set (0.001 sec)

MariaDB [black_beans]> SELECT FN_CekHargaByNama('Chocolate Muffin');
+---------------------------------------+
| FN_CekHargaByNama('Chocolate Muffin') |
+---------------------------------------+
| Terjangkau                            |
+---------------------------------------+
1 row in set (0.001 sec)
```

Trigger

Trigger adalah sepotong kode SQL yang secara otomatis dieksekusi oleh server database (MySQL) ketika suatu peristiwa tertentu terjadi di dalam tabel (seperti penyisipan, pembaruan, atau penghapusan data). Triger ini dapat digunakan untuk menerapkan logika bisnis tambahan, memvalidasi data, atau melakukan tindakan lainnya secara otomatis dan transparan ketika operasi tertentu dilakukan pada tabel yang terkait.

Fungsi Trigger

Fungsi dari trigger dalam basis data MySQL adalah untuk menanggapi atau merespons secara otomatis terhadap peristiwa yang terjadi pada tabel basis data. Ini memungkinkan kita untuk melakukan tindakan tertentu secara otomatis ketika peristiwa tertentu terjadi, tanpa perlu intervensi manual.

Berikut adalah beberapa fungsi utama dari trigger:

1. Menjaga Integritas Data

   Trigger dapat digunakan untuk memastikan bahwa data yang dimasukkan ke dalam tabel memenuhi persyaratan tertentu. Misalnya ketika diprogram untuk memeriksa apakah nilai yang dimasukkan ke dalam kolom tertentu adalah unik atau tidak null. Dengan menggunakan trigger, kita dapat memastikan bahwa data dalam tabel selalu dalam keadaan konsisten dan terstruktur dengan baik.

2. Mencegah Human Error

   Dalam situasi di mana pengguna memasukkan data secara manual ke dalam database, kesalahan manusia dapat terjadi. Trigger dapat digunakan untuk memastikan bahwa data yang dimasukkan ke dalam tabel memenuhi persyaratan tertentu, sehingga dapat membantu mencegah kesalahan manusia dan mengurangi peluang terjadinya kesalahan yang tidak diinginkan.

3. Mencegah Proses Transaksi yang Tidak Sah

   Trigger dapat digunakan untuk memeriksa apakah data yang dimasukkan ke dalam tabel sesuai dengan peraturan. Misalnya, trigger dapat diprogram untuk memeriksa apakah jumlah barang yang dimasukkan ke dalam database sesuai dengan jumlah yang dibayar. Dengan menggunakan trigger, kita dapat mencegah terjadinya transaksi yang tidak sah.

Struktur Trigger

```
DELIMITER $$
    CREATE TRIGGER nama_trigger
    /* time and event */
    {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
        ON nama_table
        FOR EACH ROW
    BEGIN
        /* query yang dijalankan saat trigger aktif */
        QUERY SQL;
        QUERY SQL;
END $$
```

Trigger Time & Event

Terdapat beberapa event yang bisa kita gunakan dalam mengeksekusi trigger berdasarkan kebutuhan.

Query	Keterangan
BEFORE INSERT	Trigger dijalankan sebelum data dimasukkan ke database
AFTER INSERT	Trigger dijalankan sesudah data dimasukkan ke database
BEFORE UPDATE	Trigger dijalankan sebelum data diubah di database
AFTER UPDATE	Trigger dijalankan sesudah data diubah di database
BEFORE DELETE	Trigger dijalankan sebelum data dihapus di database
AFTER DELETE	Trigger dijalankan sesudah data dihapus di database

Keyword

Selain event, terdapat juga keyword yang digunakan pada saat mengeksekusi trigger yang digunakan untuk mengambil data dalam tabel.

Keyword	Keterangan
Old	Mengambil data di tabel lama
New	Mengambil data di tabel baru

Contoh Implementasi

Membuat tabel Log Perubahan Harga Menu

1. Buat tabel Log

```
CREATE TABLE log_perubahan_harga (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    item_id INT,
    nama_menu VARCHAR(100),
    harga_lama DECIMAL(10,2),
    harga_baru DECIMAL(10,2),
    waktu_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. Buat trigger untuk tabel Log

```
DELIMITER $$

CREATE TRIGGER trg_log_harga_update
AFTER UPDATE ON menu_items
FOR EACH ROW
BEGIN
    IF OLD.price <> NEW.price THEN
        INSERT INTO log_perubahan_harga(item_id, nama_menu,
harga_lama, harga_baru)
        VALUES (OLD.item_id, OLD.name, OLD.price, NEW.price);
    END IF;
END $$
```

```
DELIMITER ;
```

3. Coba Update

```
UPDATE menu_items SET price = 30000.00 WHERE menu_items.item_id =
3;
```

Output

```
MariaDB [black_beans]> select * from log_perubahan_harga;
+--------+---------+-----------+------------+------------+---------------------+
| log_id | item_id | nama_menu | harga_lama | harga_baru | waktu_update        |
+--------+---------+-----------+------------+------------+---------------------+
|      1 |       3 | Americano |   27000.00 |   30000.00 | 2025-04-22 15:59:34 |
+--------+---------+-----------+------------+------------+---------------------+
1 row in set (0.000 sec)
```

Mengantisipasi Nilai NULL untuk Insert menu

1. Buat Trigger

```
DELIMITER $$

CREATE TRIGGER tr_menu_items_null
BEFORE INSERT ON menu_items
FOR EACH ROW
BEGIN
  -- Cek jika nama menu kosong
  IF NEW.name = '' OR NEW.name IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Nama menu tidak boleh kosong!';
  END IF;

  -- Cek jika harga menu kosong atau kurang dari 0
  IF NEW.price IS NULL OR NEW.price < 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Harga menu tidak boleh kosong atau
negatif!';
  END IF;

  -- Cek jika kategori menu kosong
  IF NEW.category_id IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Kategori menu tidak boleh kosong!';
  END IF;
```

```
  -- Cek jika deskripsi menu kosong
  IF NEW.description = '' OR NEW.description IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Deskripsi menu tidak boleh kosong!';
  END IF;

  -- Cek jika gambar menu kosong
  IF NEW.image = '' OR NEW.image IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Gambar menu tidak boleh kosong!';
  END IF;

END $$

DELIMITER ;
```

Output:

```
MariaDB [black_beans]> INSERT INTO menu_items (name, price, category_id, description, image)
    -> VALUES ('', 15000, 1, 'Menu yang lezat', 'gambar.jpg');
ERROR 1644 (45000): Nama menu tidak boleh kosong!
MariaDB [black_beans]>
```

Tugas Praktikum

LP 8 selamat mengerjakan love u all, semangat terus !!!!

LP 8 : STORED PROCEDURE STORED FUNCTION TRIGGER

DL : 8 MEI 2025 23:59.

Penutup

Kami mengucapkan terima kasih kepada Tuhan, Bangsa, dan Almamater. Yang telah berkontribusi dalam pembuatan modul ini. Semoga pengetahuan dan pengalaman yang diperoleh dapat memberikan manfaat yang signifikan bagi masa depan. Aamiin.

Referensi

Asisten Praktikum Sistem Basis Data (2024). Modul Sistem Basis Data : Modul 8 : Stored Procedure Stored Function dan Trigger