MODUL PRAKTEK

SISTEM MANAJEMEN BASIS DATA

DENGAN MYSQL

PROGRAM STUDI TEKNIK INFORMATIKA
JENJANG S-1

FAKULTAS TEKNOLOGI INDUSTRI
INSTITU SAINS &TEKNOLOGI AKPRIND
YOGYAKARTA

SUMBER DOKUMEN:

SUMBER : MODUL PRAKTIKUM BASIS DATA
PROGRAM PRA S-2
JURUSAN ILMU KOMPUTER & ELEKTRONIKA, FMIPA,
UNIVERSITAS GADJAH MADA YOGYAKARTA

PENYUSUN : ARIF NURWIDYANTORO, S.KOM., M.CS.

MODUL SATU

Pertemuan 1 Pengantar MySQL

Arsitektur MySQL

MySQL bekerja di lingkungan jaringan menggunakan arsitektur client/server. Sebuah program berjalan sebagai server, dan berbagai macam client mengirimkan request ke server. MySQL memiliki komponen sebagai berikut:

- MySQL Server, atau mysqld, adalah program server database. Program ini mengatur akses ke database di disk dan memory. MySQL Server bersifat multi-thread dan mendukung beberapa koneksi secara simultan. MySQL Server memiliki arsitektur modular, sehingga mendukung berbagai macam storage engine untuk menangani beberapa jenis tabel yang berbeda.
- Program Client, adalah program yang digunakan untuk berkomunikasi dengan server untuk memanipulasi informasi dalam database yang ditangani server. Beberapa contoh program client:
  - MySQL Query Browser dan MySQL Administrator, menyediakan antarmuka grafis untuk akses ke server
  - mysql, program command line untuk akses ke server berbasis teks (ini yang akan kita gunakan)
  - program command line lain, seperti mysqlimport untuk import database ke file, mysqldump untuk membuat backup, mysqladmin untuk administrasi server, dan mysqlcheck untuk mengecek integritas file database
- MySQL non-client utilities, program yang beraksi secara independen terhadap server. Program-program ini **tidak perlu** melakukan koneksi terlebih dahulu ke server untuk beroperasi. Contoh: myisamchk untuk operasi mengecek tabel dan repair.

Koneksi dan diskoneksi ke Server

Program client yang kita gunakan adalah mysql. Untuk mengetahui parameter apa saja yang diterima oleh program mysql, kita dapat memasukkan command:

```
shell> mysql --help
```

Secara umum, untuk melakukan koneksi ke dalam database, kita gunakan perintah sebagai berikut:

```
shell> mysql -h localhost -u user -p
```

Perintah di atas berarti kita akan mengakses Server di localhost dengan nama user “user” dan password. Command line kemudian akan meminta password kita. Setelah kita berhasil masuk ke mysql, prompt kita akan berubah menjadi mysql>

```
mysql>
```

Beberapa parameter lain untuk koneksi adalah sebagai berikut:

--protocol	The protocol to use for the connection
--host atau -h	The host where the server is running (default: localhost)
--port atau -P	The port number for TCP/IP connections (default: 3306)
--shared-memory-base-name	The shared-memory name for shared-memory connections
--socket	The Unix socket filename or named-pipe name

Sedangkan parameter untuk identifikasi user adalah sebagai berikut:

--user atau -u	The MySQL account username

--password atau -p The MySQL account password

Beberapa instalasi MySQL mengijinkan user untuk mengakses sebagai anonymous (tanpa nama) ke server dalam localhost, sehingga kita dapat mengakses secara langsung menggunakan perintah:

```
shell> mysql
```

Untuk mengetahui status sambungan ke mysql, kita bisa menggunakan command STATUS.

```
mysql> STATUS;
--------------
mysql  Ver 14.14 Distrib 5.5.23, for debian-linux-gnu (x86_64) using readline 6.1

Connection id:          59
Current database:
Current user:       root@localhost
SSL:                Not in use
Current pager:          stdout
Using outfile:          ''
Using delimiter:    ;
Server version:         5.5.23-2 (Debian)
Protocol version:   10
Connection:         Localhost via UNIX socket
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
UNIX socket:        /run/mysqld/mysqld.sock
Uptime:                 41 min 1 sec

Threads: 1  Questions: 671  Slow queries: 0  Opens: 495  Flush tables: 1  Open tables: 49
Queries per second avg: 0.272
--------------
```

Untuk memutus sambungan ke mysql, kita bisa lakukan dengan command quit (atau \q) di mysql> prompt.

```
mysql> quit
Bye
```

Memasukkan Query

Setelah kita berhasil masuk ke mysql, kita dapat memasukkan beberapa command ke dalam mysql. Berikut adalah contoh sederhana untuk mengetahui nomor versi dan tanggal sekarang.

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 5.5.23-2  | 2012-05-30   |
```

```
+-----------+--------------+
1 row in set (0.00 sec)
```

Query tersebut mengilustrasikan tentang mysql:

- Sebuah command normalnya terdiri atas perintah SQL diikuti oleh tanda titik koma (;).
- Ketika kita memasukkan command, mysql mengirimkannya ke server untuk dieksekusi dan ditampilkan hasilnya, kemudian menampilkan prompt mysql> lagi untuk menandakan bahwa ia telah siap menerima perintah.
- mysql menampilkan hasil query dalam bentuk tabular (baris dan kolom). Baris pertama menunjukkan label untuk kolom tersebut. Baris selanjutnya adalah hasil query.
- mysql menampilkan jumlah baris yang ditemukan dan waktu eksekusi query yang menunjukkan secara kasar performa server.

Keyword dapat dimasukkan dengan lettercase apa saja. Query-query berikut ini adalah query yang sama.

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Contoh berikut merupakan query untuk melakukan kalkulasi.

```
mysql> select sin(pi()/4), (4+1)*5;
+------------------+---------+
| sin(pi()/4)      | (4+1)*5 |
+------------------+---------+
| 0.707106781186547 |      25 |
+------------------+---------+
1 row in set (0.00 sec)
```

Kita juga dapat memasukkan beberapa statement dalam satu baris

```
mysql> select version(); select now();
+-------------------+
| version()         |
+-------------------+
| 5.1.61-0+squeeze1 |
+-------------------+
1 row in set (0.00 sec)

+---------------------+
| now()               |
+---------------------+
| 2012-05-27 08:34:27 |
+---------------------+
1 row in set (0.00 sec)
```

Sebuah command juga dapat ditulis dalam beberapa baris. mysql menentukan kapan statement berakhir dengan mencari tanda titik koma (;). Dengan kata lain, mysql menerima masukkan tetapi belum dieksekusi sebelum menemukan tanda titik koma.

