Mengenal MySQL *Stored Procedure*

Didik Setiawan
di2k.setiawan@gmail.com

Stored procedure memungkinkan sebuah aplikasi berbasis database (SQL) mendefinisikan dan kemudian memanggil prosedur melalui pernyataan SQL. *Stored procedure* dapat digunakan untuk aplikasi terdistribusi (*client server*) maupun aplikasi tidak terdistribusi.

Keuntungan besar dalam menggunakan *Stored procedure* pada aplikasi terdistribusi adalah dapat digunakannya satu perintah `CALL` pada aplikasi untuk mengerjakan perintah yang sama. Tersimpannya *Stored procedure* pada sistem database terpusat memungkinkan dilakukannya perubahan untuk menyesuaikan dengan perubahan sistem terkini dan dapat berlaku untuk keseluruh sistem aplikasi yang ada tanpa perlu dilakukan perubahan untuk setiap modul aplikasi.

Dengan menggunakan *Stored procedure*, SQL tidak akan melakukan *loading* seluruh tabel yang ter-relasi, tetapi langsung melakukan *filtering* berdasarkan *query* yang dimaksud sehingga dari sisi performa eksekusi, utilitas jaringan, dan keamanan dapat lebih terjaga.

Tulisan ini berisi pengenalan dan gambaran (deskripsi) tentang fungsionalitas *Stored Procedure* pada database MySQL.

Stored Procedure

A. Definisi

A stored procedure is a procedure (like a subprogram in a regular computing language) that is stored (in the database).

(Peter Gulutzan, 2006:5)

Stored Procedure dapat diartikan sebagai program ("*routines*") yang tersimpan dalam database seperti halnya data. MySQL mendukung dua jenis '*routines*', yaitu : *Stored Procedure* dan *functions* yang mengembalikan nilai.

Sebuah *Stored procedure* setidaknya memiliki nama, dapat mengandung parameter (ataupun tidak), dan berisi SQL *statement* yang dapat terdiri dari satu atau banyak SQL *statement* lain didalamnya. Fungsi baru yang terdapat pada MySQL *Stored Procedure* antara lain variabel lokal (*local variable*), penanganan kesalahan (*error handling*), kontrol alur (*loop control*), dan pengkondisian (*conditional*).

Format umum untuk membuat *Stored Procedure* adalah sebagai berikut :

```
CREATE PROCEDURE procedure1 /* name */
(IN parameter1 INTEGER) /* parameters */
BEGIN /* start of block *002F
   DECLARE variable1 CHAR(10); /* variables */
   IF parameter1 = 17 THEN /* start of IF */
      SET variable1 = 'birds'; /* assignment */
   ELSE
      SET variable1 = 'beasts'; /* assignment */
   END IF; /* end of IF */
   INSERT INTO table1 VALUES (variable1);/* statement */
END /* end of block */
```

MySQL mendukung *Stored Procedure* untuk versi 5.0 atau setelahnya. Untuk memeriksa apakah versi database MySQL yang digunakan telah mendukung Stored Procedure dapat dilakukan dengan cara memberikan perintah berikut :

```
mysql> SHOW VARIABLES LIKE 'version';
```

atau

```
mysql> SELECT VERSION();
```

B. Mengapa *Stored Procedure*

Berikut beberapa alasan mengapa *Stored Procedure* digunakan dalam sebuah sistem aplikasi :

a. Program lebih ringkas dan cepat, dengan *Stored procedure*, algoritma akan tersimpan di sisi server, program cukup memanggil *Stored procedure* yang diinginkan, selanjutnya server yang akan mengeksekusi perintah atau proses yang diinginkan;

b. *Stored procedure* bersifat *component*¸ dimana perubahan pada bahasa pemrograman di sisi aplikasi tidak akan mengubah logika dari sisi database. Perubahan pada proses berlaku untuk semua user yang terhubung ke database, bahkan saat program masih;

c. *Stored procedure* bersifat *Portable*, *Stored procedure* yang ditulis dalam SQL, akan dapat berjalan di semua jenis platform yang mendukung MySQL, tanpa harus melakukan instalasi *runtime* tambahan atau mengatur hak akses pada sistem operasi untuk menjalankan *Stored procedure* tersebut;

d. *Stored procedure* bersifat *migratory*, perintah SQL pada MySQL mengacu pada standar SQL:2003, sehingga memungkinkan untuk melakukan pemindahan *Stored procedure* ke database lain dengan sedikit perubahan.

