MODUL VII
STORED PROCEDURE

A. TUJUAN

- ✓ Memahami konsep dasar stored procedure, kelebihan dan kekurangannya.
- ✓ Memahami implementasi stored procedure di dalam basis data.
- ✓ Mampu menyelesaikan operasi-operasi data spesifik dengan memanfaatkan stored procedure.

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. Stored Procedure

Stored procedure adalah sebuah prosedur—layaknya subprogram (subrutin) di dalam bahasa pemrograman reguler—yang tersimpan di dalam katalog basis data.

Beberapa kelebihan yang ditawarkan stored procedure antara lain: meningkatkan performa, mereduksi trafik jaringan, *reusable*, dan meningkatkan kontrol sekuriti.

Di balik kelebihan-kelebihannya, stored procedure juga memiliki kekurangan, di antaranya: berpotensi meningkatkan beban server dan penulisannya tidak mudah (memerlukan pengetahuan spesifik).

Sintaks stored procedure diperlihatkan sebagai berikut:

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body
```

Untuk memanggil stored procedure, digunakan perintah `CALL` (beberapa DBMS ada yang menggunakan `EXECUTE`).

```
CALL sp_name
```

Dalam implementasi nyata, penggunaan stored procedure sering melibatkan parameter. Di MySQL, parameter stored procedure dibedakan menjadi tiga mode: `IN`, `OUT`, dan `INOUT`.

- IN

  Parameter yang merupakan mode default ini mengindikasikan bahwa sebuah parameter dapat di-*pass* ke stored procedure tetapi nilainya tidak dapat diubah (dari dalam stored procedure).

- OUT

  Mode ini mengindikasikan bahwa stored procedure dapat mengubah parameter dan mengirimkan kembali ke program pemanggil.

- INOUT

  Mode ini pada dasarnya merupakan kombinasi dari mode `IN` dan `OUT`.

Sintaks pendefinisian parameter diperlihatkan sebagai berikut:

```
MODE param_name param_type(param_size)
```

Stored procedure dapat mencerminkan beragam operasi data, misalnya seleksi, penambahan, pengubahan, penghapusan, dan juga operasi-operasi DDL.

Seperti halnya prosedur di bahasa pemrograman, stored procedure juga dapat melibatkan variabel, pernyataan kondisional, dan pengulangan.

D. LATIHAN

1. Stored Procedure

Seperti halnya tabel, stored procedure diciptakan dengan menggunakan perintah `CREATE`. Sebagai contoh, buat stored procedure `getMahasiswa()` untuk menampilkan semua data mahasiswa.

1. Ketikkan pernyataan pembuatan stored procedure berikut di editor teks.

```
DELIMITER //
CREATE PROCEDURE getMahasiswa()

 BEGIN

   /* Ini baris komentar */
   SELECT * FROM mahasiswa;

 END //

DELIMITER ;
```

Perintah `DELIMITER` digunakan untuk mengubah delimiter standar, misalnya di sini dari titik koma (**;**) menjadi slash ganda (**//**). Langkah

ini umumnya dilakukan ketika isi stored procedure mengandung titik koma—yang merupakan delimiter standar di SQL.

Pernyataan di antara `BEGIN` dan `END` merupakan badan (*body*) stored procedure.

Perintah `DELIMITER` di akhir baris digunakan untuk mengembalikan delimiter ke karakter semula.

2. Eksekusi file stored procedure (sesuaikan path lokasi penyimpanan file).
3. Setelah tahap pembuatan berhasil, panggil stored procedure `getMahasiswa()`.

```
mysql> CALL getMahasiswa();
+-----+------+---------------+--------------+
| nim | nama | jenis_kelamin | alamat       |
+-----+------+---------------+--------------+
| 101 | Arif | L             | Jl. Kenangan |
| 102 | Budi | L             | Jl. Jombang  |
| 103 | Wati | P             | Jl. Surabaya |
| 104 | Ika  | P             | Jl. Jombang  |
| 105 | Tono | L             | Jl. Jakarta  |
| 106 | Iwan | L             | Jl. Bandung  |
| 107 | Sari | P             | Jl. Malang   |
+-----+------+---------------+--------------+
7 rows in set (0.00 sec)
```

Untuk mendapatkan informasi mengenai status stored procedure, gunakan perintah `SHOW PROCEDURE STATUS`.