```
mysql> select
    -> user()
    -> ,
    -> current_date;
+----------------+--------------+
| user()         | current_date |
+----------------+--------------+
| root@localhost | 2012-05-27   |
+----------------+--------------+
1 row in set (0.00 sec)
```

Pada contoh di atas, prompt mysql> berubah menjadi -> setelah memasukkan baris pertama dari sebuah query berbaris banyak. Ini menunjukkan mysql menunggu sampai semua statement selesai dimasukkan.

Jika kita ingin membatalkan statement yang sedang kita masukkan, kita dapat memasukkan perintah \c:

```
mysql> select
    -> user()
    -> \c
mysql>
```

Pada contoh di atas, prompt berubah kembali menjadi mysql> yang menunjukkan bahwa ia telah siap menerima command baru.

Berikut adalah prompt yang ditunjukkan mysql dan maksudnya.

Prompt	Arti
mysql>	Siap menerima command baru
->	Menunggu baris selanjutnya dari multiline command
'>	Menunggu baris selanjutnya, untuk mengakhiri string yang diawali oleh (')
“>	Menunggu baris selanjutnya, untuk mengakhiri string yang diawali oleh (“)
`>	Menunggu baris selanjutnya, untuk mengakhiri identifier yang diawali oleh (`)
/*>	Menunggu baris selanjutnya, untuk mengakhiri komentar yang diawali oleh (/*)

Prompt '> dan “> terjadi ketika mysql sedang menerima string (atau mysql sedang menunggu string selesai dimasukkan). Di MySQL, kita bisa menulis string yang diawali dan diakhiri dengan tanda (') atau (“), misalnya 'goodbye' atau “goodbye”. mysql juga memperbolehkan masukan string lebih dari satu baris. Ketika prompt ini muncul, artinya kita memasukkan string yang diawali tanda ['] atau [“] tetapi belum kita tutup.

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

Pada perintah di atas berarti kita kurang menutup string dengan tanda ('). Kita bisa mengakhiri kesalahan ini dengan membatalkan query sebagai berikut.

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
```

Artinya, kita menutup tanda string dan membatalkan perintah dengan \c. Jika kita hanya memasukkan tanda \c saja, maka \c dianggap sebagai bagian dari string tersebut.

Untuk mengetahui perintah apa saja yang diterima oleh mysql, kita bisa menggunakan command help atau \h sebagai berikut:

```
mysql> \h

For information about MySQL products and services, visit:
   http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
   http://dev.mysql.com/
To buy MySQL Enterprise support, training, or other products, visit:
    https://shop.mysql.com/

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use       (\u) Use another database. Takes database name as argument.
charset    (\C) Switch to another charset. Might be needed for processing binlog with
multi-byte charsets. warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.

For server side help, type 'help contents'
```

Menulis Komentar

Komentar dapat ditulis dengan diawali oleh karakter #. Semua karakter setelah karakter # akan dianggap sebagai komentar dan tidak diproses.

```
mysql> SELECT 1+1;     # This comment continues to the end of line
+-----+
| 1+1 |
+-----+
|   2 |
+-----+
1 row in set (0.00 sec)
mysql>
```

Selain menggunakan karakter #, komentar juga dapat ditulis dengan diawali oleh sekuens '–-'.

```
mysql> SELECT 1+1;     -- This comment continues to the end of line
+-----+
| 1+1 |
+-----+
|   2 |
+-----+
1 row in set (0.00 sec)
mysql>
```

Untuk komentar yang lebih dari satu baris, komentar dapat dikelilingi oleh karakter '/*' dan '*/'. Semua karakter di antara kedua tanda tersebut tidak akan diproses.

```
mysql> SELECT 1 /* this is an in-line comment */ + 1;
+---------+
| 1   + 1 |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)
mysql> SELECT 1+
    -> /*
   /*> this is a
   /*> multiple-line comment
   /*> */
    -> 1;
+-------+
| 1+

1 |
+-------+
|     2 |
+-------+
```

```
1 row in set (0.00 sec)
mysql>
```

Operator

MySQL memiliki berbagai macam operator sebagai berikut:

Name	Description
AND, &&	Logical AND
BINARY	Cast a string to a binary string
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
/	Division operator
DIV	Integer division
<=>	NULL-safe equal to operator
=	Equal operator
>=	Greater than or equal operator
>	Greater than operator
IS NULL	NULL value test
IS	Test a value against a boolean
<<	Left shift
<=	Less than or equal operator
<	Less than operator
LIKE	Simple pattern matching
-	Minus operator
%	Modulo operator
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOT, !	Negates value
||, OR	Logical OR

Name	Description
+	Addition operator
REGEXP	Pattern matching using regular expressions
>>	Right shift
RLIKE	Synonym for REGEXP
SOUNDS LIKE	Compare sounds
~	Invert bits
*	Times operator
-	Change the sign of the argument
XOR	Logical XOR

Seperti bahasa pemrograman, operator-operator tersebut juga memiliki *precedences*. Precedences itu adalah sebagai berikut (dari yang paling rendah ke yang paling tinggi):

1. :=
2. ||, OR, XOR
3. &&, AND
4. NOT
5. BETWEEN, CASE, WHEN, THEN, ELSE
6. =, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
7. |
8. &
9. <<, >>
10. -, +
11. *, /, DIV, %, MOD
12. ^
13. - (unary minus), ~ (unary bit inversion)
14. !
15. BINARY, COLLATE

Untuk menghindari precedence, dapat menggunakan tanda kurung buka dan kurung tutup.

```
mysql> SELECT 1+2*3;
+-------+
| 1+2*3 |
+-------+
|     7 |
+-------+
1 row in set (0.00 sec)
mysql> SELECT (1+2)*3;
+---------+
| (1+2)*3 |
+---------+
|       9 |
```

```
+---------+
1 row in set (0.00 sec)
mysql>
```

Membuat dan Memilih Database

Setelah mengetahui bagaimana cara memasukkan command, kita dapat mulai mengakses database.

Untuk melihat database apa saja yang terdapat dalam server, kita bisa melihatnya menggunakan perintah

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| phpmyadmin         |
+--------------------+
3 rows in set (0.00 sec)
```

Pembuatan database dapat kita lakukan dengan menggunakan perintah CREATE DATABASE. Misalnya, kita mencoba membuat database dengan nama 'pethouse'. Kita dapat melakukannya sebagai berikut:

```
mysql> CREATE DATABASE pethouse;
Query OK, 1 row affected (0.00 sec)
```

Kemudian kita dapat melihat apakah database yang telah kita buat berhasil dengan menggunakan command SHOW DATABASES

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| pethouse         |
| phpmyadmin         |
+--------------------+
4 rows in set (0.00 sec)
```

Untuk memilih database mana yang akan kita gunakan, kita bisa menggunakan command USE. Misalnya, kita ingin mengakses database 'pethouse' yang telah kita buat, maka kita bisa menggunakan command sebagai berikut:

```
mysql> USE pethouse
Database changed
```

Command USE, sama seperti QUIT, tidak membutuhkan tanda titik koma (;), namun jika kita menggunakan tanda titik koma, hal itu tidak menjadi masalah.

Selanjutnya, untuk mengetahui database apa yang saat ini sedang kita gunakan, kita dapat menggunakan command SELECT DATABASE(); sebagai berikut:

```
mysql> SELECT DATABASE();
```

```
+------------+
| DATABASE() |
+------------+
| pethouse |
+------------+
1 row in set (0.00 sec)
```

Menghapus Database

Untuk menghapus database, dapat menggunakan perintah DROP DATABASE. Data yang tersimpan dalam database juga akan ikut terhapus.

```
mysql> DROP DATABASE pethouse;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

Membuat Tabel

Setelah membuat database, kita akan membuat tabel. Pastikan kita telah menggunakan database pethouse (menggunakan USE). Untuk melihat semua tabel yang terdapat dalam sebuah database, kita bisa menggunakan command:

```
mysql> show tables;
Empty set (0.00 sec)
```

Selanjutnya, kita membuat tabel 'pet' yang berisikan nama-nama binatang beserta informasi yang berkaitan dengan binatang tersebut. Berikut adalah commandnya:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
Query OK, 0 rows affected (0.41 sec)
```

Untuk melihat apakah tabel kita sudah ditambahkan kita bisa melihat tabel dengan SHOW TABLES

```
mysql> SHOW TABLES;
+--------------------+
| Tables_in_pethouse |
+--------------------+
| pet                |
+--------------------+
1 row in set (0.00 sec)
```

Untuk melihat apakah tabel yang dibuat telah sesuai dengan yang kita inginkan, kita dapat menggunakan command DESCRIBE:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
```

```
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

Memasukkan Data ke Tabel

Setelah tabel berhasil dibuat, mari kita masukkan data ke dalamnya. Masukkan data di bawah ini menggunakan INSERT.