C. *Delimiter*

Delimiter adalah karakter atau *string* yang digunakan untuk menyatakan akhir SQL *statement*, default *delimiter* pada MySQL adalah *semicolon* (;).

Perubahan *delimiter* diperlukan karena dalam sebuah *Stored Procedure* dapat terdiri dari lebih satu *statement* dan setiap akhir *statement* diakhiri dengan *delimiter*.

Tulisan ini akan menggunakan delimiter // untuk menyatakan akhir SQL *statement.* Perubahan *delimiter* dilakukan dengan menggunakan perintah :

```
mysql> DELIMITER //
```

untuk mengembalikan *delimiter* kembali menjadi *semicolon* (;) berikan perintah berikut :

```
mysql> DELIMITER ;
```

D. Ketentuan Umum

Terdapat beberapa ketentuan umum yang harus diperhatikan dalam membuat *procedure* di MySQL, diantaranya adalah :

a. Nama *procedure* bersifat tidak *case sensitive*, artinya *procedure* 'p1' adalah sama dengan *procedure* 'P1';

b. Nama *procedure* adalah terbatas, dapat berupa spasi dengan panjang maksimum adalah 64 karakter dan tidak diperkenankan menggunakan nama yang ada pada fungsi *built-in* MySQL;

c. MySQL tidak memperkenankan terjadinya *overloading*, kondisi dimana terdapat dua *procedure* dengan nama yang sama dalam satu database.

Struktur Perintah

A. Memulai Stored Procedure

Sebuah procedure dapat dibuat dengan perintah sebagaimana contoh berikut :

```
mysql> CREATE PROCEDURE p1 () SELECT * FROM t_sdana;//
Query OK, 0 rows affected (0.47 sec)
```

Dari perintah diatas tampak bahwa struktur pembentuk *Stored Procedure* di MySQL terdiri dari :

- SQL *statement* untuk membuat *procedure* : `CREATE PROCEDURE`;
- Nama *procedure* : p1;
- *Parameter list*, ditandai dengan (), diperlukan untuk *procedure* yang membutuhkan parameter;
- SQL *statement*, yang merupakan isi *procedure*, pada contoh diatas adalah perintah "`SELECT * FROM t_sdana;`".

SQL *statement* yang dapat digunakan pada bagian isi *procedure* meliputi :

- Standard SQL, seperti : `INSERT, UPDATE, DELETE, SELECT, DROP, CREATE, REPLACE, SET, COMMIT, ROLLBACK.`

- MySQL *Extra feature*, berupa *statement Data Definition Language* (DDL), seperti : `DROP table`.
- MySQL *Extension*, berupa *direct* select, Contoh : `SELECT 'A'`.

Adapun SQL *statement* yang tidak dapat digunakan pada bagian isi procedure adalah `CREATE PROCEDURE`, `ALTER PROCEDURE`, `DROP PROCEDURE`, `CREATE FUNCTION`, `CREATE TRIGGER`, `CREATE EVENT`, `USE database`, `LOAD DATA INFILE`, `LOCK TABLES`, `CHECK`.

Untuk memanggil atau menjalankan *procedure* dilakukan dengan memberikan perintah `CALL nama_procedure` diikuti dengan *delimiter*.

Menjalankan dan hasil *procedure* p1 adalah sebagai berikut :

```
mysql> CALL p1 //
+---------+---------+-------------------------+
| kdsdana | nmsdana | nmsdana2                |
+---------+---------+-------------------------+
| 01      | RM      | RUPIAH MURNI            |
| 02      | PLN     | PINJAMAN LUAR NEGERI    |
| 03      | RMP     | RUPIAH MURNI PENDAMPING |
| 04      | PNP     | PNBP                    |
| 05      | PDM     | PINJAMAN DALAM NEGERI   |
| 06      | BLU     | BADAN LAYANAN UMUM      |
| 07      | STM     | STIMULUS                |
| 08      | HDN     | HIBAH DALAM NEGERI      |
| 09      | HLN     | HIBAH LUAR NEGERI       |
+---------+---------+-------------------------+
9 rows in set (0.00 sec)
Query OK, 0 rows affected (0.05 sec)
```

perintah diatas adalah untuk memanggil *procedure* `p1` yang berisi perintah : `select * from t_sdana`.

