Modul Praktikum Sistem Basis Data (MySQL)

Book · May 2016

CITATIONS
0

READS
22,882

1 author:

Muhammad Yunus
Politeknik Negeri Jember
15 PUBLICATIONS **2** CITATIONS

SEE PROFILE

MODUL PRAKTIKUM

SISTEM BASIS DATA

Versi 2.2

Disusun Oleh :
Muh. Yunus, M.Kom.
NIDN. 0831128607

S1 TEKNIK INFORMATIKA

STMIK BUMIGORA MATARAM

TAHUN 2016

LEMBAR PENGESAHAN

MODUL PRAKTIKUM

Mata Kuliah Praktikum	: Sistem Basis Data
Kode Mata Kuliah Praktikum	: TSKK220214
SKS	: 1
Program Studi	: S1 Teknik Informatika
Semester	: II (Genap)
Versi	: 2.2 (Edisi Revisi)

Mataram, 09-05-2016

Kepala Lab. Komputer,	Dosen Pengampu,
Muhammad Yunus, M.Kom. NIK. 10.6.135	Muhammad Yunus, M.Kom. NIK. 10.6.135
Mengetahui, Pembantu Ketua I,	Menyetujui, Ketua Prodi S1 Teknik Informatika
Khasnur Hidjah, S.Kom.,M.Cs. NIP. 197202072005012001	Galih Hendro Martono, M.Eng. NIK. 14.6.194

Kata Pengantar

Segala puji bagi Allah SWT yang berkat rahmat dan karuniaNya sehingga penyusun dapat menyelesaikan penulisan modul praktikum Sistem Basis Data ini tepat pada waktunya. Shalawat serta salam semoga senantiasa tercurah kepada junjungan alam, nabi besar Muhammad SAW yang telah memberikan contoh tauladan yang utama bagi kita semua.

Modul praktikum ini merupakan salah satu bahan ajar pendukung untuk memperkuat mata kuliah Sistem Basis Data yang diajarkan di kelas teori. Dengan adanya modul ini diharapkan mahasiswa/praktikan dapat dengan mudah mempelajari, memahami, dan mempraktikkan materi – materi yang telah diajarkan pada kelas teori mata kuliah Sistem Basis Data.

Selanjutnya juga diharapkan modul ini bisa membantu atau menjadi referensi dalam pemecahan permasalahan umum di luar materi perkuliahan. Sebagian besar isi dari modul praktikum ini merupakan rangkuman dari sumber-sumber yang telah dibuat penulis lain. Penyusun berharap agar modul ini dapat bermanfaat bagi semua kalangan pembaca.

Terima kasih untuk semuanya yang telah memberikan banyak kritik dan saran serta dukungan dalam penyusunan modul praktikum ini.

Mataram, Mei 2016

Penyusun

DAFTAR ISI

MODUL I
Pengenalan MySQL
Pert. 1 (1x170 menit)

Tujuan :

Setelah menyelesaikan modul ini, anda diharapkan dapat :

1. Mengenal lingkungan kerja MySQL
2. Mengenal format perintah di MySQL
3. Mengenal perintah-perintah sederhana di MySQL

Dasar Teori

MySQL adalah suatu perangkat lunak database relasi (Relational Database Management System atau RDBMS), seperti halnya ORACLE, Postgresql, MS SQL, dan sebagainya. MySQL dikembangkan sekitar tahun 1994 oleh sebuah perusahaan pengembang software dan konsultan database bernama MYSQL AB yang berada di Swedia. Waktu itu perusahaan tersebut masih bernama TcX DataKonsult AB, dan tujuan awal dikembangkannya MySQL adalah untuk mengembangkan aplikasi berbasis web pada client. MySQL menyebut produknya sebagai database open source terpopuler di dunia. Berdasarkan riset dinyatakan bahwa bahwa di platform Web, dan baik untuk kategori open source maupun umum, MySQL adalah database yang paling banyak dipakai. Menurut perusahaan pengembangnya, MySQL telah terpasang di sekitar 3 juta komputer. Puluhan hingga ratusan ribu situs mengandalkan MySQL bekerja siang malam memompa data bagi para pengunjungnya.

Format Perintah

Berikut adalah ketentuan-ketentuan memberi perintah pada MySQL:

- Setiap perintah harus diakhiri dengan tanda titik koma, kecuali untuk perintah tertentu, misalnya : quit
- Setiap perintah akan disimpan dalam buffer (memori sementara) untuk menyimpan histori perintah-perintah yang pernah diberikan.
- Perintah dapat berupa perintah SQL atau perintah khusus MySQL.
- Perintah-perintah dalam lingkungan MySQL tidak menerapkan aturan *case sensitive*, tetapi *case insensitive* yaitu perintah bisa dituliskan dalam huruf besar atau pun huruf kecil.

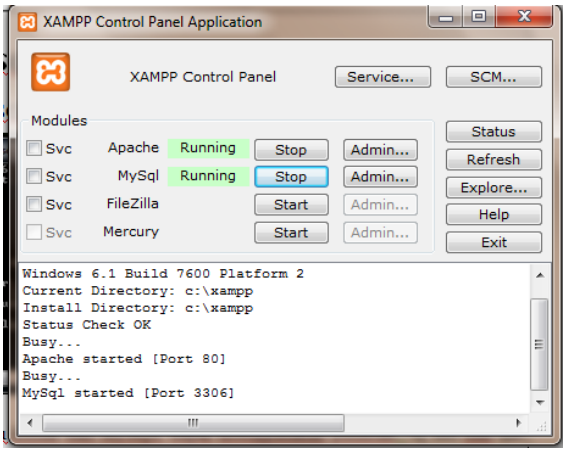
- Aturan *case sensitive* diterapkan pada penamaan objek-objek dalam database seperti nama database atau nama table, namun aturan ini hanya ada dalam lingkungan Unix dan Linux.

Ada beberapa tanda yang sering muncul di prompt :

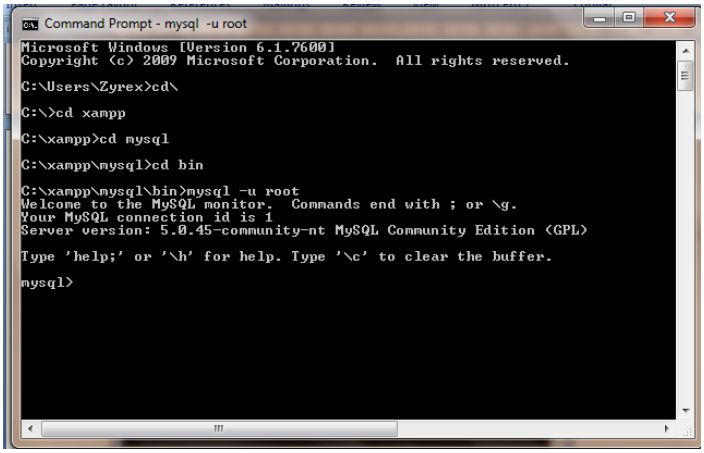
Prompt	Arti
mysql>	Siap menerima perintah baru
->	Menunggu baris berikut untuk perintah yang lebih dari satu baris
'>	Menunggu baris berikut, menunggu penutup string yang dimulai dengan tanda kutip satu (“'”)
">	Menunggu baris berikut, menunggu penutup string yang dimulai dengan tanda kutip dua (“"”)
`>	Menunggu baris berikutnya, menunggu penutup identifier yang dimulai dengan tanda backtick (“`”)

Start dan Stop MySQL

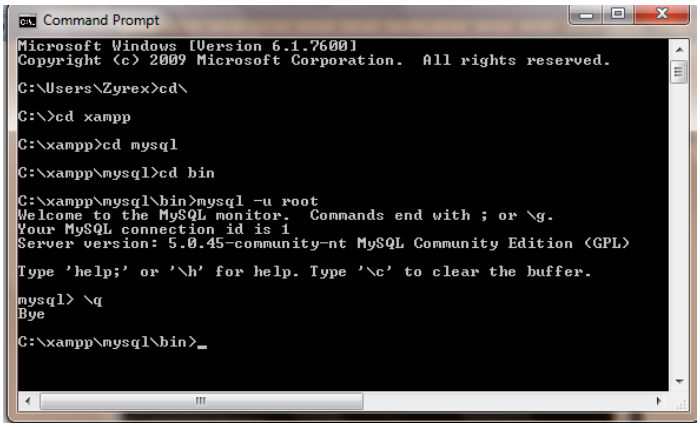
Berikut cara memulai MySQL dengan menggunakan bantuan XAMPP. Aktifkan Xampp Control Panel Application, klik start apache dan mysql.



Aktifkan command prompt, lalu ketik seperti gambar berikut:



Sedangkan untuk stop atau keluar dari MySQL dapat menggunakan perintah : \q, exit dan quit.



MODUL II
Data Definition Language (DDL)
Pert. 1 (1x170 menit)

Tujuan :

Setelah menyelesaikan modul ini, anda diharapkan dapat :

1. Membuat database dan tabel dengan data definition language
2. Mampu memodifikasi tabel

Dasar Teori

DDL (Data Definition Language), DDL merupakan kelompok perintah yang berfungsi untuk mendefinisikan atribut-atribut basis data, tabel, atribut(kolom), batasan-batasan terhadap suatu atribut, serta hubungan antar tabel. Yang termasuk dalam kelompok DDL ini adalah CREATE, ALTER, dan DROP.

a. Syntax Membuat Database : CREATE DATABASE namadatabase;

Namadatabase tidak boleh mengandung spasi dan tidak boleh memiliki nama yang sama antar database. Berikut ini perintah untuk membuat database dengan nama rental : CREATE DATABASE CV_SEJAHTERA;

```
mysql> create database CV_SEJAHTERA;
Query OK, 1 row affected (0.01 sec)
```

Syntax tambahan untuk menampilkan daftar nama database yang ada pada mysql menggunakan perintah : SHOW DATABASES;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| akademik           |
| akademik_t         |
| belajar_ajah       |
| belajarif          |
| cdcol              |
| contohlooping      |
| cv_sejahtera       |
| data_mahasiswa     |
| dataku             |
| dataku2            |
| db_akademik        |
```

b. Memilih Database : USE namadatabase;

Sebelum membuat suatu tabel, terlebih dahulu harus memilih salah satu database sebagai database aktif yang akan digunakan untuk menyimpan tabel-tabel, Berikut ini perintah untuk menggunakan database dengan nama CV_SEJAHTERA:

USE SEJAHTERA;

```
mysql> use CV_SEJAHTERA;
Database changed
```

c. Syntax Menghapus Database : DROP DATABASE namadatabase;

Database yang akan dihapus sesuai dengan namadatabase. Berikut ini perintah untuk menghapus database dengan nama rental : DROP DATABASE RENTAL;

d. Membuat Tabel : CREATE TABLE namatabel2 (Field1 TipeData1,Field2 TipeData2);

Nama tabel tidak boleh mengandung spasi (space). Field1 dan TipeData1 merupakan nama kolom pertama dan tipe data untuk kolom pertama. Jika ingin membuat tabel dengan kolom lebih dari satu, maka setelah pendefinisian tipe data sebelumnya diberikan tanda koma (,).

Berikut ini perintah untuk membuat tabel dengan nama barang :