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	

Contoh untuk memasukkan Fluffy:

```
mysql> INSERT INTO pet(name, owner, species, sex, birth, death)
    -> VALUES ('Fluffy', 'Harold', 'cat', 'f', '1993-02-04', NULL);
Query OK, 1 row affected (0.00 sec)
```

atau bisa juga menggunakan command:

```
mysql> INSERT INTO pet
    -> VALUES ('Fluffy', 'Harold', 'cat', 'f', '1993-02-04', NULL);
Query OK, 1 row affected (0.00 sec)
```

Selain menggunakan INSERT, kita juga bisa menggunakan LOAD DATA untuk memasukkan data dari sebuah file. Buatlah sebuah file bernama 'pet.txt' dengan data berikut:

name	owner	species	sex	birth	death
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	

File 'pet.txt' tersebut diisi dengan satu record setiap barisnya, masing-masin nilai dipisahkan dengan tab dengan urutan kolom seperti yang didefinisikan pada pembuatan tabel. Untuk menggantikan nilai NULL dilakukan dengan '\N'. Contoh baris dalam file tersebut:

```
Fang    Benny   dog     m       1990-08-27      \N
```

File tersebut kemudian kita masukkan ke tabel menggunakan command LOAD DATA dan diarahkan ke lokasi di mana kita menyimpan file 'pet.txt'. Pada contoh di bawah, file 'pet.txt' berada di '/app/mysql/pet.txt'.

```
mysql> LOAD DATA LOCAL INFILE '/app/mysql/pet.txt' INTO TABLE pet;
Query OK, 3 rows affected, 1 warning (0.00 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 1
```

Untuk melihat data yang telah dimasukkan, kita bisa menggunakan perintah SELECT.

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
```

```
| name     | owner | species | sex | birth        | death      |
+----------+-------+---------+-----+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny | dog      | m    | 1990-08-27 | NULL       |
| Bowser   | Diane | dog      | m    | 1979-08-31 | 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       |
| Puffball | Diane | hamster | f     | 1999-03-30 | NULL       |
+----------+-------+---------+-----+------------+------------+
```

Menghapus Tabel

Untuk menghapus tabel dapat menggunakan command DROP TABLE. Semua data yang terdapat pada tabel yang dihapus juga akan hilang.

```
mysql> DROP TABLE pet;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

TUGAS

Menggunakan mysql command line,

1. Buat sebuah tabel bernama 'event' dalam database 'pethouse' dengan kolom sebagai berikut:

Nama kolom	Tipe data	Ukuran
name	VARCHAR	20
date	DATE	
type	VARCHAR	15
remark	VARCHAR	255

2. Isikan tabel tersebut dengan data sebagai berikut menggunakan INSERT

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened

3. Isikan tabel tersebut dengan data sebagai berikut menggunakan LOAD DATA

name	date	type	remark
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

4. Tampilkan data yang sudah berhasil dimasukkan menggunakan command SELECT.

Ambil screenshotnya dan masukkan ke sebuah dokumen disertai identitas.

MODUL DUA

Pertemuan 2 Tabel dan Manipulasi Data

Membuat tabel

Secara umum, syntax untuk membuat tabel adalah sebagai berikut:

```
CREATE TABLE nama_tbl (
nama_kolom1 tipe_data(length) [NOT NULL|NULL] [DEFAULT nilai_default] [AUTO_INCREMENT] ,
nama_kolom2 tipe_data(length) [NOT NULL|NULL] [DEFAULT nilai_default] [AUTO_INCREMENT] ,
…
);
```

Contoh membuat tabel

```
mysql> CREATE DATABASE PRAK2;
Query OK, 1 row affected (0.34 sec)

mysql> USE PRAK2;
Database changed

mysql> CREATE TABLE TABLE1(
    -> id INT AUTO_INCREMENT,
    -> name VARCHAR(30) NOT NULL,
    -> salary FLOAT(10,2) DEFAULT 1000000
    -> );

mysql> DESCRIBE TABLE1;
+--------+-------------+------+-----+------------+-------+
| Field  | Type        | Null | Key | Default    | Extra |
+--------+-------------+------+-----+------------+-------+
| id     | int(11)     | YES  |     | NULL       |       |
| name   | varchar(30) | NO   |     | NULL       |       |
| salary | float(10,2) | YES  |     | 1000000.00 |       |
+--------+-------------+------+-----+------------+-------+
3 rows in set (0.00 sec)
```

Tipe data

Tipe data dalam MySQL terdiri atas beberapa jenis:

- **Tipe data teks**

  Tipe data teks terdiri atas beberapa tipe sebagai berikut:

CHAR( )	A fixed section from 0 to 255 characters long.
VARCHAR( )	A variable section from 0 to 255 characters long.

TINYTEXT	A string with a maximum length of 255 characters.
TEXT	A string with a maximum length of 65535 characters.
BLOB	A binary string with a maximum length of 65535 bytes.
MEDIUMTEXT	A string with a maximum length of 16777215 characters.
MEDIUMBLOB	A string with a maximum length of 16777215 bytes.
LONGTEXT	A string with a maximum length of 4294967295 characters.
LONGBLOB	A string with a maximum length of 4294967295 bytes.
ENUM(value1, value2,...)	A string object that can only have one value, chosen from the list of 'value1', 'value2', … An ENUM can have a maximum of 65535 distinct values
SET (value1, value2, ...)	A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', … SET values can have a maximum of 64 members.

- **Tipe data numerik**

Tipe data numerik adalah sebagai berikut:

TINYINT( )	-128 to 127 normal, 0 to 255 UNSIGNED.
SMALLINT( )	-32768 to 32767 normal, 0 to 65535 UNSIGNED.
MEDIUMINT( )	-8388608 to 8388607 normal, 0 to 16777215 UNSIGNED.
INT( )	-2147483648 to 2147483647 normal, 0 to 4294967295 UNSIGNED.
BIGINT( )	-9223372036854775808 to 9223372036854775807 normal, 0 to 18446744073709551615 UNSIGNED.
FLOAT (M , D)	A small number with a floating decimal point.
DOUBLE(M, D)	A large number with a floating decimal point.
DECIMAL(M, D)	A fixed decimal point.
BIT( )	Bit values (0 or 1)

Tipe data numerik FLOAT, DOUBLE, dan DECIMAL, nilai M menunjukkan jumlah digit keseluruhan, sedangkan D menunjukkan jumlah digit di belakang koma. Sebagai contoh, jika ada kolom yang didefinisikan sebagai FLOAT(7,4) maka akan tampak seperti 999.9999.

- **Tipe data date**

DATE	YYYY-MM-DD.
DATETIME	YYYY-MM-DD HH:MM:SS.
TIMESTAMP	YYYYMMDDHHMMSS.

TIME	HH:MM:SS.