B. *Parameter*

Penggunaan *parameter* pada *procedure* di MySQL dapat dibedakan sebagai berikut :

a. `CREATE PROCEDURE p1 ()`

   Procedure tanpa *parameter*. Sebagaimana contoh *procedure* p1.

b. `CREATE PROCEDURE p2 ([IN] name data-type)`

   Procedure dengan parameter input. Contoh :

```
mysql> CREATE PROCEDURE p2(p INT) SET @x = p //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL p2(12345) //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @x //
+-------+
| @x    |
+-------+
| 12345 |
+-------+
1 row in set (0.00 sec)
```

Procedure p2 membuat sebuah variable `x` yang berisi nilai sesuai dengan nilai yang ada pada *parameter* p.

c. `CREATE PROCEDURE p3 (OUT name data-type)`

Procedure dengan *parameter output*. Contoh :

```
mysql> CREATE PROCEDURE p3(OUT p INT) SET p = -5 //
Query OK, 0 rows affected (0.03 sec)
mysql> CALL p3(@y) //
Query OK, 0 rows affected (0.05 sec)
mysql> SELECT @y //
+------+
| @y   |
+------+
|   -5 |
+------+
1 row in set (0.00 sec)
```

Procedure p3 membuat sebuah *variable* yang bernilai minus 5, saat *procedure* dijalankan perintah `OUT` meminta DBMS untuk mengeluarkan nilai yang ada pada procedure dan disimpan pada variable `@y`.

d. `CREATE PROCEDURE p4 (INOUT name data-type)`

Procedure dengan *parameter intput* dan *output*. Contoh :

```
mysql> CREATE PROCEDURE p4()
-> BEGIN
->    DECLARE a, b INT;
->    SET @a = 5;
->    SET @b = 7;
->    SELECT @a+@b;
-> END; //
Query OK, 0 rows affected (0.03 sec)
mysql> CALL p4() //
+-------+
| @a+@b |
+-------+
|    12 |
+-------+
1 row in set (0.00 sec)
```

Procedure p4 menginisialisasi *variable* `a` dan `b` yang akan menghasilkan nilai saat *procedure* dipanggil berupa hasil penjumlahan `variable a` dan `variable b`.

C. Kontrol Alur

Terdapat dua macam kontrol alur (*control flow*) yang dapat digunakan pada *Stored procedure* di MySQL, yaitu pengkondisian (*conditional*) dan Perulangan (*looping*).

A. *Conditional*

Conditional merupakan suatu pengaturan alur program berdasar kondisi *Boolean* yang dijadikan patokan.

1. `IF - THEN - ELSE`

Kontrol alur (*control flow*) yang didasarkan pada nilai *Boolean* hasil sebuah ekspresi yang bernilai `true` akan menjalankan blok pernyataan yang ada. Dalam kondisi ini dapat dilakukan dengan menggunakan perintah `IF … THEN`.

Jika terdapat blok pernyataan lain yang akan dijalankan pada saat nilai *Boolean* tidak tercapai dapat menggunakan `IF … THEN … ELSE`.

Contoh :

```
mysql> CREATE PROCEDURE p5 (IN par1 INT)
-> BEGIN
->    DECLARE var1 INT;
->    SET var1 = par1 + 1;
->    IF var1 = 1 THEN
->        SELECT var1 ;
->    END IF;
->    IF par1 = 0 THEN
->        SELECT par1;
->    ELSE
->        SELECT par1 + 1;
->    END IF;
-> END;//
Query OK, 0 rows affected (0.00 sec)
mysql> CALL P5(0) //
+------+
| var1 |
+------+
|    1 |
+------+
1 row in set (0.03 sec)
+------+
| par1 |
+------+
|    0 |
+------+
1 row in set (0.04 sec)
Query OK, 0 rows affected (0.05 sec)
mysql> CALL P5(1) //
+----------+
| par1 + 1 |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

Nilai var1 akan ditampilkan jika nilai *parameter* adalah 0, selain 0 maka var1 tidak ditampilkan dan akan menampilkan par1 + 1.

2. CASE

Fungsi yang sama dengan perintah `IF` untuk menyatakan pengkondisian adalah `CASE`. Penggunaan `case` data dilihat pada contoh berikut :