```
mysql> create table barang(KDBARANG char(3) primary key,
    -> NAMA_BARANG varchar(20),
    -> SATUAN varchar(10),
    -> HARGA int(11));
Query OK, 0 rows affected (2.70 sec)
```

e. Menampilkan Tabel

Untuk menampilkan daftar nama tabel yang ada pada database yang sedang aktif/digunakan (dalam hal ini database rental) : SHOW TABLES;

```
mysql> show tables;
+-------------------------+
| Tables_in_cv_sejahtera |
+-------------------------+
| barang                  |
+-------------------------+
1 row in set (0.00 sec)
```

f. Menampilkan Atribut Tabel : DESC namatabel;

Untuk menampilkan deskripsi tabel (dalam hal ini jenisfilm) syntaxnya adalah : DESC barang;

```
mysql> desc barang;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| KDBARANG    | char(3)     | NO   | PRI |         |       |
| NAMA_BARANG | varchar(20) | YES  |     | NULL    |       |
| SATUAN      | varchar(10) | YES  |     | NULL    |       |
| HARGA       | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.05 sec)
```

g. Syntax Menghapus Tabel : DROP TABLE namatabel;

Tabel yang akan dihapus sesuai dengan namatabel, berikut ini perintah untuk menghapus tabel dengan nama jenisfilm : DROP TABLE BARANG;

h. Mendefinisikan Null/Not Null : CREATE TABLE namatabel (Field1 TipeData1 NOT NULL, Field2 TipeData2);

i. Mendefinisikan Primary Key Pada Tabel

Terdapat tiga cara untuk mendefinisikan primary key. Berikut ini adalah Syntax mendefinisikan primary key untuk Field1

```
CREATE TABLE namatabel(Field1 TipeData1 NOT NULL PRIMARY KEY, Field2 TipeData2);
```

Atau

```
CREATE TABLE namatabel ( Field1 TipeData1, Field2 TipeData2, PRIMARY KEY(Field1));
```

Atau

```
ALTER TABLE namatabel ADD CONSTRAINT namaconstraint PRIMARY KEY (namakolom);
```

j. Menghapus Primary Key Pada Tabel

Cara 1 : Jika primary key dibuat dengan menggunakan alter table :

```
ALTER TABLE namatabel DROP CONSTRAINT namaconstraint;
```

Cara 2 : Jika primary key dibuat melalui create table :

```
ALTER TABLE namatabel DROP PRIMARY KEY;
```

k. Menambah Kolom Baru Pada Tabel : `ALTER TABLE namatabel ADD fieldbaru tipe;`

Namatabel adalah nama tabel yang akan ditambah fieldnya. Fieldbaru adalah nama kolom yang akan ditambahkan, tipe adalah tipe data dari kolom yang akan ditambahkan.

Berikut ini contoh perintah untuk menambah kolom keterangan dengan tipe data varchar(25):

```
ALTER TABLE JENISFILM ADD KETERANGAN VARCHAR(25);
```

Untuk meletakkan field diawal, tambahkan sintaks first :

```
ALTER TABLE PELANGAN ADD COLUMN KODE CHAR(5) FIRST;
```

Untuk menyisipkan field setelah field tertentu, tambahkan sintaks after :

```
ALTER TABLE PELANGAN ADD COLUMN PHONE CHAR(5) AFTER ALAMAT;
```

l. Mengubah Tipe Data atau Lebar Kolom Pada Tabel : `ALTER TABLE NAMATABEL MODIFY COLUMN FIELD TIPE`

Namatabel adalah nama tabel yang akan diubah tipe data atau lebar kolomnya. Field adalah kolom yang akan diubah tipe data atau lebarnya. Tipe adalah tipe data baru atau tipe data lama dengan lebar kolom yang berbeda. Berikut ini contoh perintah untuk mengubah tipe data untuk kolom keterangan dengan char(20) :

```
ALTER TABLE JENISFILM MODIFY COLUMN KETERANGAN VARCHAR(20);
```

m. Mengubah Nama Kolom : `ALTER TABLE namatabel CHANGE COLUMN namalamakolom namabarukolom tipedatabaru;`

Namatabel adalah nama tabel yang akan diubah nama kolomnya, namalamakolom adalah kolom yang akan diganti namanya, namabarukolom adalah nama baru kolom, tipedatanya adalah tipe data dari kolom tersebut. Berikut ini contoh perintah untuk mengubah nama kolom keterangan menjadi ket :

```
ALTER TABLE JENISFILM CHANGE COLUMN KETERANGAN KET VARCHAR(20);
```

n. Menghapus Kolom Pada Tabel : `ALTER TABLE namatabel DROP COLUMN namakolom;`

Praktik !

1. Buat sebuah database dengan nama coba !
2. Buat sebuah tabel dengan nama mahasiswa di dalam database coba !
3. Tambahkan sebuah kolom : keterangan (varchar 15), sebagai kolom terakhir !
4. Tambahkan kolom nim (int 11) di awal (sebagai kolom pertama) !
5. Sisipkan sebuah kolom dengan nama phone (varchar 15) setelah kolom alamat varchar(15) !
6. Ubah kolom nim menjadi char(11) !
7. Ubah nama kolom phone menjadi telepon (varchar 20) !
8. Hapus kolom keterangan dari tabel !
9. Ganti nama tabel menjadi student!
10. Jadikan nim sebagai primary key !

Evaluasi dan Pertanyaan

1. Tulis semua perintah-perintah SQL percobaan di atas beserta outputnya !
2. Apa kegunaan dari index di tabel ?
3. Apa maksud dari int (11) ?
4. Ketika kita melihat struktur tabel dengan perintah desc, ada kolom Null yang berisi Yes dan No. Apa maksudnya ?

Kesimpulan

TULISKAN JAWABAN ANDA
PADA LEMBAR KERJA PRAKTIKUM (LKP)

MODUL III
DML dan Retrieve Data (Bagian I)
Pert. 2 (1x170 menit)

Tujuan :

Setelah menyelesaikan modul ini, Anda diharapkan dapat :

1. Mengenal data *manipulation language* dan mampu menggunakannya
2. Mampu mengelola *record* dan retrieve data

Dasar Teori

DML (Data Manipulation Language) DML adalah kelompok perintah yang berfungsi untuk memanipulasi data dalam basis data, misalnya untuk pengambilan, penyisipan, pengubahan dan penghapusan data. Perintah yang termasuk dalah kategori DML adalah : INSERT, DELETE, UPDATE dan SELECT.

1. I N S E R T

Perintah INSERT digunakan untuk menambahkan baris pada suatu tabel. Terdapat dua cara untuk menambah baris, yaitu:

Cara 1: Menambah baris dengan mengisi data pada setiap kolom :

INSERT INTO namatabel VALUES (nilai1,nilai2,nilai-n);

```
mysql> insert into barang values('B01','BUKU TULIS','LUSIN',50000),
    -> ('B02','PULPEN','LUSIN',80000),
    -> ('B03','PENGHAPUS','LUSIN',20000);
Query OK, 3 rows affected (2.61 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Cara 2 : Menambah baris dengan hanya mengisi data pada kolom tertentu :

INSERT INTO namatabel (kolom1,kolom2,kolom-n) VALUES (nilai1,nilai2,nilai-n);

```
mysql> insert into barang (KDBARANG,NAMA_BARANG,SATUAN,HARGA)
    -> values('B04','PERAUT','LUSIN',45000);
Query OK, 1 row affected (0.00 sec)
```

Keterangan :

Jika data bertipe string, date atau time (contoh : action, horor, 2007-11-10) maka pemberian nilainya diapit dengan tanda petik tunggal ('horor') atau petik ganda ("horor"). Jika data bertipe numerik (2500, 400) maka pemberian nilainya tidak diapit tanda petik tunggal maupun ganda.

2. DELETE

Perintah DELETE digunakan untuk menghapus satu baris, baris dengan kondisi tertentu atau seluruh baris. Syntax : DELETE FROM namatabel [WHERE kondisi];

Perintah dalam tanda [] bersifat opsional untuk menghapus suatu baris dengan suatu kondisi tertentu.

3. UPDATE

Perintah UPDATE digunakan untuk mengubah isi data pada satu atau beberapa kolom pada suatu table. Syntax :

UPDATE namatabel SET kolom1 = nilai1, kolom2 = nilai2 [WHERE kondisi];

```
mysql> update barang set NAMA_BARANG='PERAUT'
    -> where KDBARANG='B03';
Query OK, 1 row affected (2.65 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Perintah dalam tanda [] bersifat opsional untuk mengubah suatu baris dengan suatu kondisi tertentu.

4. SELECT

Perintah SELECT digunakan untuk menampilkan isi dari suatu tabel yang dapat dihubungkan dengan tabel yang lainnya.

a. Menampilkan data untuk semua kolom menggunakan asterisk (*) :

   SELECT * FROM namatabel;

```
mysql> select*from barang;
+----------+-------------+--------+-------+
| KDBARANG | NAMA_BARANG | SATUAN | HARGA |
+----------+-------------+--------+-------+
| B01      | BUKU TULIS  | LUSIN  | 50000 |
| B02      | PULPEN      | LUSIN  | 80000 |
| B03      | PERAUT      | LUSIN  | 20000 |
| B04      | PERAUT      | LUSIN  | 45000 |
+----------+-------------+--------+-------+
4 rows in set (0.00 sec)
```

b. Menampilkan data untuk kolom tertentu :

   SELECT kolom1,kolom2,kolom-n FROM namatabel;

```
mysql> select KDBARANG,NAMA_BARANG from barang;
+----------+-------------+
| KDBARANG | NAMA_BARANG |
+----------+-------------+
| B01      | BUKU TULIS  |
| B02      | PULPEN      |
| B03      | PERAUT      |
| B04      | PERAUT      |
+----------+-------------+
4 rows in set (0.00 sec)
```

c. Menampilkan data dengan kondisi data tertentu dengan klausa WHERE:

   SELECT * FROM namatabel WHERE kondisi;