Membuat tabel dengan primary key

Primary key adalah nilai yang menjadi pembeda antara satu record data dengan record yang lain. Membuat tabel dengan primary key ditunjukkan sebagai berikut:

```
mysql> CREATE TABLE TABLE2(
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.13 sec)

mysql> DESCRIBE TABLE2;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Pada contoh di atas, kolom 'id' diberikan 'AUTO_INCREMENT', artinya kolom ini akan diberikan nilai increment secara automatis. Kolom 'id' diset sebagai PRIMARY KEY.

Dalam satu tabel, hanya boleh ada satu kolom yang diberi status 'AUTO_INCREMENT', dan harus menjadi key. Jika tidak akan muncul pesan error sebagai berikut:

```
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
```

Membuat tabel dengan primary key juga dapat dilakukan sebagai berikut:

```
mysql> CREATE TABLE TABLE3(
    -> id INT AUTO_INCREMENT,
    -> name VARCHAR(30),
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.17 sec)
mysql> DESCRIBE TABLE3;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Memberikan nilai UNIQUE

UNIQUE memberikan batasan bahwa nilai dalam sebuah kolom harus distinct (atau tidak ada data yang nilainya sama). Error akan terjadi jika kita menambahkan nilai yang sama dengan data yang sudah ada untuk kolom tersebut.

```
mysql> CREATE TABLE cars(
    -> id INT PRIMARY KEY AUTO_INCREMENT,
    -> plate VARCHAR (10),
    -> brand VARCHAR (10),
    -> UNIQUE (plate)
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql> DESC cars;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| plate | varchar(10) | YES  | UNI | NULL    |                |
| brand | varchar(10) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> INSERT INTO cars VALUES (1, 'AB-1234-CD', 'Toyota');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM cars;
+----+------------+--------+
| id | plate      | brand  |
+----+------------+--------+
|  1 | AB-1234-CD | Toyota |
+----+------------+--------+
1 row in set (0.00 sec)

mysql> INSERT INTO cars VALUES (2, 'AB-1234-CD', 'Suzuki');
ERROR 1062 (23000): Duplicate entry 'AB-1234-CD' for key 'plate'
```

Parameter IF NOT EXISTS

Parameter IF NOT EXISTS digunakan untuk mengecek apakah sebuah tabel dengan nama yang sama sudah ada atau belum sebelum membuat tabel. Penggunaannya dapat dilihat sebagai berikut:

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_PRAK2 |
```

```
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
+-----------------+
3 rows in set (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS TABLE1(
    -> id INT AUTO_INCREMENT,
    -> name VARCHAR(30) NOT NULL,
    -> salary FLOAT(10,2) DEFAULT 1000000
    -> );
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS TABLE4(
    -> id INT AUTO_INCREMENT,
    -> name VARCHAR(30) NOT NULL,
    -> salary FLOAT(10,2) DEFAULT 1000000
    -> );
Query OK, 0 rows affected (0.14 sec)
mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
+-----------------+
4 rows in set (0.00 sec)
```

Pada contoh di atas, kita mencoba membuat tabel dengan nama TABEL1 yang sudah ada. Mysql akan memberikan satu buat warning. Namun ketika, kita membuat tabel yang belum ada, maka tabel tersebut akan terbentuk.

Menyalin tabel dengan statement SELECT

Kita juga dapat membuat tabel dari tabel lain menggunakan statement SELECT. Caranya ditunjukkan sebagai berikut.

Pertama, kita membuat terlebih dahulu tabel sumber dengan nama 'employee' seperti sebagai berikut:

```
mysql> CREATE TABLE employee(
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> first_name VARCHAR(15),
    -> last_name VARCHAR(15),
```

```
    -> start_date DATE,
    -> end_date DATE,
    -> salary FLOAT(8,2),
    -> city VARCHAR(10),
    -> description VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql> DESCRIBE employee;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| first_name  | varchar(15) | YES  |     | NULL    |                |
| last_name   | varchar(15) | YES  |     | NULL    |                |
| start_date  | date        | YES  |     | NULL    |                |
| end_date    | date        | YES  |     | NULL    |                |
| salary      | float(8,2)  | YES  |     | NULL    |                |
| city        | varchar(10) | YES  |     | NULL    |                |
| description | varchar(15) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Kemudian kita isikan dengan data ke tabel 'employee' sebagai berikut:

1	Jason	Martin	1996-07-25	2006-07-25	1234.56	Toronto	Programmer
2	Alison	Mathews	1976-03-21	1986-02-21	6661.78	Vancouver	Tester
3	James	Smith	1978-12-12	1990-03-15	6544.78	Vancouver	Tester
4	Celia	Rice	1982-10-24	1999-04-21	2344.78	Vancouver	Manager
5	Robert	Black	1984-01-15	1998-08-08	2334.78	Vancouver	Tester
6	Linda	Green	1987-07-30	1996-01-04	4322.78	New York	Tester
7	David	Larry	1990-12-31	1998-02-12	7897.78	New York	Manager
8	James	Cat	1996-09-17	2002-04-15	1232.78	Vancouver	Tester

Pengisian tabel dapat menggunakan statement LOAD DATA INFILE dengan cara membuat file teks, misalkan bernama 'employee.txt' dan memisahkan kolom dengan **tab**. Selanjutnya memasukkan file dapat dilakukan sebagai berikut:

```
mysql> LOAD DATA INFILE '/app/mysql/employee.txt' INTO TABLE employee FIELDS TERMINATED
BY '\t';
Query OK, 8 rows affected (0.08 sec)
```

```
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
```

Selanjutnya, kita mulai menyalin tabel. Pada contoh di bawah, data yang akan disalin ke tabel yang baru adalah data employee yang mulai bekerja antara 1 Januari 1970 sampai 31 Desember 1990.

```
mysql> CREATE TABLE employee_copy AS
    -> SELECT *
    -> FROM employee
    -> WHERE start_date BETWEEN '1970-01-01' AND '1990-12-31';
Query OK, 6 rows affected (0.18 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
| employee        |
| employee_copy   |
+-----------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM employee_copy;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date    | salary   | city        |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6661.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6544.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2344.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2334.78 | Vancouver | Tester
|
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4322.78 | New York  | Tester
|
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7897.78 | New York  | Manager
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
6 rows in set (0.00 sec)
```

Membuat temporary tabel

Temporary tabel adalah tabel yang visible atau terlihat hanya pada koneksi yang sedang aktif dan akan langsung dihapus ketika koneksi sudah terputus.

```
mysql> CREATE TEMPORARY TABLE TEMPTBL(
    -> id INT PRIMARY KEY AUTO_INCREMENT,
    -> name VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
+-----------------+
9 rows in set (0.00 sec)

mysql> SELECT * FROM TEMPTBL;
Empty set (0.00 sec)

mysql> \q
Bye

mysql -h localhost -u root -p
Enter password:
mysql> USE PRAK2
Database changed

mysql> SELECT * FROM TEMPTBL;
ERROR 1146 (42S02): Table 'PRAK2.TEMPTBL' doesn't exist
```

Membuat tabel dengan foreign key

MySQL memiliki beberapa engine database. Salah satunya adalah engine InnoDB yang memiliki dukungan terhadap ACID. Pembuatan tabel menggunakan engine InnoDB dapat membantu memberikan *referential integrity*.

Berikut adalah contoh membuat tabel yang memiliki foreign key dengan engine InnoDB:

```
mysql> CREATE TABLE models (
    -> modelid smallint not null auto_increment,
```

```
    -> name varchar(40) not null,
    -> primary key (modelid)
    -> )
    -> engine=InnoDB;
Query OK, 0 rows affected (0.13 sec)

mysql> desc models;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| modelid | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| name    | varchar(40)          | NO   |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE orders (
    -> id SMALLINT NOT NULL PRIMARY KEY,
    -> modelid SMALLINT NOT NULL,
    -> description VARCHAR(40),
    -> FOREIGN KEY (modelid) REFERENCES models (modelid) ON UPDATE CASCADE ON DELETE
CASCADE
    -> ) engine=InnoDB;
Query OK, 0 rows affected (0.18 sec)

mysql> desc orders;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | smallint(6) | NO   | PRI | NULL    |       |
| modelid     | smallint(6) | NO   | MUL | NULL    |       |
| description | varchar(40) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Pada perintah sql di atas, kita membuat dua buah tabel dengan nama models dan orders. Tabel orders memiliki foreign key modelid yang bereferensi dengan primary key tabel models, yaitu modelid.