```
mysql> CREATE PROCEDURE p6 (IN par1 INT)
    -> BEGIN
    ->    DECLARE var1 INT;
    ->    SET var1 = par1 + 1;
    ->    CASE var1
    ->       WHEN 1 THEN SELECT var1;
    ->       WHEN 2 THEN SELECT var1 + 1;
    ->       ELSE SELECT var1 + 99 ;
    ->    END CASE;
    -> END;//
Query OK, 0 rows affected (0.59 sec)

mysql> CALL p6(0)//
+------+
| var1 |
+------+
| 1    |
+------+
1 row in set (0.28 sec)
Query OK, 0 rows affected (0.31 sec)

mysql> CALL p6(2)//
+-----------+
| var1 + 99 |
+-----------+
|       102 |
+-----------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

Procedure p6 akan menghasilkan nilai *parameter* + 1 untuk data masukan 0 atau 1 dan akan menghasilkan nilai *parameter* + 99 untuk nilai *parameter* selain 0 dan 1.

`Case` dapat digunakan untuk menggantikan pengkondisian `IF` terutama untuk pilihan kondisi dalam jumlah lebih dari dua kondisi.

B. Perulangan (*Looping*)

Looping adalah perulangan suatu blok *procedure* berdasarkan kondisi yang ditentukan sampai tercapai kondisi untuk menghentikannya (*terminasi*). Setiap perulangan memiliki empat bagian yaitu : *Inisialisasi*, proses, *iterasi*, *terminasi*.

Inisialisasi untuk menyiapkan keadaan awal perulangan. Proses adalah pernyataan yang akan diulangi. *Iterasi* adalah bagian yang dijalankan setelah proses, tetapi sebelum proses tersebut

diulangi. *Terminasi* adalah pernyataan *Boolean* yang diperiksa setiap kali selama perulangan untuk melihat apakah sebuah iterasi sudah saatnya akan dihentikan.

Terdapat tiga standar perulangan dalam *Stored Procedure* MySQL, yaitu : `while … end while`, `repeat … end repeat`, dan `loop … end loop`.

a. `WHILE – END WHILE`

Berikut contoh penggunaan perintah `while … end while` dalam *procedure* MySQL :

```
mysql> CREATE PROCEDURE p7()
    -> BEGIN
    ->    DECLARE v INT;
    ->    SET v = 0;
    ->    WHILE v < 3 DO
    ->       SELECT v;
    ->       SET v = v + 1;
    ->    END WHILE;
    -> END;
    -> //
Query OK, 0 rows affected (0.31 sec)
mysql> CALL p7()//
+------+
| v    |
+------+
|    0 |
+------+
1 row in set (0.05 sec)
+------+
| v    |
+------+
|    1 |
+------+
1 row in set (0.06 sec)
+------+
| v    |
+------+
|    2 |
+------+
1 row in set (0.06 sec)
```

Pada procedure diatas *variable* v didefinisikan dan diinisialisasi dengan nilai 0 selanjutnya di lakukan *iterasi* hingga kondisi *variable* kurang dari 3 tercapai.

b. `REPEAT … END REPEAT`

Berikut contoh penggunaan perintah `repeat … end repeat` dalam *procedure* MySQL :

```
mysql> CREATE PROCEDURE p8()
    -> BEGIN
    ->    DECLARE v INT;
    ->    SET v = 0;
    ->    REPEAT
    ->       SELECT v;
    ->       SET v = v + 1;
    ->       UNTIL v >= 3
   ->     END REPEAT;
   -> END;//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CALL p8() //
+-------+
| v     |
+-------+
|     0 |
+-------+
1 row in set (0.00 sec)
+-------+
| v     |
+-------+
|     1 |
+-------+
1 row in set (0.01 sec)
+-------+
| v     |
+-------+
|     2 |
+-------+
1 row in set (0.01 sec)
```

Hasil pemanggilan *procedure* p8 adalah sama dengan procedure p7, *iterasi* dilakukan hingga kondisi *variable* v mencapai nilai lebih atau sama dengan 3, jika kondisi tercapai maka proses akan diakhiri.