```
mysql> SHOW PROCEDURE STATUS;
```

Seperti di tabel, kita juga bisa mendapatkan informasi pembuatan stored procedure.

```
mysql> SHOW CREATE PROCEDURE getMahasiswa;
```

Untuk menghapus stored procedure, gunakan perintah `DROP PROCEDURE`.

```
mysql> DROP PROCEDURE getMahasiswa;
Query OK, 0 rows affected (0.00 sec)
```

2. Parameter IN

Stored procedure di contoh sebelumnya memperlihatkan bentuk default (tanpa parameter). Di sini kita juga bisa mendefinisikan parameter yang nantinya dapat digunakan oleh pernyataan di body stored procedure.

Sebagai contoh, kita bisa mendapatkan semua data matakuliah di semester tertentu.

```
DELIMITER //

CREATE PROCEDURE getMhsBySemester(IN smt INT(2))

  BEGIN
    SELECT *
```

```
    FROM matakuliah
    WHERE semester = smt;
  END //

DELIMITER ;
```

Untuk memanggil stored procedure yang memiliki parameter, maka kita harus menspesifikasikan argumennya. Misalkan kita ingin mendapatkan data matakuliah di semester 3.

```
mysql> CALL getMhsBySemester(3);
+---------+----------------------+-----+----------+----------+
| kode_mk | nama_mk              | sks | semester | kode_dos |
+---------+----------------------+-----+----------+----------+
| PTI447  | Praktikum Basis Data |   1 |        3 |       11 |
| PTI777  | Sistem Informasi     |   2 |        3 |       99 |
| TIK342  | Praktikum Basis Data |   1 |        3 |       11 |
+---------+----------------------+-----+----------+----------+
3 rows in set (0.00 sec)
```

Apabila pemanggilan stored procedure di atas mengabaikan argumen, DBMS akan merespon dengan pesan kesalahan.

Bergantung kebutuhan, pendefinisian parameter pada stored procedure juga bisa lebih dari satu. Sebagai contoh, buat stored procedure dengan dua buah parameter seperti berikut:

```
DELIMITER //

CREATE PROCEDURE getMhsBySemSks(IN smt INT(2), IN s INT(2))

  BEGIN

    SELECT *
    FROM matakuliah
    WHERE semester = smt
    AND sks = s;

  END //

DELIMITER ;
```

Pemanggilan stored procedure di atas tentunya akan memerlukan dua buah argumen.

```
mysql> CALL getMhsBySemSks(3, 2);
+---------+-----------------+-----+----------+----------+
| kode_mk | nama_mk         | sks | semester | kode_dos |
+---------+-----------------+-----+----------+----------+
| PTI777  | Sistem Informasi |   2 |        3 |       99 |
+---------+-----------------+-----+----------+----------+
1 row in set (0.00 sec)
```

Variabel

Di MySQL, kita juga bisa mendeklarasikan variabel global—ruang lingkup session—dengan menggunakan perintah `SET` dan notasi `@`. Sebagai contoh, perintah berikut akan mendeklarasikan variabel bernama **smt** dan diinisialisasi dengan nilai **3**.

```
mysql> SET @smt = 3;
Query OK, 0 rows affected (0.00 sec)
```

Untuk memeriksa nilai variabel, gunakan perintah `SELECT`.