```
mysql> INSERT INTO models VALUES (1, 'testsatu');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT  * FROM  models;
+---------+----------+
| modelid | name     |
+---------+----------+
|       1 | testsatu |
```

```
+---------+----------+
1 row in set (0.00 sec)

mysql> insert into orders value (1, 1, 'deskripsisatu');
Query OK, 1 row affected (0.05 sec)

mysql> select * from orders;
+----+---------+---------------+
| id | modelid | description   |
+----+---------+---------------+
|  1 |       1 | deskripsisatu |
+----+---------+---------------+
1 row in set (0.00 sec)
```

Pada perintah sql di atas, kita memasukkan data ke tabel models dan orders. Data yang dimasukkan pada tabel orders memiliki referensi terhadap primary key pada tabel models, yaitu modelid yang nilainya adalah 1.

```
mysql> DELETE FROM models WHERE modelid = 1;
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM models;
Empty set (0.00 sec)

mysql> select * from orders;
Empty set (0.00 sec)
```

Pada perintah sql di atas, kita menghapus data di tabel models yang memiliki modelid = 1. Secara automatis, data pada tabel orders yang memiliki modelid = 1 juga ikut terhapus untuk memastikan *referential integrity*.

Memodifikasi tabel dengan ALTER TABLE

Tabel yang telah dibuat dapat dimodifikasi menggunakan command ALTER TABLE. Bentuk command untuk ALTER TABLE adalah:

```
ALTER [IGNORE] TABLE table_name specification [,specification]
```

dengan specification sebagai berikut:

```
ADD [COLUMN] column name (column definitions) [FIRST or AFTER column_name]
ADD INDEX [index_name] (column_list)
ADD PRIMARY KEY (column_list)
ADD UNIQUE [index_name] (column_list)
ALTER [COLUMN] column_name {SET DEFAULT default_value or DROP DEFAULT}
CHANGE [COLUMN] old_col_name create_definition
DROP [COLUMN] col_name
DROP PRIMARY KEY
DROP INDEX index_name
MODIFY [COLUMN] create_definition
```

```
RENAME [AS] new_tbl_name
```

Menambah kolom pada tabel

```
mysql> CREATE TABLE TABLE5(
    -> id int
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql> desc TABLE5;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> ALTER TABLE TABLE5
    -> ADD name VARCHAR(5);
Query OK, 0 rows affected (0.24 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc TABLE5;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| id    | int(11)    | YES  |     | NULL    |       |
| name  | varchar(5) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE TABLE5
    -> ADD first_name VARCHAR(30) AFTER id;
Query OK, 0 rows affected (0.84 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc TABLE5;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| first_name | varchar(30) | YES  |     | NULL    |       |
| name       | varchar(5)  | YES  |     | NULL    |       |
```

```
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Memodifikasi nama dan definisi tabel

```
mysql> ALTER TABLE TABLE5
    -> CHANGE name last_name VARCHAR(30);
Query OK, 0 rows affected (0.60 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc TABLE5;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| first_name | varchar(30) | YES  |     | NULL    |       |
| last_name  | varchar(30) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Memodifikasi definisi tabel

```
mysql> ALTER TABLE TABLE5
    -> MODIFY first_name VARCHAR(15),
    -> MODIFY last_name VARCHAR(15);
Query OK, 0 rows affected (0.24 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc TABLE5;+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| first_name | varchar(15) | YES  |     | NULL    |       |
| last_name  | varchar(15) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Menambahkan primary key

```
mysql> ALTER TABLE TABLE5 ADD PRIMARY KEY (id);
Query OK, 0 rows affected (0.58 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc TABLE5;
```

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | 0       |       |
| first_name | varchar(15) | YES  |     | NULL    |       |
| last_name  | varchar(15) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Menghapus kolom

```
mysql> ALTER TABLE TABLE5 DROP first_name;
Query OK, 0 rows affected (0.55 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC TABLE5;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | int(11)     | NO   | PRI | 0       |       |
| last_name | varchar(15) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Mengganti nama tabel

Mengganti nama tabel yang sudah dibuat dapat dilakukan dengan menggunakan perintah ALTER TABLE … RENAME. Berikut adalah contoh penggunaannya:

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
| employee        |
| employee_copy   |
+-----------------+
6 rows in set (0.00 sec)

mysql> ALTER TABLE employee_copy RENAME senior_employee;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
| employee        |
| senior_employee |
+-----------------+
6 rows in set (0.00 sec)
```

Menghapus semua data dalam tabel

Untuk menghapus semua data di dalam tabel digunakan command TRUNCATE sebagai berikut:

```
mysql> CREATE TABLE employee_copy AS
    -> SELECT * FROM employee;
Query OK, 8 rows affected (0.16 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM employee_copy;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city        |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1234.56 | Toronto     |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6661.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6544.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2344.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2334.78 | Vancouver | Tester
|
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4322.78 | New York  | Tester
|
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7897.78 | New York  | Manager
|
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1232.78 | Vancouver | Tester
|
```

```
+----+-----------+----------+-----------+-----------+---------+-----------
+-------------+
8 rows in set (0.00 sec)

mysql> TRUNCATE TABLE employee_copy;
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT * FROM employee_copy;
Empty set (0.00 sec)
```

Menghapus Tabel

Untuk menghapus tabel dari database, digunakan command DROP TABLE.

```
mysql> DROP TABLE employee_copy;
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
| TABLE5          |
| employee        |
| senior_employee |
+-----------------+
7 rows in set (0.00 sec)
```

Memasukkan data dengan INSERT

Memasukkan data ke dalam tabel dilakukan menggunakan command INSERT. Bentuk command INSERT dasar adalah sebagai berikut:

```
INSERT INTO <table_name>
VALUES (
value1,
value2,
etc......
);
```

Bentuk INSERT di atas bisa digunakan ketika jumlah value sudah sama dengan jumlah kolom yang akan diberikan data.

```
mysql> DESC employee;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| first_name  | varchar(15) | YES  |     | NULL    |                |
| last_name   | varchar(15) | YES  |     | NULL    |                |
| start_date  | date        | YES  |     | NULL    |                |
| end_date    | date        | YES  |     | NULL    |                |
| salary      | float(8,2)  | YES  |     | NULL    |                |
| city        | varchar(10) | YES  |     | NULL    |                |
| description | varchar(15) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
8 rows in set (0.01 sec)

mysql> INSERT INTO EMPLOYEE VALUES (
    -> 9,
    -> 'James',
    -> 'Bond',
    -> '1982-04-21',
    -> '2002-09-23',
    -> 1234.56,
    -> 'London',
    -> 'Spy'
    -> );
Query OK, 1 row affected (0.05 sec)
```

Tabel employee memiliki 8 kolom. Kita memasukkan 8 nilai menggunakan command INSERT. Satu buah nilai diisi ke kolom sesuai dengan urutan penulisan nilai.

Selain menggunakan INSERT seperti di atas, kita juga dapat menggunakan command INSERT sebagai berikut:

```
mysql> INSERT INTO employee
    -> (first_name, last_name, start_date, end_date, salary, city, description)
    -> VALUES
    -> ('Hercule',
    -> 'Poirot',
    -> '1973-05-23',
    -> '2001-08-09',
    -> 4312.98,
    -> 'Brussels',
    -> 'Detective'
    -> );
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM employee;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city        |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1234.56 | Toronto     |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6661.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6544.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2344.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2334.78 | Vancouver | Tester
|
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4322.78 | New York  | Tester
|
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7897.78 | New York  | Manager
|
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1232.78 | Vancouver | Tester
|
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1234.56 | London    | Spy
|
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4312.98 | Brussels  | Detective
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
10 rows in set (0.00 sec)
```

Menggunakan command INSERT di atas, kita memberikan spesifikasi kolom mana saja yang akan kita isi. Kolom yang tidak kita isi (dalam contoh ini adalah 'id') secara automatis akan diisi dengan nilai DEFAULT yang telah dispesifikasikan saat pembuatan tabel. Kolom 'id' kita spesifikasikan AUTO_INCREMENT

sehingga kolom tersebut akan diisi dengan increment dari nilai sebelumnya.