```
mysql> select *from barang where KDBARANG='B02';
+----------+-------------+--------+-------+
| KDBARANG | NAMA_BARANG | SATUAN | HARGA |
+----------+-------------+--------+-------+
| B02      | PULPEN      | LUSIN  | 80000 |
+----------+-------------+--------+-------+
1 row in set (0.00 sec)
```

Beberapa operator perbandingan yang dapat digunakan pada klausa WHERE adalah "="(sama dengan) , > (lebih dari), < (kurang dari), < > (tidak sama dengan), >= (lebih dari atau sama dengan), <= (kurang dari atau sama dengan). Adapun operator lain, yaitu : AND, OR, NOT, BETWEEN-AND, IN dan LIKE.

Praktik!

1. Buatlah sebuah database dengan nama coba2!
2. Buatlah sebuah tabel dengan nama pet pada database coba2!

Field	Type	Null	Key	Default	Extra
name	varchar(20)	NO			
owner	varchar(20)	YES		NULL	
species	varchar(20)	YES		NULL	
sex	char(1)	YES		NULL	
birth	date	YES		NULL	
death	date	YES		NULL	

3. Isi data pada tabel pet, sbb:

name	owner	species	sex	birth	death
Puffball	Diane	Hamster	f	1999-03-03	NULL
Claws	Gwen	cat	m	1994-03-17	NULL
Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL
Fang	Benny	dog	m	1990-08-27	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL

4. Tampilkan semua isi/record tabel pet!
5. Ubah data tanggal lahir hewan yang bernama Bowser menjadi: 1979-08-31 !
6. Tampilkan satu baris / record data yang telah diubah tadi yaitu record dengan nama Bowser saja!
7. Hapus hewan yang bernama Chirpy!
8. Tampilkan record atau data yang tanggal kelahirannya lebih dari atau sama dengan

1998-1-1 !

9. Tampilkan semua hewan dengan spesies kucing dan kucing tersebut berjenis kelamin betina!
10. Dengan satu perintah select, tampilkan semua hewan dengan spesies ular atau spesies burung (dalam satu tabel ada hewan spesies ular dan burung saja)!
11. Dengan satu perintah select, tampilkan semua hewan yang berspesies kucing dengan kelamin laki-laki atau hewan yang berspesies anjing dengan kelamin betina (tampilkan dalam satu tabel)!

Evaluasi dan Pertanyaan

1. Tulis semua perintah-perintah SQL percobaan di atas beserta outputnya !
2. Tulislah kesimpulan Anda!

TULISKAN JAWABAN ANDA

PADA LEMBAR KERJA PRAKTIKUM (LKP)

MODUL IV
DML dan Retrieve Data (Bagian 2)
Pert. 3 (1x170 menit)

Tujuan :
Setelah menyelesaikan modul ini, Anda diharapkan dapat :

1. Mencari dan menampilkan data dengan perintah **select**
2. Mengkombinasikan perintah select dengan perintah lainnya

Dasar Teori

a. Memberikan nama lain pada kolom : SELECT namakolomlama AS namakolombaru FROM namatabel;

   Berikut ini perintah untuk memberikan nama lain pada kolom jenis menjadi jenis_film pada tabel jenisfilm:

```
SELECT JENIS AS TYPE FROM JENISFILM;
```

b. Menggunakan alias untuk nama tabel: SELECT namalias .jenis, namalias .harga FROM namatabel namalias;

   Berikut ini perintah untuk memberikan alias pada tabel jenisfilm :

```
SELECT J.JENIS, J.HARGA FROM JENISFILM J;
```

c. Menampilkan data lebih dari dua tabel: SELECT * FROM namatabel1, namatabel2, namatabel-n;

d. Nested Queries / Subquery (IN, NOT IN, EXISTS, NOT EXISTS)

   Subquery berarti query di dalam query. Dengan menggunakan subquery, hasil dari query akan menjadi bagian dari query di atasnya. Subquery terletak di dalam klausa WHERE atau HAVING. Pada klausa WHERE, subquery digunakan untuk memilih baris-baris tertentu yang kemudian digunakan oleh query. Sedangkan pada klausa HAVING, subquery digunakan untuk memilih kelompok baris yang kemudian digunakan oleh query.

   Contoh 1: perintah untuk menampilkan data pada tabel jenisfilm yang mana data pada kolomjenis-nya tercantum pada tabel film menggunakan IN :

```
SELECT * FROM JENISFILM WHERE JENIS IN (SELECT JENIS FROM FILM);
```

   atau menggunakan EXISTS

```
SELECT * FROM JENISFILM WHERE EXISTS (SELECT * FROM FILM WHERE
```

HARGA > 2000);

Pada contoh di atas:

SELECT JENIS FROM FILM disebut subquery, sedangkan :

SELECT * F ROM JENISFILM berkedudukan sebagai query. Perhatikan, terdapat data jenis dan harga pada tabel jenisfilm yang tidak ditampilkan. Hal ini disebabkan data pada kolom jenis tidak terdapat pada kolom jenis di tabel film.

Contoh 2: perintah untuk menampilkan data pada tabel jenisfilm yang mana data pada kolom jenis-nya tidak tercantum pada tabel film menggunakan NOT IN:

```
SELECT * FROM JENISFILM WHERE JENIS NOT IN (SELECT JENIS FROM
FILM);
```

atau menggunakan NOT EXISTS

```
SELECT * FROM JENISFILM WHERE NOT EXISTS (SELECT * FROM FILM
WHERE HARGA > 2000);
```

e. Operator comparison ANY dan ALL

Operator ANY digunakan berkaitan dengan subquery. Operator ini menghasilkan TRUE (benar) jika paling tidak salah satu perbandingan dengan hasil subquery menghasilkan nilai TRUE. Ilustrasinya jika:

Gaji > ANY (S)

Jika subquery S menghasilkan G1, G2, ..., Gn, maka kondisi di atas identik dengan:

(gaji > G1) OR (gaji > G2) OR ... OR (gaji > Gn)

Contoh: perintah untuk menampilkan semua data jenisfilm yang harganya bukan yang terkecil:

```
SELECT * FROM JENISFILM WHERE HARGA > ANY (SELECT HARGA FROM
JENISFILM);
```

Operator ALL digunakan untuk melakukan perbandingan dengan subquery. Kondisi dengan ALL menghasilkan nilai TRUE (benar) jika subquery tidak menghasilkan apapun atau jika perbandingan menghasilkan TRUE untuk setiap nilai query terhadap hasil subquery.

Contoh : perintah untuk menampilkan data jenisfilm yang harganya paling tinggi:

```
SELECT * FROM JENISFILM WHERE HARGA >= ALL (SELECT HARGA FROM
JENISFILM);
```

f. Sintak ORDER BY

Klausa ORDER BY digunakan untuk mengurutkan data berdasarkan kolom tertentu sesuai dengan tipe data yang dimiliki. Contoh : perintah untuk mengurutkan data film berdasarkan kolom judul:

```
SELECT * FROM FILM ORDER BY JUDUL;
```

atau tambahkan ASC untuk pengurutan secara ascending (menaik) :

```
SELECT * FROM FILM ORDER BY JUDUL ASC;
```

atau tambahkan DESC untuk pengurutan secara descending (menurun):

```
SELECT * FROM FILM ORDER BY JUDUL DESC;
```

g. Sintak DISTINCT

Distinct adalah kata kunci ini untuk menghilangan duplikasi. Sebagai Contoh, buat sebuah tabel pelanggan yang berisi nama dan kota asal dengan beberapa record isi dan beberapa kota asal yang sama. Kemudian ketikkan perintah berikut:

```
SELECT DISTINCT KOTA FROM PELANGGAN;
```

Dengan perintah di atas maka nama kota yang sama hanya akan ditampilkan satu saja.

h. UNION, INTERSECT dan EXCEPT

UNION merupakan operator yang digunakan untuk menggabungkan hasil query, dengan ketentuan jumlah, nama dan tipe kolom dari masing-masing tabel yang akan ditampilkan datanya harus sama. Berikut ini perintah untuk memperoleh data pada tabel film dimana jenisnya action dan horor:

```
SELECT JENIS, JUDUL FROM FILM WHERE JENIS = 'ACTION' UNION SELECT
JENIS, JUDUL FROM FILM WHERE JENIS = 'HOROR';
```

Perintah di atas identik dengan:

SELECT JENIS, JUDUL FROM FILM WHERE JENIS = 'ACTION' OR JENIS = 'HOROR';

Namun tidak semua penggabungan dapat dilakukan dengan OR, yaitu jika bekerja pada dua tabel atau lebih.

INTERSECT merupakan operator yang digunakan untuk memperoleh data dari dua buah query dimana data yang ditampilkan adalah yang memenuhi kedua query tersebut dengan ketentuan jumlah, nama dan tipe kolom dari masing-masing tabel yang akan ditampilkan datanya harus sama.

SELECT * FROM namatabel1 INTERSECT SELECT * FROM namatabel2;

Pada MySQL tidak terdapat operator INTERSECT namun sebagai gantinya dapat menggunakan operator IN seperti contoh 1 pada bagian Nested Queries.

EXCEPT / Set Difference merupakan operator yang digunakan untuk memperoleh data dari dua buah query dimana data yang ditampilkan adalah data yang ada pada hasil query 1 dan tidak terdapat pada data dari hasil query 2 dengan ketentuan jumlah, nama dan tipe kolom dari masing-masing tabel yang akan ditampilkan datanya harus sama.

SELECT * FROM namatabel1 EXCEPT SELECT * FROM namatabel2;

Pada MySQL tidak terdapat operator EXCEPT namun sebagai gantinya dapat menggunakan operator NOT IN seperti contoh 2 pada bagian Nested Queries.

Praktik!

1. Buat tabel pegawai sebagai berikut:

Field	Type	Null	Key	Default	Extra
idpegawai	char(6)	NO			
namadepan	varchar(20)	YES		NULL	
namabelakang	varchar(25)	NO			
email	varchar(25)	NO			
telepon	varchar(20)	YES		NULL	
tglkontrak	date	NO			
idjob	varchar(10)	NO			
gaji	int(8)	YES		NULL	
tunjangan	int(8)	YES		NULL	
idmanajer	char(6)	YES		NULL	
iddepartemen	char(4)	YES		NULL	

2. Isi data tabel (data dapat dilihat pada halaman terakhir)!
3. Tampilkan semua kolom di tabel!
4. Tampilkan kolom idpegawai, namabelakang dan gaji saja!
5. Tampilkan kolom idpegawai, namabelakang, gaji, tunjangan dan sebuah kolom baru yaitu tunjangan+gaji yang berisi jumlah tunjangan dan gaji !
6. Ubah tunjangan menjadi NULL untuk pegawai dengan idpegawai = E003. Kemudian lakukan kembali percobaan 5.
7. Seperti percobaan 5, tampilkan kolom idpegawai, namabelakang, gaji, tunjangan dan sebuah kolom baru (gunakan alias) yaitu total_pendapatan yang berisi jumlah tunjangan dan gaji!
8. Tambahkan record baru dengan value: E006,lincoln, burrows, linc@yahoo.com, 085275384544, 2008-09-01, L0006, 1750000, NULL, ex, coml.
9. Untuk pegawai yang ber-id E004 dan E005 ubah idmanajernya menjadi al!
10. Sekarang tampilkan kolom idmanajer saja!
11. Dari percobaan 10, terdapat 3 idmanajer yang sama dengan total record 6, sekarang tampilkan
12. idmanajer tanpa duplikasi idmanajer sehingga akan tampil 4 record dengan idmanajer yang berbeda!
13. Tampilkan pegawai yang gajinya antara 1750000 - 1250000!
14. Tampilkan tabel pegawai yang terurut berdasarkan namabelakang (dari a ke z)!
15. Tampilkan tabel pegawai yang diurutkan berdasarkan nama depan dengan urutan terbalik (dari z ke a)!

Evaluasi dan pertanyaan

1. Tulis semua perintah-perintah SQL percobaan di atas beserta outputnya!
2. Beri kesimpulan Anda!

TULISKAN JAWABAN ANDA
PADA LEMBAR KERJA PRAKTIKUM (LKP)

MODUL V

FUNGSI AGREGAT

Pert. 4 (1x170 menit)

Tujuan :

Setelah menyelesaikan modul ini mahasiswa diharapkan mahir menggunakan perintah fungsi agregat

Dasar Teori

Aggregate Functions (COUNT, SUM, AVG, MIN, MAX)

a. C OU N T

Perintah yang digunakan untuk menghitung jumlah baris suatu kolom pada tabel.

Contoh : Perintah untuk menghitung jumlah baris kolom jenis pada tabeljenisfilm:

```
SELECT COUNT(namafield) FROM nama_tabel;
```

b . SUM

Perintah yang digunakan untuk menghitung jumlah nilai suatu kolom pada tabel. Contoh : perintah untuk menghitung jumlah nilai kolom harga pada tabel jenisfilm :

```
SELECT SUM(namafield) FROM nama_tabel;
```

c . AVG

Perintah yang digunakan untuk menghitung rata- rata dari ni lai suatu kolom pada tabel. Contoh : perintah untuk menghitung rata-rata dari kolom harga pada tabel jenisfilm:

```
SELECT AVG(namafield) FROM nama_tabel;
```

d . MIN

Perintah yang digunakan untuk menampilkan nilai terkeci l dari suatu kolom pada tabel. Contoh : perintah untuk menampilkan nilai terkecil dari kolom harga pada tabel jenisfilm:

```
SELECT MIN(namafield) FROM nama_tabel;
```

e . MAX

Perintah yang digunakan untuk menampilkan nilai terbesar dari suatu kolom pada table. Contoh : perintah untuk menampi lkan nilai terbesar dari kolom harga pada table jenisfilm :

```
SELECT MAX(namafield) FROM nama_tabel;
```

Praktik!!

1. Buat sebuah nama database dengan nama perdagangan.
2. Buat sebuah table dengan nama barang dimana ketentuannya seperti dibawah ini :

Field	Type	Null	Key	Default	Extra
kode_barang	varchar(6)	NO	PRI		
nama_barang	varchar(30)	NO			
satuan_barang	varchar(20)	NO			
stok_barang	int(11)	YES		NULL	
harga_barang	int(11)	YES		NULL	

3. Isi data ke dalam table barang seperti berikut ini :

kode_barang	nama_barang	satuan_barang	stok_barang	harga_barang
B1	HARDISK	BUAH	12	500000
B2	MP3 PLAYER	UNIT	30	200000
B3	DVD PLAYER	UNIT	50	350000
B4	FLASHDISK	BUAH	12	100000
B5	MOUSE	BUAH	34	50000

4. Tampilkan semua isi record pada table barang.
5. Ubah nama barang DVD Player menjadi TAPE dan stok barang menjadi 25.
6. Tampilkan satu baris / record data yang telah diubah tadi yaitu record dengan nama DVD Player.
7. Hapus kode barang yang bernama MOUSE.
8. Tampilkan record / data yang mempunyai satuan barang bernilai UNIT.
9. Tampilkan jumlah baris dan kolom dari field kode barang dan nama barang pada table barang.
10. Tampilkan jumlah stok barang dan jumlah harga barang pada table barang.
11. Tampilkan jumlah rata – rata harga barang pada table barang
12. Tampilkan jumlah stok barang terkecil pada table barang
13. Tampilkan jumlah stok barang terbesar pada table barang

TULISKAN JAWABAN ANDA
PADA LEMBAR KERJA PRAKTIKUM (LKP)

MODUL VI

RETRIEVE, GROUP, FILTER DAN PATTERN MATCHING

Pert. 5 & 6 (@ 1x170 menit)

Tujuan :

Setelah melakukan percobaan ini, Anda diharapkan dapat :

1. Mampu meretrieve data dan mengelompokkannya
2. Mampu menampilkan data dengan pencocokan pola / karakter

Dasar Teori

- **RETRIEVE SQL dengan GROUP BY dan HAVING**

Klausa GROUP BY digunakan untuk melakukan pengelompokkan data. Syntax :

```
SELECT field1,SUM(field2) FROM namatable GROUP BY field1;
```

Sebagai contoh, terdapat table barang dengan data sebagai berikut :

kode_barang	nama_barang	satuan_barang	stok_barang	harga_barang
B1	HARDISK	BUAH	12	500000
B2	MP3 PLAYER	UNIT	30	200000
B3	TAPE	UNIT	25	350000
B4	FLASHDISK	BUAH	12	100000

Klausa HAVING digunakan untuk menentukan kondisi bagi klausa GROUP BY. Kelompok yang memenuhi HAVING saja yang akan dihasilkan. Syntax :

```
SELECT field1 FROM namatabel GROUP BY field1 HAVING COUNT(field2);
```

- **PATTERN MATCHING (Pencocokan Pola / Karakter)**

Fungsi string digunakan untuk menampilkan data yang di dasarkan pada pencarian dengan karakter. Pada pencarian data digunakan syntax LIKE, pada dasarnya syntax LIKE hampir sama dengan syntax = .

Bedanya kalau syntax = , maka pencarian karakter harus sesuai dengan kata yang kita buat tetapi dengan menggunakan LIKE karakter yang akan kita tampilkan tidak harus lengkap hanya dengan menuliskan salah satu huruf atau kata saja, maka semua data yang akan kita cari akan ditampilkan.

SQL mempunyai dua symbol khusus yang dipakai untuk pencocokan pola :

1. % : digunakan untuk mencocokkan karakter sebelum atau sesudah tanda %;
2. _ : digunakan untuk mencari karakter sebanyak jumlah tanda _.

Contoh:

LIKE '%GLASGOW%' artinya digunakan untuk mencari data pada kolom tertentu yang megandung karakter 'GLASGOW'.

Bentuk umumnya :

```
SELECT * FROM nama_tabel WHERE nama_kolom LIKE 'char%';
SELECT * FROM nama_tabel WHERE nama_kolom LIKE '%char';
SELECT * FROM nama_tabel WHERE nama_kolom LIKE '%char%';
SELECT * FROM nama_tabel WHERE nama_kolom NOT LIKE '%char%';
SELECT * FROM nama_tabel WHERE nama_kolom LIKE '_';
```

Praktik 1!

Tampilkan record / data hanya kolom satuan barang dan digabungkan dengan jumlah stok barang yang dikelompokkan berdasarkan kolom satuan barang pada table barang diatas!