Yang harus dicermati dalam penggunaan perintah `repeat … end repeat` adalah pada baris pernyataan `UNTIL` yang tidak menggunakan tanda *semicolon* (;) sebagai tanda akhir sebuah baris pernyataan.

c. `LOOP … END LOOP`

Berikut contoh penggunaan perintah `loop … end loop` dalam *procedure* MySQL :

```
mysql> CREATE PROCEDURE p9()
    -> BEGIN
    ->    DECLARE v INT;
    ->    SET v = 0;
    ->    loop_label: LOOP
    ->       SELECT v;
    ->       SET v = v + 1;
    ->       IF v >= 3 THEN
    ->          LEAVE loop_label;
    ->       END IF;
    ->    END LOOP;
    -> END;//
Query OK, 0 rows affected (0.03 sec)

mysql> CALL p9()//
+-------+
| v     |
+-------+
|     0 |
+-------+
1 row in set (0.00 sec)
+-------+
| v     |
+-------+
|     1 |
+-------+
1 row in set (0.01 sec)
+-------+
| v     |
```

```
+------+
|    2 |
+------+
1 row in set (0.01 sec)
Query OK, 0 rows affected (0.02 sec)
```

Perulangan dengan menggunakan `loop … end loop` mensyaratkan penggunaan kondisi `IF` untuk proses terminasi dan pernyataan `LEAVE` untuk menyatakan keluar dari proses perulangan. Terdapat pula perintah `ITERATE` yang berfungsi untuk melakukan perulangan (*iterasi*) yang berarti proses `loop` akan diulang kembali.

C. Penanganan Kesalahan (*Error Handling*)

Penanganan kesalahan atau *error handling* dalam *Stored Procedure* dapat digunakan untuk melakukan pencatatan (*log*) proses saat sebuah proses gagal dijalankan.

Contoh : saat proses `INSERT` terhadap sebuah tabel gagal dijalankan maka proses yang gagal dijalankan, waktu proses dan alasan kegagalan proses dapat dicatat dalam sebuah tabel tersendiri. Berikut ini adalah contoh kegagalan proses *insert* pada tabel t3 karena adanya kesalahan `constraint`.

```
mysql> CREATE TABLE t2
s1 INT, PRIMARY KEY (s1))
engine=innodb;//
mysql> CREATE TABLE t3 (s1 INT, KEY (s1),FOREIGN KEY (s1)
REFERENCES t2 (s1)) engine=innodb;//
mysql> INSERT INTO t3 VALUES (5);//
...
ERROR 1216 (23000): Cannot add or update a child row: a
foreign key constraint fails
```

Kegagalan proses akan disimpan pada tabel berikut :

```
CREATE TABLE error_log (error_message CHAR(80))//
```

Adapun procedure yang dapat digunakan adalah sebagai berikut :

```
CREATE PROCEDURE p10 (parameter1 INT)
BEGIN
   DECLARE EXIT HANDLER FOR 1216
   INSERT INTO error_log VALUES (CONCAT('Time: ',current_date,
   '. Foreign Key Reference Failure For Value = ',parameter1));
   INSERT INTO t3 VALUES (parameter1);
END;//
```

Saat terjadi kegagalan `INSERT` data pada tabel t3 dengan pesan kesalahan `ERROR 1216` maka kegagalan tersebut akan dicatat pada tabel `error_log`.

MySQL Cursor

Cursors dapat didefinisikan sebagai fungsionalitas untuk menyimpan hasil dari sebuah *query* ke sebuah *variabel*, kemudian mengeluarkannya untuk menjadi filter di *query* yang lain atau untuk kebutuhan lain.

MySQL memiliki beberapa perintah pembentuk *Cursors*, yaitu :

```
DECLARE cursor-name CURSOR FOR SELECT ...;
```

Digunakan untuk mendeklarasikan atau membentuk sebuah *cursors*.

```
OPEN cursor-name;
```

Digunakan untuk membuka sebuah *cursors*.

```
FETCH cursor-name INTO variable [, variable];
```

Digunakan untuk memasukkan nilai *cursors* pada sebuah *variable* yang telah di definisikan sebelumnya.

```
CLOSE cursor-name;
```

Digunakan untuk menutup *cursors*.

