# 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 getMatakuliahBySemester(IN smt INT(3))
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.

```
MariaDB [zakaria_140533601726]> CALL getMatakuliahBySemester(3);
+---------+---------------------+------+----------+----------+
| kode_mk | nama_mk             | sks  | semester | kode_dos |
+---------+---------------------+------+----------+----------+
| PTI447  | Praktikum Basisdata |    1 |        3 | 11       |
| PTI777  | Sistem Informasi    |    2 |        3 | 99       |
| TIK342  | Praktikum Basisdata |    1 |        3 | 11       |
+---------+---------------------+------+----------+----------+
3 rows in set (0.09 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 getMatakuliahBySemSks(
    IN Smt INT(3),
    IN Sks INT(3)
)
BEGIN
    SELECT *
    FROM matakuliah
    WHERE semester=Smt
    AND sks=Sks;
END //
DELIMITER ;
```

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

```
MariaDB [zakaria_140533601726]> CALL getMatakuliahBySemSks(3,2);
+---------+---------------------+------+----------+----------+
| kode_mk | nama_mk             | sks  | semester | kode_dos |
+---------+---------------------+------+----------+----------+
| PTI447  | Praktikum Basisdata |    1 |        3 | 11       |
| PTI777  | Sistem Informasi    |    2 |        3 | 99       |
| TIK342  | Praktikum Basisdata |    1 |        3 | 11       |
+---------+---------------------+------+----------+----------+
3 rows 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.**

```
MariaDB [zakaria_140533601726]> SET @smt=3;
Query OK, 0 rows affected (0.00 sec)
```

Untuk memeriksa nilai variabel, gunakan perintah `SELECT`.

```
MariaDB [zakaria_140533601726]> 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.

```
MariaDB [zakaria_140533601726]> CALL getMatakuliahBySemester(@smt);
+---------+---------------------+------+----------+----------+
| kode_mk | nama_mk             | sks  | semester | kode_dos |
+---------+---------------------+------+----------+----------+
| PTI447  | Praktikum Basisdata |    1 |        3 | 11       |
| PTI777  | Sistem Informasi    |    2 |        3 | 99       |
| TIK342  | Praktikum Basisdata |    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 dosen.

```
DELIMITER //
CREATE PROCEDURE AddDosen(
    IN kode_dos VARCHAR(10),
    IN nama_dos VARCHAR(50),
    IN alamat_dos VARCHAR(100)
)
BEGIN
    INSERT INTO dosen VALUES(
    kode_dos,nama_dos,alamat_dos
    );
END //
DELIMITER ;
```

```
MariaDB [zakaria_140533601726]> CALL AddDosen('212','Gunawan','Jl. Ambarawa');
Query OK, 1 row affected (0.09 sec)

MariaDB [zakaria_140533601726]> SELECT * FROM dosen;
+----------+-----------+----------------+
| kode_dos | nama_dos  | alamat_dos     |
+----------+-----------+----------------+
| 10       | Soeharto  | Jl. Jombang    |
| 11       | Martono   | Jl. Kalpataru  |
| 12       | Rahmawati | Jl. Jakarta    |
| 13       | Bambang   | Jl. Bandung    |
| 14       | Nurul     | Jl. Raya Tidar |
| 212      | Gunawan   | Jl. Ambarawa   |
+----------+-----------+----------------+
6 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 JumlahDosen(
    OUT jumlah_dos INT(3)
)
BEGIN
    SELECT COUNT(kode_does)
    INTO jumlah_dos
    /*
    Hasil Count di-return ke
    variabel jumlah_dos
    */
    FROM dosen;
END //
DELIMITER ;
```

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

```
MariaDB [zakaria_140533601726]> CALL JumlahDosen(@jumlah_dos);
Query OK, 1 row affected (0.00 sec)
```

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

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

```
MariaDB [zakaria_140533601726]> SELECT @jumlah_dos;
+-------------+
| @jumlah_dos |
+-------------+
|           6 |
+-------------+
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 CountBySks(
    INOUT var INT(3)
)
BEGIN
    SELECT COUNT(kode_mk)
    INTO var
    FROM matakuliah
    WHERE sks=var;
END //
DELIMITER ;
```

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

```
MariaDB [zakaria_140533601726]> SET @SKS=2;
Query OK, 0 rows affected (0.00 sec)

MariaDB [zakaria_140533601726]> CALL CountBySks(@SKS);
Query OK, 1 row affected (0.00 sec)

MariaDB [zakaria_140533601726]> SELECT @SKS;
+------+
| @SKS |
+------+
|    3 |
+------+
1 row in set (0.00 sec)
```

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

```
DELIMITER //
CREATE PROCEDURE CountByGender(
    IN gender VARCHAR(3),
    OUT total INT(3)
)
BEGIN
    SELECT COUNT(nim)
    INTO total
    FROM mahasiswa
    WHERE jenis_kelamin=gender;
END //
DELIMITER ;
```

Contoh penggunaannya:

```
MariaDB [zakaria_140533601726]> CALL CountByGender('L',@total);
Query OK, 1 row affected (0.00 sec)

MariaDB [zakaria_140533601726]> 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 */
    DECLARE str VARCHAR(50);
    IF(bil < 0) THEN
        SET str='Bilangan Negatif';
    ELSE
        SET str='Bilangan Positif';
    END IF;