```
mysql> select satuan_barang,sum(stok_barang) from barang group by satuan_barang;
+---------------+-----------------+
| satuan_barang | sum(stok_barang) |
+---------------+-----------------+
| BUAH          |              24 |
| UNIT          |              55 |
+---------------+-----------------+
2 rows in set (0.08 sec)
```

Evaluasi dan Pertanyaan !

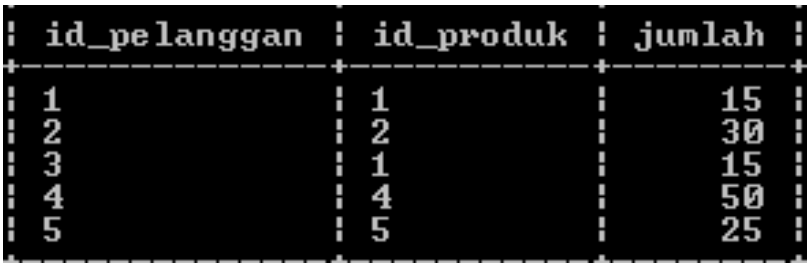
1. Jika syntax ini : SELECT kode_barang, satuan_barang, SUM(stok_barang) from barang GROUP BY satuan_barang. Apa yang akan terjadi ?. Jelaskan!
2. Jika syntax ini diketikkan pada cmd : SELECT nama_barang, satuan_barang, SUM(stok_barang) GROUP BY satuan_barang. Apa yang akan terjadi ?. Jelaskan!

Praktik 2!

1. Buat table dengan nama penjualan dengan ketentuan sebagai berikut :

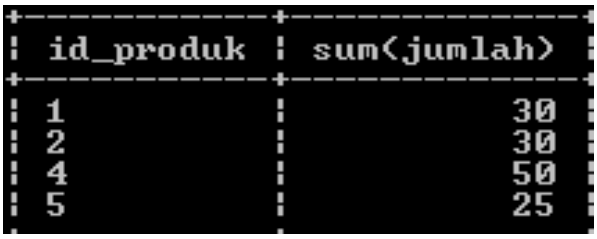
```
+--------------+---------+------+-----+---------+-------+
| Field        | Type    | Null | Key | Default | Extra |
+--------------+---------+------+-----+---------+-------+
| id_pelanggan | char(3) | YES  |     | NULL    |       |
| id_produk    | char(3) | YES  |     | NULL    |       |
| jumlah       | int(5)  | NO   |     |         |       |
+--------------+---------+------+-----+---------+-------+
```

2. Isi data table penjualan seperti dibawah ini :



id_pelanggan	id_produk	jumlah
1	1	15
2	2	30
3	1	15
4	4	50
5	5	25

3. Buat SQL sehingga tampilannya seperti berikut :



id_produk	sum(jumlah)
1	30
2	30
4	50
5	25

4. Buat SQL sehingga tampilannya sebagai berikut :

ID PRODUK	TOTAL
4	50

5. Buat table barang dan isi datanya seperti dibawah ini :

kode_brg	nama_brg	harga_modal	harga_beli	stok
B01	Sabun	2000	2500	15
B02	Pasta Gigi	2500	3000	15
B03	Sikat Gigi	3000	4000	10
B04	Rokok	6000	7000	30
B05	Korek Api	500	600	10

6. Buat SQL sehingga tampilannya sebagai berikut :

kode_brg	nama_brg	harga_modal	harga_beli	stok
B02	Pasta Gigi	2500	3000	15
B03	Sikat Gigi	3000	4000	10
B05	Korek Api	500	600	10

7. Buat SQL sehingga tampilannya sebagai berikut :

kode_brg	nama_brg	harga_modal	harga_beli	stok
B01	Sabun	2000	2500	15
B03	Sikat Gigi	3000	4000	10

8. Buat SQL sehingga tampilannya sebagaia berikut :

kode_brg	nama_brg	harga_modal	harga_beli	stok
B01	Sabun	2000	2500	15
B02	Pasta Gigi	2500	3000	15
B03	Sikat Gigi	3000	4000	10
B05	Korek Api	500	600	10

9. Buat SQL sehingga tampilannya sebagai berikut :

kode_brg	nama_brg	harga_modal	harga_beli	stok
B04	Rokok	6000	7000	30

10. Buat SQL sehingga tampilannya sebagai berikut :

kode_brg	nama_brg	harga_modal	harga_beli	Stok
B01	Sabun	2000	2500	15
B04	Rokok	6000	7000	30

Evaluasi dan Pertanyaan :

1. Tulis semua perintah – perintah SQL pada percobaan di atas beserta outputnya !
2. Berikan kesimpulan Anda !

TULISKAN JAWABAN ANDA
PADA LEMBAR KERJA PRAKTIKUM (LKP)

MODUL VII
Relasi Database
Pert. 7 & 8 (1x170 menit)

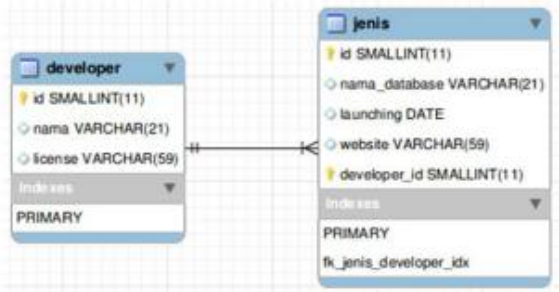
Tujuan :

Setelah melakukan percobaan ini, Anda diharapkan dapat :

- Mahasiswa dapat memahami konsep relasi tabel pada database
- Mahasiswa dapat membuat relasi beberapa tabel menggunakan database MySQL
- Mahasiswa mampu menyelesaikan beberapa permasalahan menggunakan relasi tabel database MySQL

Dasar Teori

Relasi tabel merupakan hubungan yang terjadi pada suatu tabel dengan lainnya yang mempresentasikan hubungan antar objek di dunia nyata dan berfungsi untuk mengatur mengatur operasi suatu database.



Ada 3 macam relasi tabel, diantaranya :

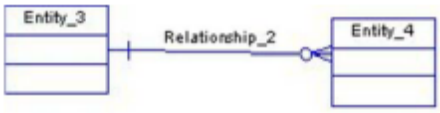
[1] One-To-One (1-1)

Mempunyai pengertian "Setiap baris data pada tabel pertama dihubungkan hanya ke satu baris data pada tabel ke dua". Contohnya : relasi antara tabel mahasiswa dan tabel orang tua. Satu baris mahasiswa hanya berhubungan dengan satu baris orang tua begitu juga sebaliknya.

[2] One-To-Many (1-N)

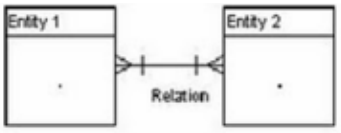
Mempunyai pengertian "Setiap baris data dari tabel pertama dapat dihubungkan ke satu baris atau lebih data pada tabel ke dua". Contohnya : relasi perwalian antara tabel dosen

dan tabel mahasiswa. Satu baris dosen atau satu dosen bisa berhubungan dengan satu baris atau lebih mahasiswa.



[3] Many-To-Many (N-M)

Mempunyai pengertian "Satu baris atau lebih data pada tabel pertama bisa dihubungkan ke satu atau lebih baris data pada tabel ke dua". Artinya ada banyak baris di tabel satu dan tabel dua yang saling berhubungan satu sama lain. Contohnya : relasi antar tabel mahasiswa dan tabel mata kuliah. Satu baris mahasiswa bisa berhubungan dengan banyak baris mata kuliah begitu juga sebaliknya.



Praktikum !

1. Buatlah database baru dengan nama library.
2. Selanjutnya buatlah tabel **category** dan tabel **book** dengan struktur sebagai berikut :

   Tabel **category**

Field	Type	Null	Key	Default	Extra
category_id	int(10)	NO	PRI	NULL	auto_increment
category_name	varchar(30)	NO		NULL	

   Tabel **book**

Field	Type	Null	Key	Default	Extra
book_id	int(10)	NO	PRI	NULL	
title	varchar(30)	NO		NULL	
author	varchar(30)	NO		NULL	
publisher	varchar(30)	NO		NULL	

   Jangan lupa memastikan bahwa storage engine tabel yang dibuat bertipe INNODB karena hanya tipe ini yang mendukung relasi dan pembuatan foreign key

3. Setelah kita analisa, relasi antara tabel category dengan tabel book adalah 1:N dimana satu category buku bisa terdiri dari banyak buku. Karena relasi 1:N maka atribut kunci pada tabel category (category_id) bertamu ke tabel book sehingga ada penambahan atribut pada tabel book (category_id). Berikut struktur baru tabel book :

Field	Type	Null	Key	Default	Extra
book_id	int(10)	NO	PRI	NULL	
title	varchar(30)	NO		NULL	
author	varchar(30)	NO		NULL	
publisher	varchar(30)	NO		NULL	
category_id	int(10)	NO		NULL	

4. Langkah selanjutnya adalah menambahkan foreign key pada tabel book sebagai berikut :

```
mysql> alter table book
    -> add foreign key(category_id)
    -> references category(category_id);
```

Sehingga tampilan baru tabel book sebagai berikut :

```
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| book_id     | int(10)     | NO   | PRI | NULL    |       |
| title       | varchar(30) | NO   |     | NULL    |       |
| author      | varchar(30) | NO   |     | NULL    |       |
| publisher   | varchar(30) | NO   |     | NULL    |       |
| category_id | int(10)     | NO   | MUL | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

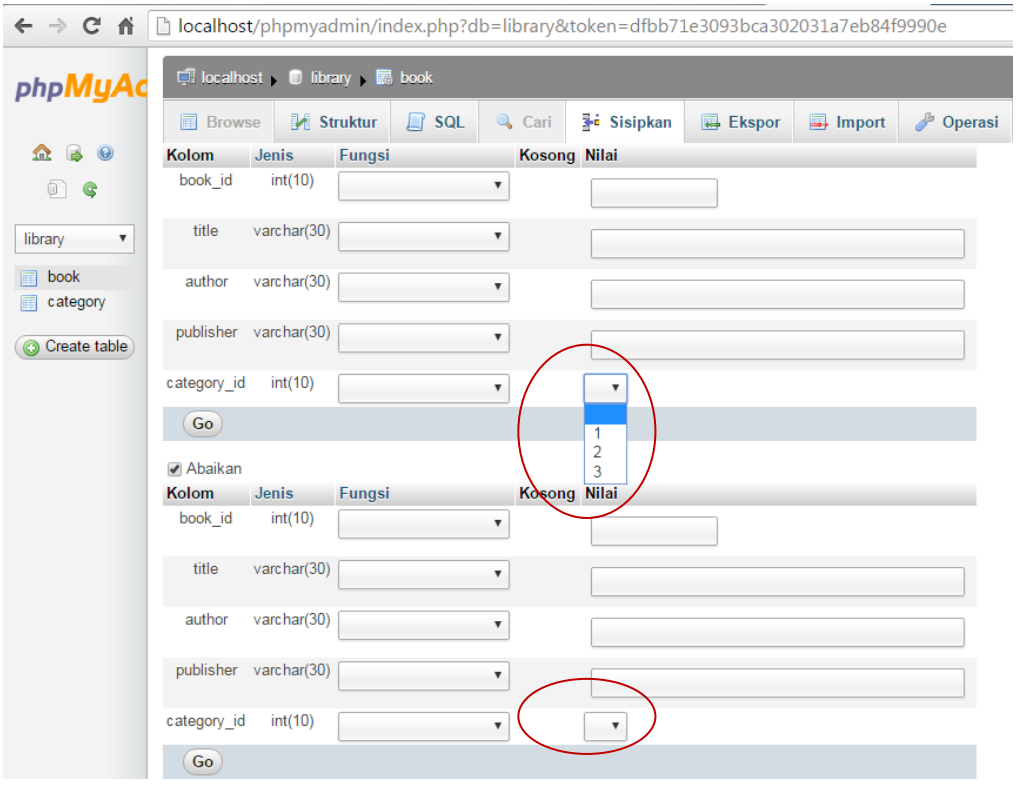
5. Ok, sekarang lanjutkan dengan mengisi dulu tabel category sebagai berikut :

```
+-------------+---------------+
| category_id | category_name |
+-------------+---------------+
|           1 | Komputer      |
|           2 | Telekomunikasi|
|           3 | Jaringan      |
+-------------+---------------+
```

6. Sekarang kita akan cek apakah relasi 1:N yang kita buat telah berhasil apa tidak. Dalam hal ini kita akan menggunakan GUI XAMPP (phpmyadmin) untuk memudahkan :

Cara 1 :

Jika relasi berhasil, maka ketika menu insert dipilih pada tabel book maka kolom category_id yang berasal dari tabel category akan otomatis menggunakan menu combo box seperti berikut :



Cara 2 :

Klik database library, kemudian pada menu operasi pilih desainer (perancang). Jika relasi sukses dan benar maka akan menampilkan ERD secara physic sebagai berikut :

Catatan Penting :

Jika saat membuat relasi (menambahkan FK berhasil) tetapi saat dicek hasilnya tidak seperti diatas maka hal yang paling awal harus dilakukan adalah dengan mengecek storage engine dari tabel-tabel yang berelasi. Pastikan mesin penyimpanannya menggunakan INNODB, karena biasanya default ketika membuat tabel yaitu bertipe MYISAM. Jika ternyata bertipe MYISAM, harus segera diganti kemudian filed tamu juga dihapus dan dibuat ulang untuk menghilangkan pembacaan perintah ketika masih bertipe MYISAM.

7. Baik, selanjutnya masukkan data ke tabel book sebagai berikut :

book_id	title	author	publisher	category_id
131181	24 Jam Belajar PHP	Edy Winarni ST, M.Eng, Ali	Elex Media Komputindo	1
131182	Sistem Telekomunikasi di Indo	Gauzali Saydam	Alfabeta	2
131183	Pengantar Jaringan Komputer d	Iwan Sofana	Informatika	3

8. Tambahkan data pada tabel book sehingga datanya menjadi sebagai berikut :

book_id	title	author	publisher	category_id
131181	24 Jam Belajar PHP	Edy Winarni ST, M.Eng, Ali	Elex Media Komputindo	1
131182	Sistem Telekomunikasi di Indo	Gauzali Saydam	Alfabeta	2
131183	Pengantar Jaringan Komputer d	Iwan Sofana	Informatika	3
131184	Pemrograman VB.Net	M. Yunus	Andi Offset	1

Praktek Mandiri :

1. Hapuslah data category 3 dari tabel category.
2. Tambahkan data pada tabel book berupa **book_id** : 131185, **title** : Mikrotik, **author** : Rinaldi, **publisher** : Informatika, **category_id** : 10.
3. Tentukan solusi yang tepat untuk bisa menyelesaikan soal 1 dan 2.
4. Buatlah kesimpulan dari percobaan yang anda telah kerjakan pada lembar LKP beserta urutan instruksi/perintah dari setiap soal.