Berikut contoh penggunaan *cursors* pada sebuah *procedure* :

```
CREATE PROCEDURE p11 (OUT return_val INT)
BEGIN
   DECLARE a,b INT;
   DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
   DECLARE CONTINUE HANDLER FOR NOT FOUND
   SET b = 1;
   OPEN cur_1;
   REPEAT
      FETCH cur_1 INTO a;
      UNTIL b = 1
   END REPEAT;
   CLOSE cur_1;
   SET return_val = a;
END;//

mysql> CALL p11(@return_val)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @return_val//
+-------------+
| @return_val |
+-------------+
|           5 |
+-------------+
1 row in set (0.00 sec)
```

Cursor pada MySQL memiliki karakteristik sebagai berikut :

a. *Read Only*, MySQL versi 5 hanya memungkinkan untuk mengambil data dari *cursors*, tidak dapat dilakukan proses update data.

   Perintah berikut tidak berlaku pada *cursors* :

```
FETCH cursor1 INTO variable1;
UPDATE t1 SET column1 = 'value1' WHERE CURRENT OF cursor1;
```

b. *Not Scrollable*, data pada sebuah *cursors* hanya dapat dibaca berdasarkan urutan, tidak dimungkinkan untuk memanggil data berdasarkan urutan tertentu atau membaca kembali data yang telah terlewati sebelumnya.

   Perintah berikut tidak berlaku pada cursors :

```
FETCH PRIOR cursor1 INTO variable1;
FETCH ABSOLUTE 55 cursor1 INTO variable1;
```

c. *Asensitive*, proses update data pada sebuah tabel harus dihindari saat tabel tersebut digunakan untuk membentuk sebuah *cursors.*

Dynamic SQL

Terdapat sebuah fitur baru yang dapat digunakan untuk melakukan perubahan pada sebuah *string* yang mengandung SQL *statement* yaitu *Dynamic SQL*. Fitur *Dynamic* SQL memungkinkan dilakukannya proses `PREPARE` dan `EXECUTE` pada sebuah *statement*.

Berikut contoh *Stored Procedur* yang menggunakan *Dynamic* SQL :

```
mysql> CREATE PROCEDURE p12 (search_value INT)
    -> BEGIN
    ->     SET @sql = CONCAT(
    ->     'SELECT * FROM t WHERE s1 = ',search_value);
    ->     PREPARE stmt1 FROM @sql;
    ->     EXECUTE stmt1;
    -> END; //
Query OK, 0 rows affected (0.01 sec)
mysql> CALL p12(0)//
+------+
| s1   |
+------+
|    0 |
+------+
1 row in set (0.00 sec)
```

Secara teori seluruh *statement* dan *algoritma* dapat digunakan di *Dynamic* SQL, adapun batasan yang mungkin terjadi adalah :

1. Terdapat *statement* yang mungkin tidak atau belum dipersiapkan (*Preparable*);
2. Penurunan kecepatan proses, karena MySQL harus mem-*parsing statement* yang dibuat;
3. Sulit untuk melakukan uji coba dan *debugging*.

Metadata

Metadata berfungsi untuk melihat informasi dimana *Stored Procedure* disimpan. Untuk melihat metadata sebuah *Stored Procedure* dapat menggunakan `SHOW` *statement* dan `SELECT` *statement*. Berikut beberapa perintah yang dapat digunakan untuk menampilkan *metadata* pada MySQL *Stored Procedure.*

1. `SHOW CREATE PROCEDURE`

   Hasil `Show Create Procedure` dapat menunjukkan perintah yang digunakan saat membuat sebuah *Stored Procedure*.

```
mysql> show create procedure p6//
+-----------+----------+-------------------+
| Procedure | sql_mode | Create Procedure  |
+-----------+----------+-------------------+
| p6        |          | CREATE PROCEDURE  |
|           |          | `db5`.`p6`(out p  |
|           |          | int) set p = -5   |
+-----------+----------+-------------------+
1 row in set (0.00 sec)
```

2. `SHOW PROCEDURE STATUS`

   Digunakan untuk menunjukkan status sebuah *Stored Procedure*.

```
mysql> SHOW PROCEDURE STATUS LIKE 'p6'//
+------+------+-----------+----------------+
| Db   | Name | Type      | Definer        | ...
+------+------+-----------+----------------+
| db5  | p6   | PROCEDURE | root@localhost | ...
+------+------+-----------+----------------+
1 row in set (0.01 sec)
```