Memasukkan lebih dari satu baris dalam satu INSERT

Memasukkan lebih dari satu baris dapat dilakukan menggunakan satu buah command INSERT. Hal ini dilakukan dengan memisahkan input tiap baris dengan koma (','). Berikut adalah contohnya:

```
mysql> INSERT INTO employee (first_name, last_name, start_date, end_date, salary, city,
description) VALUES
   ->   ('Lincoln',   'Rhyme',   '1999-05-25',   '2011-07-13',   3212.98,   'New   York',
'Forensics' ),
   -> ('Sherlock','Holmes', '1923-08-12','1945-07-21',4123.21, 'London', 'Detective' ) ;
Query OK, 2 rows affected (0.35 sec)
Records: 2  Duplicates: 0  Warnings: 0


mysql> SELECT * FROM employee;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date    | salary   | city        |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|   1 | Jason        | Martin     | 1996-07-25 | 2006-07-25 | 1234.56 | Toronto    |
Programmer  |
|  2 | Alison     | Mathews    | 1976-03-21 | 1986-02-21 | 6661.78 | Vancouver | Tester
|
|  3 | James      | Smith      | 1978-12-12 | 1990-03-15 | 6544.78 | Vancouver | Tester
|
|  4 | Celia      | Rice       | 1982-10-24 | 1999-04-21 | 2344.78 | Vancouver | Manager
|
|  5 | Robert     | Black      | 1984-01-15 | 1998-08-08 | 2334.78 | Vancouver | Tester
|
|  6 | Linda      | Green      | 1987-07-30 | 1996-01-04 | 4322.78 | New York  | Tester
|
|  7 | David      | Larry      | 1990-12-31 | 1998-02-12 | 7897.78 | New York  | Manager
|
|  8 | James      | Cat        | 1996-09-17 | 2002-04-15 | 1232.78 | Vancouver | Tester
|
|   9 | James       | Bond       | 1982-04-21 | 2002-09-23 | 1234.56 | London     | Spy
|
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4312.98 | Brussels  | Detective
|
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3212.98 | New York  | Forensics
|
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4123.21 | London    | Detective
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
```

```
12 rows in set (0.00 sec)
```

INSERT menggunakan data dari tabel lain

Kita juga dapat memasukkan data dari tabel yang lain menggunakan command INSERT … INTO … SELECT.

```
mysql> CREATE TABLE employee2 (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(15),
    -> city VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql> DESC employee2;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(15) | YES  |     | NULL    |                |
| city  | varchar(20) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> INSERT INTO employee2
    -> (id, name, city)
    -> SELECT id, CONCAT(first_name,'  ',last_name), city
    -> FROM employee;
Query OK, 12 rows affected (0.05 sec)

mysql> SELECT * FROM employee2;
+----+-----------------+-----------+
| id | name            | city      |
+----+-----------------+-----------+
|  1 | Jason Martin    | Toronto   |
|  2 | Alison Mathews  | Vancouver |
|  3 | James Smith     | Vancouver |
|  4 | Celia Rice      | Vancouver |
|  5 | Robert Black    | Vancouver |
|  6 | Linda Green     | New York  |
|  7 | David Larry     | New York  |
|  8 | James Cat       | Vancouver |
|  9 | James Bond      | London    |
| 10 | Hercule Poirot  | Brussels  |
```

```
| 11 | Lincoln Rhyme   | New York  |
| 12 | Sherlock Holmes | London    |
+----+-----------------+-----------+
12 rows in set (0.00 sec)
```

Pada command di atas, kita hanya memilih id, nama, dan kota dari tabel employee untuk dimasukkan ke dalam tabel employee2.

Mengubah data menggunakan UPDATE

Data yang telah kita masukkan ke dalam tabel dapat kita ubah menggunakan command UPDATE. Bentuk umum command UPDATE adalah sebagai berikut:

```
UPDATE <table_name>
SET <column_name> = 'new_value'
WHERE (<column_name> = 'some_value');
```

Berikut adalah contoh menggunakan UPDATE:

```
mysql> UPDATE employee2
    -> SET city = 'Ottawa'
    -> WHERE city = 'Vancouver';
Query OK, 5 rows affected (0.08 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM employee2;
+----+-----------------+----------+
| id | name            | city     |
+----+-----------------+----------+
|  1 | Jason Martin    | Toronto  |
|  2 | Alison Mathews  | Ottawa   |
|  3 | James Smith     | Ottawa   |
|  4 | Celia Rice      | Ottawa   |
|  5 | Robert Black    | Ottawa   |
|  6 | Linda Green     | New York |
|  7 | David Larry     | New York |
|  8 | James Cat       | Ottawa   |
|  9 | James Bond      | London   |
| 10 | Hercule Poirot  | Brussels |
| 11 | Lincoln Rhyme   | New York |
| 12 | Sherlock Holmes | London   |
+----+-----------------+----------+
12 rows in set (0.00 sec)
```

Mengubah nilai berdasarkan nilai yang dimiliki sekarang

```
mysql> SELECT salary FROM employee;
```

```
+---------+
| salary  |
+---------+
| 1234.56 |
| 6661.78 |
| 6544.78 |
| 2344.78 |
| 2334.78 |
| 4322.78 |
| 7897.78 |
| 1232.78 |
| 1234.56 |
| 4312.98 |
| 3212.98 |
| 4123.21 |
+---------+
12 rows in set (0.00 sec)

mysql> UPDATE employee SET salary = salary + 1;
Query OK, 12 rows affected (0.06 sec)
Rows matched: 12  Changed: 12  Warnings: 0

mysql> SELECT salary FROM employee;
+---------+
| salary  |
+---------+
| 1235.56 |
| 6662.78 |
| 6545.78 |
| 2345.78 |
| 2335.78 |
| 4323.78 |
| 7898.78 |
| 1233.78 |
| 1235.56 |
| 4313.98 |
| 3213.98 |
| 4124.21 |
+---------+
12 rows in set (0.00 sec)
```

Mengganti data dari tabel

Data yang kita sudah masukkan dalam tabel dapat kita ganti dengan menggunakan command REPLACE.

Format penggunaan command REPLACE sama dengan INSERT, namun ia menggantikan data yang memiliki nilai sama di indeks PRIMARY KEY atau UNIQUE.

```
mysql> SELECT * FROM employee2;
+----+-----------------+----------+
| id | name            | city     |
+----+-----------------+----------+
|  1 | Jason Martin    | Toronto  |
|  2 | Alison Mathews  | Ottawa   |
|  3 | James Smith     | Ottawa   |
|  4 | Celia Rice      | Ottawa   |
|  5 | Robert Black    | Ottawa   |
|  8 | James Cat       | Ottawa   |
|  9 | James Bond      | London   |
| 10 | Hercule Poirot  | Brussels |
| 12 | Sherlock Holmes | London   |
+----+-----------------+----------+
9 rows in set (0.00 sec)

mysql> REPLACE INTO employee2 (id,name, city) VALUES (12,'Sherlock Holmes','Manchester');
Query OK, 2 rows affected (0.05 sec)

mysql> SELECT * FROM employee2;
+----+-----------------+------------+
| id | name            | city       |
+----+-----------------+------------+
|  1 | Jason Martin    | Toronto    |
|  2 | Alison Mathews  | Ottawa     |
|  3 | James Smith     | Ottawa     |
|  4 | Celia Rice      | Ottawa     |
|  5 | Robert Black    | Ottawa     |
|  8 | James Cat       | Ottawa     |
|  9 | James Bond      | London     |
| 10 | Hercule Poirot  | Brussels   |
| 12 | Sherlock Holmes | Manchester |
+----+-----------------+------------+
9 rows in set (0.00 sec)
```