Case Study :

- Buat database kepegawaian dan Tabel jabatan

	Fields	Tipe data
PK	id_jabatan	INT,10,Not Null, Auto increment, Primary key
	nama_jabatan	VARCHAR, 30, Not Null

- Buat tabel divisi

	Fields	Tipe data
PK	id_divisi	INT,10,Not Null, Auto increment, Primary key
	nama_divisi	VARCHAR, 30, Not Null

- Buat tabel data_pegawai

	Fields	Tipe data
PK	nik	INT,10,Not Null, Auto increment, Primary key
	nama	VARCHAR, 30, Not Null
	alamat	VARCHAR, 30, Not Null
FK	id_jabatan	INT,10,Not Null
FK	dd_divisi	INT,10,Not Null

- Relasi tabel yang dihasilkan

- Memasukkan data tabel jabatan

id_jabatan	nama_jabatan
1001	Kepala Divisi
1002	Manager
1003	Karyawan

- Memasukkan data tabel divisi

id_divisi	nama_divisi
2001	Training
2002	Sistem dan Jaringan
2003	HRD

- Memasukkan data tabel data_pegawai

nik	nama	alamat	id_jabatan	id_divisi
200000065	Aji Firmansyah	Surabaya	1002	2003
200000066	Rudi Hartono	Bandung	1001	2002
200000067	Aisyah Nila	Semarang	1003	2001

- Menampilkan nik, nama, alamat, nama_jabatan, nama_divisi
- Update nama dan alamat
- Delete data dengan nik = 200000067

⇨ Kerjakan case study diatas di lembar LKP.

MODUL VIII

QUERY DAN VIEW

Pert. 9 (1x170 menit)

Tujuan :

Setelah melakukan percobaan ini, Anda diharapkan dapat :

1. Mampu menampilkan data dengan view
2. Mampu menampilkan gabungan data dari tabel yang berbeda.

Dasar Teori

❖ **Query**

Query adalah pernyataan yang meminta pengguna mengambil informasi. Bagian DML yang terlibat dalam pengambilan informasi disebut bahasa query. Istilah bahasa query sering disamakan dengan istilah bahasa manipulasi data. Sedangkan SQL adalah sebuah sintaks untuk mengeksekusi query.

❖ **View**

Merupakan salah satu objek database, yang secara logika merepresentasikan sub himpunan dari data yang berasal dari satu atau lebih tabel. Kegunaan view adalah untuk membatasi akses database, membuat query kompleks secara mudah, mengijinkan independensi data dan untuk menampilkan view (pandangan) data yang berbeda dari data yang sama.

❖ **Tujuan dari view adalah:**

a. Menurunkan Network Traffic (beban Network).
b. Menyimpan suatu perintah SQL (terutama yang komplek) dimana perintah tersebut sering digunakan dan diakses.
c. Mencegah user untuk dapat mengakses suatu tabel sepenuhnya.
d. Misal user dapat mengakses nama dan nomor telepon tetapi tidak bisa mengakses tanggal lahir dan gaji.

❖ **Sintak dari View adalah :**

CREATE [OR REPLACE] [<algorithm attributes>]

VIEW [database.]< name> [(<columns>)]

AS <SELECT statement> [<check options>];

Contoh:

a) Membuat View:

```
CREATE VIEW pelanggan_simpati AS
SELECT nama,alamat,tgl_lahir,telepon
FROM pelanggan WHERE telepon REGEXP '^081[23]'
ORDER BY nama;
```

b) Cara mengaksesnya:

```
SELECT * FROM pelanggan_simpati;
SELECT nama,alamat FROM pelanggan_simpati;
```

View termasuk dalam komponen database. Secara default, suatu view baru dibuat ke dalam database yang diaktifkan. Untuk membuat secara eksplisit di dalam suatu database tertentu, maka buatlah nama view dengan format: db_name.view_name.

Contoh lain yang akan diberikan adalah view untuk menyimpan informasi detail mahasiswa, dalam hal ini melibatkan 2 tabel, yaitu mahasiswa dan prodi.

Contoh:

```
mysql> create view vDetailMhs as
    ->    select m.nim, m.nama, m.alamat, p.nama_prodi, p.jurusan
    ->    from mahasiswa m, prodi p
    ->    where (m.kode_prodi=p.kode_prodi);

mysql> select * from vDetailMhs;
```

nim	nama	alamat	nama_prodi	jurusan
00543	Moh. Riyan	Karangmalang A-50	Eks Ilmu Komputer	Matematika
10041	Sugiharti	Karangmalang A-23	Ilmu Komputer	Matematika
10043	Ahmad Sholihun	Karangmalang D-17	Ilmu Komputer	Matematika

Dari contoh diatas dapat dijelaskan bahwa view tersebut berisi informasi mahasiswa (nim, nama, alamat) dan informasi prodi mahasiswa yang bersangkutan (nama_prodi dan jurusan). Implementasi view dalam program aplikasi adalah untuk memudahkan dalam mendesain laporan (report).

Soal dan Praktik : Case Study

MODUL IX

JOIN

Pert. 10 & 11 (@1x170 menit)

Tujuan :

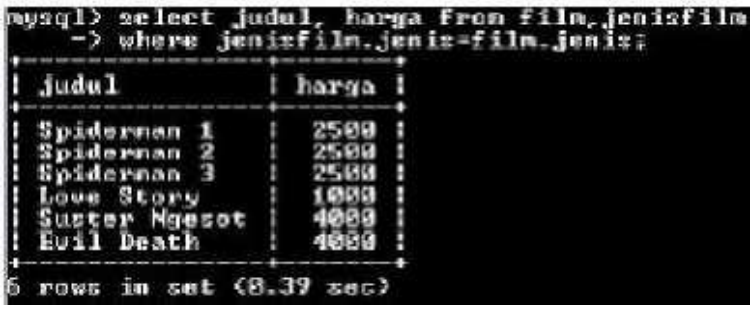
Setelah menyelesaikan modul ini, anda diharapkan dapat:

1. Mampu mengenal beberapa join
2. Mampu melakukan operasi join beberapa tabel

Dasar Teori

❖ **Join**

Operasi Join, Join merupakan operasi yang digunakan untuk menggabungkan dua tabel atau lebih dengan hasil berupa gabungan dari kolom-kolom yang berasal dari tabel-tabel tersebut. Pada join sederhana, tabel-tabel digabungkan dan didasarkan pada pencocokan antara kolom pada tabel yang berbeda.



Pada contoh di atas , jenisfilm.jenis=film.jenis merupakan kondisi untuk menco cokkan data antara kolom jenis milik tabel jenisfilm dan film.

a. Inner Join

Inner join digunakan untuk menampilkan data dari dua tabel yang berisi data sesuai dengan syarat dibelakang on (tidak boleh null), dengan kata lain semua data dari tabel kiri mendapat pasangan data dari tabel sebelah kanan . Berikut ini perintah untuk menampilkan data dari tabel jenisfilm dan film dengan syarat berdasarkan kolom jenis:

```
SELECT * FROM JENISFILM INNER JOIN FILM ON (JENISFILM.JENIS =
FILM.JENIS);
```

b. Left Join

Left join digunakan untuk menampilkan semua data dari tabel sebelah kiri per intah left join beserta pasangannya dari tabel sebelah kanan. Meskipun terdapat data dari sebelah kiri tidak memiliki pasangan, tetap akan ditampilkan dengan pasangannya berupa nilai NULL.

```
SELECT * FROM JENISFILM LEFT JOIN FILM ON (JENISFILM.JENIS =
FILM.JENIS);
```

c. Right Join

Right join digunakan untuk menampilkan semua data dari tabel sebelah kanan perintah right join beserta pasangannya dari tabel sebelah kiri. Meskipun terdapat data dari sebelah kanan tidak memiliki pasangan , tetap akan ditampilkan dengan pasangannya berupa nilai NULL.

```
SELECT * FROM JENISFILM RIGHT JOIN FILM ON (JENISFILM.JENIS =
FILM.JENIS);
```

d. Natural Join

Natural join digunakan untuk menampilkan semua data dari dua tabel dimana jika terdapat kolom yang sama, maka yang akan ditampilkan hanya salah satunya saja, yaitu kolom dari tabel sebelah kiri perintah natural join.

```
SELECT * FROM JENISFILM NATURAL JOIN FILM;
```

Terdapat Penggabungan Natural Join dengan Left dan Right Join:

1. Natural Left Join

Natural left join digunakan untuk menampilkan semua data dari tabe l sebelah kiri perintah natural left join beserta pasangannya dari tabel sebelah kanan. Meskipun terdapat data dari sebelah kiri tidak memiliki pasangan, tetap akan ditampilkan dengan pasangannya berupa nilai NULL.

```
SELECT * FROM JENISFILM NATURAL LEFT JOIN FILM;
```

2. Natural Right Join

Natural right join digunakan untuk menampilkan semua data dari tabel sebelah kanan perintah natural right join beserta pasangannya dari tabel sebelah kiri. Meskipun terdapat data dari sebelah kanan tidak memiliki pasangan, tetap akan ditampilkan dengan pasangannya berupa nilai NULL.

SELECT * FROM JENISFILM NATURAL RIGHT JOIN FILM;

Praktik!

1. Buat sebuah database dengan nama sewa mobil :
2. Buat tabel mobil dan isi datanya :

Field	Type	Null	Key	Default	Extra
kode	varchar(5)	NO	PRI		
jenis	varchar(10)	NO			
merk	varchar(15)	NO			
tarif	int(11)	NO			
nopol	varchar(8)	NO			

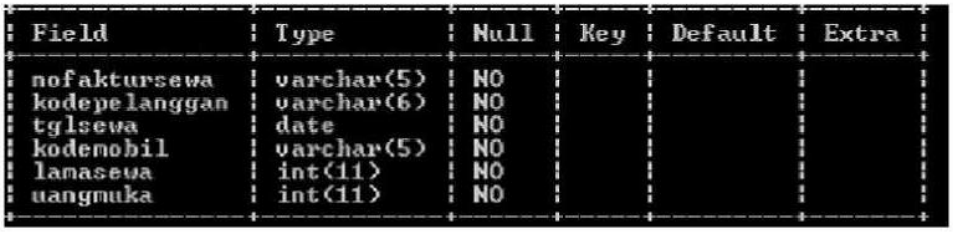
kode	jenis	merk	tarif	nopol
M001	SEDAN	BMW E5	500000	BG1234AA
M002	SEDAN	HONDA CRV	350000	BG2345BB
M003	BUS	MERCEDEZ	1000000	BG3456CC
M004	BUS	DYNA	800000	BG8443DD
M005	TRUCK	HYNO ZX	1500000	BG4638EE
M006	TRUCK	DYNA X1	1500000	BG8473FF