3. `SELECT FROM MYSQL.PROC`

   Hasil `Select From mysql.proc` hampir identik dengan hasil `Show Procedure Status`.

```
SELECT * FROM mysql.proc WHERE name = 'p6'//
+------+------+-----------+---------------+
| db   | name | type      | specific_name | ...
+------+------+-----------+---------------+
| db5  | p6   | PROCEDURE | p6            | ...
+------+------+-----------+---------------+
1 row in set (0.00 sec)
```

4. `SELECT FROM INFORMATION_SCHEMA`

Untuk melihat *metadata* sebuah *Stored Procedure* terlebih dahulu harus diketahui urutan `routine_shema` *Stored Procedure* bersangkutan.

Tabel `information_schema` berisi informasi sebagai berikut :

```
mysql> SELECT TABLE_NAME, COLUMN_NAME, COLUMN_TYPE FROM
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = 'ROUTINES';//
+------------+--------------------+---------------+
| TABLE_NAME | COLUMN_NAME        | COLUMN_TYPE   |
+------------+--------------------+---------------+
| ROUTINES   | SPECIFIC_NAME      | varchar(64)   |
| ROUTINES   | ROUTINE_CATALOG    | longtext      |
| ROUTINES   | ROUTINE_SCHEMA     | varchar(64)   |
| ROUTINES   | ROUTINE_NAME       | varchar(64)   |
| ROUTINES   | ROUTINE_TYPE       | varchar(9)    |
| ROUTINES   | DTD_IDENTIFIER     | varchar(64)   |
| ROUTINES   | ROUTINE_BODY       | varchar(8)    |
| ROUTINES   | ROUTINE_DEFINITION | longtext      |
| ROUTINES   | EXTERNAL_NAME      | varchar(64)   |
| ROUTINES   | EXTERNAL_LANGUAGE  | varchar(64)   |
| ROUTINES   | PARAMETER_STYLE    | varchar(8)    |
| ROUTINES   | IS_DETERMINISTIC   | varchar(3)    |
| ROUTINES   | SQL_DATA_ACCESS    | varchar(64)   |
| ROUTINES   | SQL_PATH           | varchar(64)   |
| ROUTINES   | SECURITY_TYPE      | varchar(7)    |
| ROUTINES   | CREATED            | varbinary(19) |
| ROUTINES   | LAST_ALTERED       | varbinary(19) |
| ROUTINES   | SQL_MODE           | longtext      |
| ROUTINES   | ROUTINE_COMMENT    | varchar(64)   |
| ROUTINES   | DEFINER            | varchar(77)   |
+------------+--------------------+---------------+
20 rows in set (0.01 sec)
```

Untuk mengetahui berapa banyak *Stored Procedure* pada sebuah database dapat dilakukan dengan perintah berikut :

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.ROUTINES
    -> WHERE ROUTINE_SCHEMA = 'db5';//
+----------+
| COUNT(*) |
+----------+
|       28 |
+----------+
1 row in set (0.02 sec)
```

`ROUTINE_DEFINITION` dalam `INFORMATION_SCHEMA.ROUTINES` berisi *procedure* atau *function* yang hanya dapat dilihat oleh admin, pembuat atau oleh user yang memiliki kewenangan.

Fungsi `SHOW PROCEDURE STATUS [WHERE condition];` memiliki parameter kondisi sebagaimana kondisi pada perintah `SELECT`, contoh :

```
mysql> SHOW PROCEDURE STATUS WHERE Db = 'db6';//
Empty set (0.03 sec)

mysql> SHOW PROCEDURE STATUS WHERE ROUTINE_NAME = 'p1';//
+------+------+-----------+----------------+
| Db   | Name | Type      | Definer        | ...
+------+------+-----------+----------------+
| db5  | p    | PROCEDURE | root@localhost | ...
+------+------+-----------+----------------+
1 row in set (0.00 sec)
```

Parameter ini bermanfaat untuk membatasi informasi status *procedure* yang akan ditampilkan.

Semoga tulisan ini dapat bermanfaat bagi kita semua, terima kasih.

Referensi

1. http://www.mysqltutorial.org;
2. Peter Gulutzan, "*MySQL Stored Procedures*", MySQL AB, Edmonton, Canada, 2006.

Biografi Penulis

Didik Setiawan
Pranata Komputer Kementerian Keuangan RI