```
mysql> SELECT @smt;
+------+
| @smt |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

Langkah selanjutnya, kita bisa memanfaatkan variabel—yang telah dideklarasikan—untuk operasi-operasi lain, misalnya sebagai argumen stored procedure.

```
mysql> CALL getMhsBySemester(@smt);
+---------+----------------------+-----+----------+----------+
| kode_mk | nama_mk              | sks | semester | kode_dos |
+---------+----------------------+-----+----------+----------+
| PTI447  | Praktikum Basis Data |   1 |        3 |       11 |
| PTI777  | Sistem Informasi     |   2 |        3 |       99 |
| TIK342  | Praktikum Basis Data |   1 |        3 |       11 |
+---------+----------------------+-----+----------+----------+
3 rows in set (0.00 sec)
```

Penambahan Data

Pada operasi penambahan, data-data terkait diisikan melalui argumen. Selanjutnya, isi stored procedure tinggal memasukkan data ke tabel.

Contoh berikut memperlihatkan stored procedure untuk penambahan data di tabel jurusan.

```
DELIMITER //

CREATE PROCEDURE addJurusan(
  IN kode VARCHAR(2),
  IN nama VARCHAR(30),
  IN dos INT(3)
)

  BEGIN

    INSERT INTO jurusan
    VALUES(kode, nama, dos);

  END //

DELIMITER ;
```

Contoh eksekusi stored procedure penambahan data.

```
mysql> SELECT * FROM jurusan;
+----------+---------------+----------+
| kode_jur | nama_jur      | kode_dos |
+----------+---------------+----------+
| TE       | Teknik ELektro |       10 |
| TM       | Teknik Mesin   |       13 |
| TS       | Teknik Sipil   |       23 |
+----------+---------------+----------+
3 rows in set (0.00 sec)
```

```
mysql> CALL addJurusan('TG', 'Teknik Geodesi', 9);
Query OK, 1 row affected (0.17 sec)

mysql> SELECT * FROM jurusan;
+----------+----------------+----------+
| kode_jur | nama_jur       | kode_dos |
+----------+----------------+----------+
| TE       | Teknik ELektro |       10 |
| TM       | Teknik Mesin   |       13 |
| TS       | Teknik Sipil   |       23 |
| TG       | Teknik Geodesi |        9 |
+----------+----------------+----------+
4 rows in set (0.00 sec)
```

Operasi-operasi manipulasi data lainnya bisa Anda coba sendiri, dan tak jauh beda dengan pernyataan SQL reguler.

3. Parameter OUT

Dalam konteks bahasa pemrograman, parameter `OUT` analog dengan *passing-by-reference*. Dengan demikian, parameter ini nilainya bisa diubah oleh stored procedure.

```
DELIMITER //

CREATE PROCEDURE countMK(OUT total INT(2))

  BEGIN

    SELECT COUNT(kode_mk)

    /* Hasil counting di-pass back ke variabel total */
    INTO total
    FROM matakuliah;

  END //

DELIMITER ;
```

Untuk mengeksekusi stored procedure dengan parameter `OUT`, kita harus menspesifikasikan argumennya.

```
mysql> CALL countMK(@total);
Query OK, 0 rows affected (0.00 sec)
```

Perhatikan, argumen harus menggunakan notasi `@`, yang mengindikasikan sebagai suatu parameter `OUT`.

Langkah selanjutnya, untuk mendapatkan nilai variabel, gunakan pernyataan `SELECT`.

```
mysql> SELECT @total AS total_mk;
+----------+
| total_mk |
+----------+
| 7        |
+----------+
1 row in set (0.00 sec)
```

Parameter mode `OUT` juga bisa dikombinasikan dengan mode `IN` (akan dijelaskan nanti).

4. Parameter INOUT

Pada parameter dengan mode ini, kita bisa mengirimkan parameter ke stored procedure dan mendapatkan nilai kembalian yang baru.