3. Buat tabel pelanggan dan isi datanya:

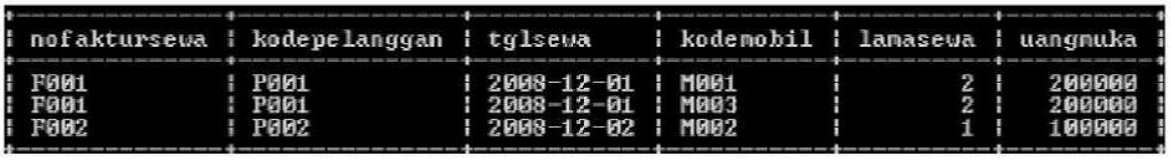
Field	Type	Null	Key	Default	Extra
kode	varchar(6)	NO	PRI		
nama	varchar(15)	NO			
kontak	varchar(15)	NO			
alamat	varchar(30)	NO			
kota	varchar(15)	NO			
kodepos	varchar(5)	NO			
telpon	varchar(15)	NO			
fax	varchar(15)	NO			

kode	nama	kontak	alamat	kota	kodepos	telpon	fax
P001	PT FOX RIVER	HENDRA	JL. JEND. SUDIRMAN 657	BENGKULU	30245	1234567	1234568
P002	CV FOXCON	IWAN	JL. WAHID HASYIM 743	JAKARTA	73429	234567	234568
P003	PT FARMACOM	YANI	JL. AHMAD DAHLAN 45	LAMPUNG	28349	3334445	3334446

4. Buat tabel sewa dan isi datanya:



Field	Type	Null	Key	Default	Extra
nofaktursewa	varchar(5)	NO			
kodepelanggan	varchar(6)	NO			
tglsewa	date	NO			
kodemobil	varchar(5)	NO			
lamasewa	int(11)	NO			
uangmuka	int(11)	NO			



nofaktursewa	kodepelanggan	tglsewa	kodemobil	lamasewa	uangmuka
F001	P001	2008-12-01	M001	2	200000
F001	P001	2008-12-01	M003	2	200000
F002	P002	2008-12-02	M002	1	100000

Evaluasi dan Pertanyaan :

1. Tulis semua perintah – perintah SQL pada percobaan di atas beserta outputnya !
2. Berikan kesimpulan Anda!

TULISKAN JAWABAN ANDA
PADA LEMBAR KERJA PRAKTIKUM (LKP)

MODUL X

STORE PROCEDURE & FUNCTION

Pert. 12 (@1x170 menit)

Tujuan :

Setelah melakukan percobaan ini, Anda diharapkan dapat :

1. Mampu membuat store procedure dan function pada MySQL
2. Mampu menjelaskan manfaat penggunaan store procedure dan function pada MySQL

Dasar Teori

Store procedure dan function adalah rangkaian program yang disimpan dalam database dan dapat dipanggil oleh program lain atau melalui SQL Prompt. Store procedure ditulis dalam bentuk suatu script.

Untuk membuat *stored procedure/function* pada *database* digunakan pernyataan CREATE PROCEDURE atau CREATE FUNCTION.

Keuntungan :

1. Cepat, kompilasi dilakukan didatabase (kadang disebut "pre-compilation") sehingga mengurangi traffic
2. Adanya pemisahan antara database dan access logic dengan application logic sehingga program aplikasi menjadi lebih sederhana dan lebih ringkas
3. Berupa obyek dalam database, sehingga menghilangkan ketergantungan terhadap bahasa program yang digunakan
4. Bersifat portable, jika bisa berjalan didatabase tersebut maka dipastikan jika database bisa terinstall dimanapun maka store procedure pasti bisa dijalankan

Bentuk Umum Store Procedure

Sintak :

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

Keterangan :

- **sp_name** : Nama *routine* yang akan dibuat
- **proc_parameter** : Parameter *stored procedue*, terdiri dari :
  - ✓ IN : parameter yang digunakan sebagai masukan.
  - ✓ OUT : parameter yang digunakan sebagai keluaran
  - ✓ INOUT : parameter yang digunakan sebagai masukan sekaligus keluaran.
- ***routine_body:*** terdiri dari statemen prosedur SQL yang valid.

Agar lebih jelas, perhatikan contoh penggunaannya berikut ini :

contoh 1:

```
mysql> use akademik;
Database changed
mysql> delimiter //
mysql> create procedure pMHS(out x varchar(25))
    -> begin
    -> select nama into x from mhs where kd_prodi = 'P01';
    -> end
    -> //
Query OK, 0 rows affected (0.10 sec)

mysql> call pMHS(@Nama);
    -> select @Nama;
    -> //
Query OK, 1 row affected (0.04 sec)

+-----------+
| @Nama     |
+-----------+
| Moh.Riyan |
+-----------+
1 row in set (0.04 sec)
```

Dari contoh diatas terlihat bahwa parameter "x" (sebagai OUT) digunakan untuk menampung hasil dari perintah `routine_body`. Pernyataan "into x", inilah yang mengakibatkan "x" menyimpan informasi **nama** (sebagai kolom yang ter-*select*). Untuk menjalankan *procedure* digunakan ststemen **`call`**. Pernyataan "call pMHS(@Nama)" menghasilkan informasi yang kemudian disimpan pada parameter "@Nama". Kemudian untuk menampilkan informasi ke layar digunakan pernyataan "select @Nama".

Contoh 2 :

```
mysql> delimiter //
mysql> create procedure dMHS (out x varchar(25), out y varchar(30), in z char(3)
)
    -> begin
    -> select nama,alamat into x,y from mhs where kd_prodi=z;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> call dMHS(@Nama, @Alamat,'P01');
    -> select @Nama, @Alamat;
    -> //
Query OK, 1 row affected (0.00 sec)

+-----------+------------------+
| @Nama     | @Alamat          |
+-----------+------------------+
| Moh.Riyan | Karangmalang A-50 |
+-----------+------------------+
1 row in set (0.00 sec)
```

Dari contoh yang kedua ini terlihat bahwa parameter "z" (sebagai IN) digunakan sebagai jalur untuk masukan *routine* dan parameter "x" dan "y" digunakan untuk menampung hasil dari perintah `routine_body`. Pernyataan "into x, y", inilah yang mengakibatkan "x" dan "y" menyimpan informasi **nama** dan **alamat** (sebagai kolom yang ter-*select*).
Pernyataan "call dMHS(@Nama, @Alamat)" menghasilkan informasi yang kemudian disimpan pada parameter **`@Nama`** dan **`@Alamat,`** sedangkan parameter "z" digunakan untuk menampung string 'P01' yang kemudian digunakan untuk memproses `routine_body` . Kemudian untuk menampilkan informasi ke layar digunakan pernyataan "select @Nama, @Alamat".
Jika diperhatikan pada **contoh1** dan **contoh2**, dalam membuat *routine* selalu menggunakan **delimiter.** Hal ini digunakan untuk mengubah pernyataan *delimiter* dari ";" ke "//" ketika suatu *procedure* sedang didefinisikan. Sehingga sebelum *delimiter*

ditutup, meskipun sudah ditekan **enter** masih dianggap satu-kesatuan perintah. Jika menggunakan perintah **delimiter**, maka untuk menutupnya digunakan karakter *backslash* ('\') karena karakter ini merupakan karakter **escape** untuk MySQL.

Function

Secara *default*, *routine* (*procedure/function*) diasosiasikan dengan *database* yang sedang aktif. Untuk dapat mengasosiasikan *routine* secara eksplisit dengan *database* yang lain, buat *routine* dengan format: **`db_name.sp_name`**.

MySQL mengijinkan beberapa *routine* berisi statemen DDL, seperti CREATE dan DROP. MySQL juga mengijinkan beberapa *stored procedure* (tetapi tidak *stored function*) berisi statemen SQL *transaction*, seperti COMMIT. *Stored function* juga berisi beberapa statemen baik yang secara eksplisit atau implisit **commit** atau **rollback.**

Sintak :

```
CREATE FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

    proc_parameter:
    [ IN | OUT | INOUT ] param_name type

    func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statement or statements
```

Keterangan :

- **`sp_name`**: Nama *routine* yang akan dibuat
- **`proc_parameter`**: Spesifikasi parameter sebagai IN, OUT, atau INOUT valid hanya untuk PROCEDURE. (parameter FUNCTION selalu sebagai parameter IN)
- **`returns`**: Klausa RETURNS dispesifikan hanya untuk suatu FUNCTION. Klausa ini digunakan untuk mengembalikan tipe fungsi, dan *`routine_body`* harus berisi suatu statemen nilai RETURN.
- **`comment`**: Klausa COMMENT adalah suatu ekstensi MySQL, dan mungkin digunakan untuk mendeskripsikan *stored procedure*. Informasi ini ditampilkan dengan statemen SHOW CREATE PROCEDURE dan SHOW CREATE FUNCTION.

Contoh :

```
mysql> delimiter //

mysql> create function fcNamaMHS(x char(25)) returns char(40)
    -> return concat('Nama : ', x);
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> select fcNamaMHS('Sholihun');
+----------------------+
| fcNamaMHS('Sholihun')|
+----------------------+
| Nama : Sholihun      |
+----------------------+
1 row in set (0.02 sec)
```

Dari contoh diatas terlihat bahwa parameter "x" diperlakukan sebagai IN karena sebagaimana dijelaskan sebelumnya bahwa fungsi hanya bisa dilewatkan dengan parameter IN. Kemudian untuk pengembalian nilainya, digunakan tipe data dengan kisaran nilai tertentu (dalam hal ini char(40)) dengan diawali pernyataan **returns**.

Pernyataam "concat('Nama : ', x)" merupakan *routine_body* yang akan menghasilkan gabungan string "Nama :" dengan nilai dari parameter "x" yang didapat ketika fungsi ini dieksekusi. Perintah yang digunakan untuk mengeksekusi fungsi adalah "select fcNamaMHS('Sholihun')".

Soal dan Praktik : Case Study

MODUL XI

TRIGGERS

Pert. 13 & 14 (@1x170 menit)

Tujuan :

Setelah melakukan percobaan ini, Anda diharapkan dapat :

1. Mampu membuat trigger untuk perintah DML pada tabel database
2. Mampu menampilkan data hasil pemrosesan trigger dalam database

Dasar Teori

Pernyataan CREATE TRIGGER digunakan untuk membuat *trigger*, termasuk aksi apa yang dilakukan saat *trigger* diaktifkan. *Trigger* berisi program yang dihubungkan dengan suatu tabel atau *view* yang secara otomatis melakukan suatu aksi ketika suatu baris di dalam tabel atau *view* dikenai operasi INSERT, UPDATE atau DELETE.

Sintak :

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

Keterangan :

- ***[DEFINER = { user | CURRENT_USER }]***: Definisi *user* yang sedang aktif, sifatnya opsional.
- ***trigger_name***: Nama *trigger*.
- ***trigger_time***: waktu menjalankan *trigger*. Ini dapat berupa BEFORE atau AFTER.
  - ✓ BEFORE: Membuat *trigger* diaktifkan sebelum dihubungkan dengan suatu operasi.
  - ✓ AFTER: Membuat *trigger* diaktifkan setelah dihubungkan dengan suatu operasi.
- ***trigger_event***: berupa kejadian yang akan dijalankan *trigger*.
- ***trigger_event*** dapat berupa salah satu dari berikut:
  - ✓ INSERT : *trigger* diaktifkan ketika sebuah *record* baru disisipkan ke dalam tabel. Contoh: statemen INSERT, LOAD DATA, dan REPLACE.
  - ✓ UPDATE : *trigger* diaktifkan ketika sebuah *record* dimodifikasi. Contoh: statemen UPDATE.
  - ✓ DELETE : *trigger* diaktifkan ketika sebuah *record* dihapus. Contoh: statemen DELETE dan REPLACE.