    SELECT str;
END //
DELIMITER ;
```

Contoh penggunaan:

```
MariaDB [zakaria_140533601726]> CALL DemoIF(7);
+------------------+
| str              |
+------------------+
| Bilangan Positif |
+------------------+
1 row in set (0.00 sec)

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

Contoh penggunaan looping adalah sebagai berikut.

```
DELIMITER //
CREATE PROCEDURE DemoLoop(
    IN bil INT(3)
)
BEGIN
    /* Deklarasi variabel */
    DECLARE str VARCHAR(200);
    DECLARE i INT(3);
    SET i=1;
    SET str='';

    WHILE i <= bil DO
        SET str=CONCAT(str, i,',');
        SET i=i+1;
    END WHILE;

    SELECT str;
END //
DELIMITER ;
```

Contoh eksekusi stored procedure perulangan.

```
MariaDB [zakaria_140533601726]> 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 (0.02 sec)
```

## E. TUGAS PRAKTIKUM

1. Definisikan stored procedure untuk mendapatkan jumlah mahasiswa yang diajar oleh seorang dosen dengan kode dosen sesuai dengan parameter yang dimasukkan (parameter IN) . Jika tidak ada mahasiswa yang diajar, maka akan menampilkan angka 0 (**bukan null**), sehingga seperti tampilan berikut.

```
MariaDB [zakaria_140533601726]> CALL GetCountMhs('11');
+--------+
| jumlah |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

MariaDB [zakaria_140533601726]> CALL GetCountMhs('14');
+--------+
| jumlah |
+--------+
|      0 |
+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.02 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**".

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.

## F. TUGAS RUMAH

**Buatlah tabel dengan ketentuan sebagai berikut.**

**Deskripsi tabel kategori_pembayaran**

```
MariaDB [zakaria_140533601726]> desc kategori_pembayaran;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id_kat   | varchar(5)  | NO   | PRI | NULL    |       |
| nama_kat | varchar(30) | YES  |     | NULL    |       |
| nominal  | double      | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

**Isi data tabel kategori_pembayaran**

```
+--------+---------------------+---------+
| id_kat | nama_kat            | nominal |
+--------+---------------------+---------+
| 1      | Pembayaran KPL      | 200000  |
| 2      | Pembayaran Yudisium | 500000  |
| 3      | Pembayaran Wisuda   | 400000  |
+--------+---------------------+---------+
```

**Deskripsi tabel transaksi_pembayaran**

```
MariaDB [zakaria_140533601726]> desc transaksi_pembayaran;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| nim    | varchar(12) | YES  |     | NULL    |       |
| id_kat | varchar(5)  | YES  |     | NULL    |       |
| waktu  | date        | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

**Isi data tabel transaksi_pembayaran**

```
+------+--------+------------+
| nim  | id_kat | waktu      |
+------+--------+------------+
| 102  | 3      | 2016-01-09 |
| 101  | 2      | 2016-01-29 |
| 104  | 1      | 2016-03-09 |
| 104  | 2      | 2016-01-09 |
| 105  | 2      | 2016-01-11 |
| 105  | 3      | 2016-02-01 |
| 105  | 1      | 2016-01-19 |
| 107  | 1      | 2016-02-17 |
| 107  | 3      | 2016-02-17 |
+------+--------+------------+
```

1. Buatlah *Stored Procedure* untuk mendapatkan total nominal dari pembayaran setiap jenis kategori selama 1 bulan! Bulan pembayaran ditentukan melalui parameter saat procedure dipanggil.
   Contoh: Total pembayaran pada bulan Februari.

```
MariaDB [zakaria_140533601726]> CALL TotalPerkategori(2);
+--------------------+--------+
| nama_kat           | Total  |
+--------------------+--------+
| Pembayaran KPL     | 200000 |
| Pembayaran Wisuda  | 800000 |
+--------------------+--------+
2 rows in set (0.06 sec)
```

Bulan Februari

2. Buatlah *Stored Procedure* untuk memeriksa tunggakan yang dimiliki mahasiswa! Jika mahasiswa memiliki tunggakan maka akan menampilkan data kategori pembayaran yang belum dibayarkan, jika mahasiswa sudah membayar semua jenis kategori pembayaran maka akan menampilkan pesan bahwa mahasiswa tersebut tidak memiliki tunggakan.
   Contoh hasil eksekusi procedure.

```
MariaDB [zakaria_140533601726]> CALL CekTunggakan('102');
+--------+---------------------+---------+
| id_kat | nama_kat            | nominal |
+--------+---------------------+---------+
| 1      | Pembayaran KPL      |  200000 |
| 2      | Pembayaran Yudisium |  500000 |
+--------+---------------------+---------+
2 rows in set (0.03 sec)

Query OK, 0 rows affected (0.05 sec)

MariaDB [zakaria_140533601726]> CALL CekTunggakan('105');
+---------------------+
| pesan               |
+---------------------+
| TIDAK ADA TUNGGAKAN |
+---------------------+
1 row in set (0.00 sec)

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

NIM

NIM

**"Hidup itu pilihan, berpikirlah yang matang akan pilihan yang kau ambil □"**