Menghapus data dari tabel

Menghapus data tertentu dari sebuah tabel dalam database dapat dilakukan menggunakan command

DELETE. Berikut adalah bentuk dasar dari command DELETE:

```
DELETE FROM <table_name>
WHERE (<column_name> = 'some_value');
```

Berikut adalah contoh menghapus data menggunakan DELETE:

```
mysql> SELECT * FROM employee2;
+----+-----------------+----------+
| id | name            | city     |
+----+-----------------+----------+
|  1 | Jason Martin    | Toronto  |
|  2 | Alison Mathews  | Ottawa   |
|  3 | James Smith     | Ottawa   |
|  4 | Celia Rice      | Ottawa   |
|  5 | Robert Black    | Ottawa   |
|  6 | Linda Green     | New York |
|  7 | David Larry     | New York |
|  8 | James Cat       | Ottawa   |
|  9 | James Bond      | London   |
| 10 | Hercule Poirot  | Brussels |
| 11 | Lincoln Rhyme   | New York |
| 12 | Sherlock Holmes | London   |
+----+-----------------+----------+
12 rows in set (0.00 sec)

mysql> DELETE FROM employee2 WHERE city = 'New York';
Query OK, 3 rows affected (0.39 sec)

mysql> SELECT * FROM employee2;
+----+-----------------+----------+
| id | name            | city     |
+----+-----------------+----------+
|  1 | Jason Martin    | Toronto  |
|  2 | Alison Mathews  | Ottawa   |
|  3 | James Smith     | Ottawa   |
|  4 | Celia Rice      | Ottawa   |
|  5 | Robert Black    | Ottawa   |
|  8 | James Cat       | Ottawa   |
|  9 | James Bond      | London   |
| 10 | Hercule Poirot  | Brussels |
| 12 | Sherlock Holmes | London   |
+----+-----------------+----------+
9 rows in set (0.00 sec)
```

MODUL TIGA

Pertemuan 3 Query

SELECT Statement

Mengambil data dengan SELECT

Data yang telah kita simpan dalam tabel dapat kita ambil menggunakan statement SELECT. Bentuk dasar statement SELECT adalah

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

- what_to_select adalah informasi apa yang ingin kita lihat, biasanya berupa sekumpulan kolom. Karakter "*" (bintang) dapat digunakan untuk menampilkan semua kolom.
- which_table menunjukkan dari tabel mana informasi tersebut akan kita ambil.
- Klause WHERE bersifat opsional, diikuti dengan conditions_to_satisfy yang menunjukkan kondisi yang harus dipenuhi oleh sebuah baris informasi agar dapat dipilih.

Contoh penggunaan SELECT adalah sebagai berikut:

```
mysql> use PRAK2
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-----------------+
| Tables_in_PRAK2 |
+-----------------+
| TABLE1          |
| TABLE2          |
| TABLE3          |
| TABLE4          |
| TABLE5          |
| cars            |
| employee        |
| employee2       |
| senior_employee |
+-----------------+
9 rows in set (0.00 sec)

mysql> SELECT * FROM employee;
+----+-----------+-----------+------------+------------+---------+-----------
+-------------+
```

```
| id | first_name | last_name | start_date | end_date   | salary  | city      |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver | Tester
|
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  | Tester
|
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  | Manager
|
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver | Tester
|
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    | Spy
|
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  | Detective
|
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  | Forensics
|
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    | Detective
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
12 rows in set (0.00 sec)
```

Pada contoh di atas, kita menggunakan tabel employee yang telah kita buat pada Pertemuan 2 kemarin.

Perintah `SELECT * FROM employee` menampilkan semua baris data dan kolom dalam tabel employee.

Mengambil data pada kolom tertentu

Untuk mengambil data hanya pada kolom tertentu saja kita dapat menggunakan command SELECT sebagai berikut:

```
mysql> SELECT first_name, last_name, city FROM employee;
+------------+-----------+-----------+
| first_name | last_name | city      |
+------------+-----------+-----------+
| Jason      | Martin    | Toronto   |
| Alison     | Mathews   | Vancouver |
| James      | Smith     | Vancouver |
| Celia      | Rice      | Vancouver |
| Robert     | Black     | Vancouver |
```

```
| Linda      | Green     | New York  |
| David      | Larry     | New York  |
| James      | Cat       | Vancouver |
| James      | Bond      | London    |
| Hercule    | Poirot    | Brussels  |
| Lincoln    | Rhyme     | New York  |
| Sherlock   | Holmes    | London    |
+------------+-----------+-----------+
12 rows in set (0.00 sec)
```

Query menggunakan parameter kondisi WHERE

Statement WHERE dapat digunakan untuk memfilter data yang ingin kita ambil. Berikut adalah beberapa contoh penggunaan parameter kondisi WHERE.

```
mysql> SELECT first_name, last_name, city FROM employee WHERE city = 'New York';
+------------+-----------+----------+
| first_name | last_name | city     |
+------------+-----------+----------+
| Linda      | Green     | New York |
| David      | Larry     | New York |
| Lincoln    | Rhyme     | New York |
+------------+-----------+----------+
3 rows in set (0.00 sec)
```

Query menggunakan beberapa parameter kondisional

Kita dapat memilih data menggunakan beberapa kombinasi parameter kondisional dihubungkan dengan statement AND atau OR. Statement AND dapat juga ditulis sebagai '&&', sedangkan statement OR juga dapat ditulis sebagai '||'. Statement AND memiliki **precedence yang lebih tinggi** dibandingkan statement OR.

```
mysql> SELECT first_name, last_name, salary, city FROM employee WHERE city = 'New York'
AND salary > 4000  ;
+------------+-----------+---------+----------+
| first_name | last_name | salary  | city     |
+------------+-----------+---------+----------+
| Linda      | Green     | 4323.78 | New York |
| David      | Larry     | 7898.78 | New York |
+------------+-----------+---------+----------+
2 rows in set (0.00 sec)

mysql> SELECT first_name, last_name, city, description
    -> FROM employee
    -> WHERE city = 'Toronto' OR description = 'Tester';
+------------+-----------+-----------+-------------+
```

```
| first_name | last_name | city      | description |
+------------+-----------+-----------+-------------+
| Jason      | Martin    | Toronto   | Programmer  |
| Alison     | Mathews   | Vancouver | Tester      |
| James      | Smith     | Vancouver | Tester      |
| Robert     | Black     | Vancouver | Tester      |
| Linda      | Green     | New York  | Tester      |
| James      | Cat       | Vancouver | Tester      |
+------------+-----------+-----------+-------------+
6 rows in set (0.00 sec)
```

Memberikan alias hasil query pada SELECT

Kita dapat memberikan alias pada kolom hasil pencarian dengan menggunakan keyword AS. Berikut adalah contohnya:

```
mysql> SELECT CONCAT(first_name," ",last_name) AS name, description
    -> FROM employee
    -> WHERE description = "Detective";
+-----------------+-------------+
| name            | description |
+-----------------+-------------+
| Hercule Poirot  | Detective   |
| Sherlock Holmes | Detective   |
+-----------------+-------------+
2 rows in set (0.00 sec)
```

Query data bertipe teks dengan pattern matching

Pattern matching dapat kita gunakan untuk memilih data bertipe teks dengan karakteristik tertentu. Command yang digunakan untuk melakukan pencocokan adalah LIKE dan NOT LIKE. Berikut adalah beberapa fasilitas pattern matching yang disediakan oleh MySQL.