  Catatan : `trigger_event` **tidak** merepresentasikan statemen SQL yang diaktifkan *trigger* sebagai suatu operasi tabel. Sebagai contoh, *trigger* `BEFORE` untuk `INSERT` akan diaktifkan tidak hanya oleh statemen `INSERT` tetapi juga statemen `LOAD DATA`.
- ***tbl_name:*** Nama tabel yang berasosiasi dengan *trigger*.
- ***trigger_stmt***: Statemen (tunggal atau jamak) yang akan dijalankan ketika *trigger* aktif.

Sekarang kita masuk ke bahasan utama, yaitu implementasi. Untuk menerapkan TRIGGER, PROCEDURE, FUNCTION dan VIEW dibutuhkan suatu relasi, misalkan: *mahasiswa* dan *prodi*, sebagaimana yang diilustrasikan dengan perintah SQL di bawah ini.

Praktikkan !

1. Buatlah database dengan nama akademik.
2. Kemudian buatlah tabel mhs dan prodi dengan struktur sebagai berikut :

```
mysql> desc mhs;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| nim      | char(5)     | NO   | PRI |         |       |
| nama     | varchar(25) | YES  |     | NULL    |       |
| alamat   | varchar(50) | YES  |     | NULL    |       |
| kd_prodi | char(3)     | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.02 sec)

mysql> desc prodi;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| kd_prodi   | char(3)     | NO   | PRI |         |       |
| nama_prodi | varchar(25) | YES  |     | NULL    |       |
| jurusan    | varchar(20) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

3. Agar kedua tabel diatas berelasi, buatlah kunci tamu pada tabel mhs yang mereferensi ke tabel prodi (kd_prodi) sebagai berikut :

```
mysql> alter table mhs add foreign key(kd_prodi)
    -> references prodi(kd_prodi);
Query OK, 0 rows affected (0.25 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc mhs;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| nim      | char(5)     | NO   | PRI |         |       |
| nama     | varchar(25) | YES  |     | NULL    |       |
| alamat   | varchar(50) | YES  |     | NULL    |       |
| kd_prodi | char(3)     | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

4. Isikan data ke tabel mhs dan prodi sebagai berikut :

```
mysql> select * from prodi;
+----------+--------------------+------------+
| kd_prodi | nama_prodi         | jurusan    |
+----------+--------------------+------------+
| P01      | Eks Ilmu Komputer  | Matematika |
| P02      | Ilmu Komputer      | Matematika |
| P03      | D3 Komsi           | Matematika |
| P04      | D3 Rekmed          | Matematika |
| P05      | D3 Ellins          | Fisika     |
+----------+--------------------+------------+
5 rows in set (0.00 sec)

mysql> select * from mhs;
+-------+----------------+--------------------+----------+
| nim   | nama           | alamat             | kd_prodi |
+-------+----------------+--------------------+----------+
| 00543 | Muhammad       | Karangmalang A-50  | P01      |
| 10041 | Sugiharta      | Karangmalang A-23  | P02      |
| 10043 | Ahmad Sholihun | Karangmalang D-17  | P02      |
+-------+----------------+--------------------+----------+
3 rows in set (0.00 sec)
```

5. Untuk pembuatan TRIGGER, contoh yang akan dibahas adalah mencatat kejadian-kejadian yang terjadi beserta waktunya pada tabel mhs, dan catatan-catatan tadi disimpan dalam tabel yang lain, misal **log_mhs**. Misalkan struktur tabel **log_mhs** adalah sebagai berikut :

```
mysql> create table log_mhs
    -> (kejadian varchar(25),
    -> waktu datetime);
Query OK, 0 rows affected (0.06 sec)

mysql> desc log_mhs;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| kejadian | varchar(25) | YES  |     | NULL    |       |
| waktu    | datetime    | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

6. Buatlah TRIGGER untuk mencatat kejadian setelah dilakukan perintah INSERT pada tabel mhs dan disimpan ke dalam tabel log_mhs sebagai berikut :

```
mysql> create trigger ins_mhs after insert on mhs
    -> for each row insert into log_mhs values ('Tambah data',now());
Query OK, 0 rows affected (0.11 sec)

mysql> insert into mhs values
    -> ('00631','Hanif','Kalasan','P01');
Query OK, 1 row affected (0.07 sec)

mysql> select * from mhs;
+-------+----------------+--------------------+-----------+
| nim   | nama           | alamat             | kd_prodi  |
+-------+----------------+--------------------+-----------+
| 00543 | Muhammad       | Karangmalang A-50  | P01       |
| 00631 | Hanif          | Kalasan            | P01       |
| 10041 | Sugiharta      | Karangmalang A-23  | P02       |
| 10043 | Ahmad Sholihun | Karangmalang D-17  | P02       |
+-------+----------------+--------------------+-----------+
4 rows in set (0.00 sec)

mysql> select * from log_mhs;
+-------------+---------------------+
| kejadian    | waktu               |
+-------------+---------------------+
| Tambah data | 2016-03-11 09:33:42 |
+-------------+---------------------+
1 row in set (0.00 sec)
```

Dari contoh diatas dapat dilihat bahwa ketika satu record pada tabel ***mhs*** *disisipkan (insert), maka secara otomatis tabel* ***log_mhs*** *akan disisipkan satu record, yaitu kejadian* `'Tambah data'` *dan waktu saat record pada tabel* ***mhs*** *disisipkan.*

7. Selanjutnya buatlah TRIGGER untuk mencatat perubahan data yang dilakukan setelah mendapatkan perintah UPDATE pada tabel mhs :

```
mysql> create trigger updt_mhs after update on mhs
    -> for each row insert into log_mhs values ('Ubah Data',now());
Query OK, 0 rows affected (0.07 sec)

mysql> update mhs
    -> set nama='Moh.Riyan' where nim='00543';
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from mhs;
+-------+----------------+--------------------+-----------+
| nim   | nama           | alamat             | kd_prodi  |
+-------+----------------+--------------------+-----------+
| 00543 | Moh.Riyan      | Karangmalang A-50  | P01       |
| 00631 | Hanif          | Kalasan            | P01       |
| 10041 | Sugiharta      | Karangmalang A-23  | P02       |
| 10043 | Ahmad Sholihun | Karangmalang D-17  | P02       |
+-------+----------------+--------------------+-----------+
4 rows in set (0.00 sec)

mysql> select * from log_mhs;
+-------------+---------------------+
| kejadian    | waktu               |
+-------------+---------------------+
| Tambah data | 2016-03-11 09:33:42 |
| Ubah Data   | 2016-03-11 09:37:31 |
+-------------+---------------------+
2 rows in set (0.00 sec)
```

Dari contoh diatas dapat dilihat bahwa ketika satu record pada tabel ***mhs*** *diperbaharui (update), maka secara otomatis tabel* ***log_mhs*** *akan disisipkan satu record, yaitu kejadian* `'Ubah data'` *dan waktu saat record pada tabel* ***mhs*** *diperbaharui.*

8. Kemudian TRIGGER untuk perintah DML DELETE pada tabel mhs sebagai berikut, dimana proses kejadiannya direkam pada tabel log_mhs :

```
mysql> create trigger del_mhs after delete on mhs
    -> for each row insert into log_mhs values('Hapus Data',now());
Query OK, 0 rows affected (0.07 sec)
```

Hasilnya :

```
mysql> select * from mhs;
+-------+----------------+-------------------+----------+
| nim   | nama           | alamat            | kd_prodi |
+-------+----------------+-------------------+----------+
| 00543 | Moh.Riyan      | Karangmalang A-50 | P01      |
| 10041 | Sugiharta      | Karangmalang A-23 | P02      |
| 10043 | Ahmad Sholihun | Karangmalang D-17 | P02      |
+-------+----------------+-------------------+----------+
3 rows in set (0.00 sec)

mysql> select * from log_mhs;
+-------------+---------------------+
| kejadian    | waktu               |
+-------------+---------------------+
| Tambah data | 2016-03-11 09:33:42 |
| Ubah Data   | 2016-03-11 09:37:31 |
| Hapus Data  | 2016-03-11 09:39:42 |
+-------------+---------------------+
3 rows in set (0.00 sec)
```

Dari contoh diatas dapat dilihat bahwa ketika satu record pada tabel ***mhs*** *dihapus (delete), maka secara otomatis tabel* ***log_mhs*** *akan disisipkan satu record, yaitu kejadian* `'Hapus data'` *dan waktu saat record pada tabel* ***mhs*** *dihapus.*

9. Tampilkanlah semua TRIGGER yang telah dibuat :

```
mysql> show triggers;
+----------+--------+-----------+-------------+--------+---------+-----------+
| Trigger  | Event  | Table     | Statement   | Timing | Created | sql_mode  |
+----------+--------+-----------+-------------+--------+---------+-----------+
| ins_mhs  | INSERT | mahasiswa |      S1     | AFTER  | NULL    |    SQL1   |
| updt_mhs | UPDATE | mahasiswa |      S2     | AFTER  | NULL    |    SQL2   |
| del_mhs  | DELETE | mahasiswa |      S3     | AFTER  | NULL    |    SQL3   |
+----------+--------+-----------+-------------+--------+---------+-----------+
```

Keterangan (*record* pada kolom "statement" dan "sql_mode"):

```
S1 : insert into log_mhs values('Tambah data',now())
S2 : insert into log_mhs values('Ubah data',now())
S3 : insert into log_mhs values('Hapus data',now())
SQL1 : STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
SQL2 : STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
SQL3 : STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
```

Karena hasil eksekusi perintah "show triggers" sangat panjang, tampilan di atas sengaja diedit dengan tujuan agar mudah dipahami.

Dalam implementasinya untuk pekerjaan sehari-hari, pembuatan *trigger* dan tabel *log*, digunakan untuk mencatat kejadian suatu tabel yang dianggap rawan serangan *cracker*. Dengan struktur *trigger* yang baik sesuai kebutuhan, *administrator* dapat melakukan pelacakan dan *recovery* data dengan cepat karena mengetahui *record* mana saja yang "diserang". Atau, dihubungkan dengan program aplikasi (*user interface*) agar mengaktifkan

alarm, jika terdapat operasi *database* pada waktu yang tidak seharusnya (misalkan malam hari).

Soal dan Evaluasi

1. Pada LKP, jelaskan manfaat penggunaan TRIGGER (tulis tangan) !
2. Case Study (lanjut dibahas ke pertemuan 14)