Sebagai contoh, buat stored procedure seperti berikut:

```
DELIMITER //

CREATE PROCEDURE countBySex(INOUT arg VARCHAR(5))

  BEGIN

    SELECT COUNT(nim)
    INTO arg
    FROM mahasiswa
    WHERE jenis_kelamin = arg;

  END //

DELIMITER ;
```

Contoh penggunaannya, misal untuk mendapatkan jumlah mahasiswa yang jenis kelaminnya L.

```
mysql> SET @var = 'L';
Query OK, 0 rows affected (0.00 sec)

mysql> CALL countBySex(@var);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @var;
+------+
| @var |
+------+
| 4    |
+------+
1 row in set (0.00 sec)
```

Pendekatan `INOUT` juga bisa direpresentasikan dalam bentuk `IN` dan `OUT` secara terpisah.

```
DELIMITER //

CREATE PROCEDURE countBySex2(IN sx VARCHAR(1), OUT total INT(5))

  BEGIN

    SELECT COUNT(nim)
    INTO total
    FROM mahasiswa
    WHERE jenis_kelamin = sx;

  END //

DELIMITER ;
```

Contoh penggunaannya:

```
mysql> CALL countBySex2('L', @total);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @total;
+--------+
| @total |
+--------+
| 4      |
+--------+
1 row in set (0.00 sec)
```

5. Pencabangan dan Pengulangan

Penggunaan pernyataan-pernyataan pencabangan ataupun pengulangan di dalam stored procedure merupakan tindakan yang legal. Dengan demikian, kita bisa menghasilkan suatu prosedur yang kompleks.

Contoh berikut memperlihatkan penggunaan pernyataan IF.

```
DELIMITER //

CREATE PROCEDURE demoIF(IN bil INT(3))

  BEGIN

    -- Deklarasi variabel di dalam stored procedure
    DECLARE str  VARCHAR(30);

    IF (bil > 0) THEN
      SET str = 'Lebih dari Nol';
    ELSE
      SET str = 'Kurang dari / sama dengan Nol';
    END IF;

    -- Mencetak output ke layar
    SELECT str;

  END //

DELIMITER ;
```

Contoh penggunaan:

```
mysql> CALL demoIF(3);
+----------------+
| str            |
+----------------+
| Lebih dari Nol |
+----------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL demoIF(-5);
+------------------------------+
| str                          |
+------------------------------+
| Kurang dari / sama dengan Nol |
+------------------------------+
```

```
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Contoh implementasi pernyataan perulangan diperlihatkan sebagai berikut:

```
DELIMITER //

CREATE PROCEDURE demoLoop(IN bil INT(3))

  BEGIN

    -- Deklarasi variabel
    DECLARE x INT;
    DECLARE str VARCHAR(50);

    SET x = 1;
    SET str = '';

    WHILE x <= bil DO

      SET  str = CONCAT(str, x, ', ');

      -- Inkremen
      SET  x = x + 1;

    END WHILE;

      -- Mencetak output ke layar
      SELECT str;

  END //

DELIMITER ;
```

Contoh eksekusi stored procedure perulangan.

```
mysql> CALL demoLoop(9);
+----------------------------+
| str                        |
+----------------------------+
| 1, 2, 3, 4, 5, 6, 7, 8, 9,  |
+----------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

E. TUGAS PRAKTIKUM

1. Definisikan stored procedure untuk mendapatkan banyaknya matakuliah dan jumlah total sks dengan melibatkan dua buah parameter `OUT`. (**Point 20**)

```
mysql> SELECT @jml AS jumlah_mk, @total AS total_sks;
+-----------+-----------+
| jumlah_mk | total_sks |
+-----------+-----------+
| 7         | 15        |
+-----------+-----------+
1 row in set (0.00 sec)
```

2. Definisikan stored procedure untuk penambahan data ambil_mk. Skenarionya, penambahan dapat dilakukan ***jika dan hanya jika*** nilai nim eksis di tabel mahasiswa dan nilai kode_mk eksis di tabel matakuliah. Apabila operasi berhasil, kembalikan status "**OK**"; sebaliknya jika gagal, kembalikan pesan "**Operasi Gagal**". (**Point: 40**)

3. Definisikan stored procedure untuk memodifikasi data dosen apabila eksis dan melakukan penambahan jika belum eksis. Jadi, masukan dari argumen dapat digunakan untuk penambahan ataupun modifikasi data. (**Point: 50**)

"Terkadang batas antara keberhasilan dan kegagalan itu sangat tipis sekali;
yang membatasi keduanya adalah kesabaran" (anonim)