Simbol	Fungsi
_	Match any single character
%	Match an arbitrary number of character (including no character)

Berikut adalah contoh penggunaan pattern matching untuk memilih data:

```
mysql> SELECT CONCAT(first_name," ", last_name)
    -> FROM employee
    -> WHERE first_name LIKE 'J____';
+----------------------------------+
| CONCAT(first_name," ", last_name) |
+----------------------------------+
| Jason Martin                     |
```

```
| James Smith                       |
| James Cat                         |
| James Bond                        |
+-----------------------------------+
4 rows in set (0.00 sec)
```

Contoh di atas menunjukkan bagaimana memilih employee yang bernama depan diawali dengan huruf "J" dan diikuti oleh tepat 4 buah karakter apapun.

```
mysql> SELECT CONCAT(first_name," ", last_name)
    -> FROM employee
    -> WHERE first_name NOT LIKE '%n';
+-----------------------------------+
| CONCAT(first_name," ", last_name) |
+-----------------------------------+
| James Smith                       |
| Celia Rice                        |
| Robert Black                      |
| Linda Green                       |
| David Larry                       |
| James Cat                         |
| James Bond                        |
| Hercule Poirot                    |
| Sherlock Holmes                   |
+-----------------------------------+
9 rows in set (0.00 sec)
```

Contoh di atas menunjukkan bagaimana memilih employee yang bernama depan **tidak diakhiri** dengan karakter "n".

Query data unik menggunakan DISTINCT

DISTINCT digunakan untuk menghilangkan duplikasi dari data yang dicari sehingga didapatkan data yang unik (hanya muncul satu kali). Berikut adalah contohnya.

```
mysql> SELECT description FROM employee;
+-------------+
| description |
+-------------+
| Programmer  |
| Tester      |
| Tester      |
| Manager     |
| Tester      |
| Tester      |
| Manager     |
| Tester      |
```

```
| Spy         |
| Detective   |
| Forensics   |
| Detective   |
+-------------+
12 rows in set (0.00 sec)

mysql> SELECT DISTINCT description FROM employee;
+-------------+
| description |
+-------------+
| Programmer  |
| Tester      |
| Manager     |
| Spy         |
| Detective   |
| Forensics   |
+-------------+
6 rows in set (0.00 sec)
```

Pada contoh di atas, ketika statement DISTINCT tidak digunakan, ditampilkan semua deskripsi dari tabel employee. Ada deskripsi yang keluar lebih dari satu kali, misalnya “Programmer”. Namun, penggunaan statement DISTINCT dalam query menampilkan data tanpa duplikasi, semua data hanya keluar satu kali.

Membatasi hasil query dengan LIMIT

Data yang dihasilkan dari query yang kita masukkan dapat kita batasi menggunakan statement LIMIT. Berikut adalah contohnya.

```
mysql> SELECT *
    -> FROM employee
    -> LIMIT 5;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver | Tester
```

```
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
5 rows in set (0.00 sec)
```

Pada contoh query di atas, kita membatasi jumlah data yang ditampilkan sebanyak lima data saja menggunakan statement LIMIT 5.

Kita juga dapat membatasi disertai pemilihan batasan tersebut ditampilkan mulai data keberapa, masih menggunakan statement LIMIT. Berikut adalah contohnya.

```
mysql> SELECT *
    -> FROM employee
    -> LIMIT 2,3;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver | Tester
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
3 rows in set (0.00 sec)
```

Pada contoh query di atas, kita membatasi data yang ditampilkan dimulai dari data ke-2 sebanyak 3 data. Hal yang perlu diperhatikan, urutan data dimulai dari urutan ke-0. Sehingga jika kita menampilkan data menggunakan LIMIT 0,3 akan terlihat sebagai berikut:

```
mysql> SELECT *  FROM employee LIMIT 0,3;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver | Tester
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
3 rows in set (0.00 sec)
```

Mengelompokkan hasil query menggunakan GROUP BY

Hasil query dapat kita kelompokkan berdasarkan field/kolom menggunakan statement GROUP BY. Berikut adalah contohnya:

```
mysql> SELECT * FROM employee GROUP BY city;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  | Detective
|
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    | Spy
|
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  | Tester
|
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver | Tester
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
5 rows in set (0.00 sec)
```

Contoh query di atas menunjukkan pengelompokkan berdasarkan “city”. Hasil query menunjukkan data yang ditampilkan adalah data yang pertama untuk setiap kelompok “city”. Hal yang perlu diperhatikan adalah data yang ditampilkan terurut berdasarkan data pada kolom “city”.

Mendapatkan jumlah anggota setiap kelompok menggunakan COUNT( )

Query menggunakan GROUP BY hanya menunjukkan data pertama yang muncul. Jika kita ingin mendapatkan jumlah anggota setiap kelompok, kita bisa menggunakan fungsi COUNT(). Berikut adalah contoh penggunaannya.

```
mysql> SELECT city, COUNT(*) FROM employee GROUP BY city;
+-----------+----------+
| city      | COUNT(*) |
+-----------+----------+
| Brussels  |        1 |
| London    |        2 |
| New York  |        3 |
| Toronto   |        1 |
| Vancouver |        5 |
+-----------+----------+
5 rows in set (0.00 sec)
```

Hasil query di atas menunjukkan jumlah employee di setiap kotanya. Kita juga dapat melakukan kombinasi GROUP BY dengan parameter kondisi sebagai berikut:

```
mysql> SELECT city, COUNT(*)
    -> FROM employee
    -> WHERE description = 'Tester'
    -> GROUP BY city;
+-----------+----------+
| city      | COUNT(*) |
+-----------+----------+
| New York  |        1 |
| Vancouver |        4 |
+-----------+----------+
2 rows in set (0.00 sec)
```

Hasil query di atas menunjukkan jumlah “Tester” yang ada di setiap kota.

Parameter kondisional dengan HAVING

Statement HAVING merupakan parameter kondisional seperti WHERE.yang bertindak sebagai pembatas sekunder dari hasil query. Statemen HAVING biasanya digunakan untuk pembatas sekunder setelah statement GROUP BY, walau bisa saja digunakan tanpa menggunakan GROUP BY.

Berikut adalah contoh penggunaan parameter HAVING:

```
mysql> SELECT first_name, last_name, salary
    -> FROM employee
    -> HAVING salary > 3000;
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| Alison     | Mathews   | 6662.78 |
| James      | Smith     | 6545.78 |
| Linda      | Green     | 4323.78 |
| David      | Larry     | 7898.78 |
| Hercule    | Poirot    | 4313.98 |
| Lincoln    | Rhyme     | 3213.98 |
| Sherlock   | Holmes    | 4124.21 |
+------------+-----------+---------+
7 rows in set (0.00 sec)
```

Query di atas menunjukkan parameter HAVING dapat digunakan seperti parameter WHERE.

```
mysql> SELECT city, COUNT(*), salary FROM employee WHERE salary > 3000 GROUP BY city;
+-----------+----------+---------+
| city      | COUNT(*) | salary  |
+-----------+----------+---------+
| Brussels  |        1 | 4313.98 |
| London    |        1 | 4124.21 |
| New York  |        3 | 4323.78 |
| Vancouver |        2 | 6662.78 |
```

```
+-----------+----------+---------+
4 rows in set (0.00 sec)

mysql> SELECT city, COUNT(*), salary FROM employee GROUP BY city HAVING salary > 3000;
+-----------+----------+---------+
| city      | COUNT(*) | salary  |
+-----------+----------+---------+
| Brussels  |        1 | 4313.98 |
| New York  |        3 | 4323.78 |
| Vancouver |        5 | 6662.78 |
+-----------+----------+---------+
3 rows in set (0.00 sec)
```

Contoh query di atas menunjukkan perbedaan urutan dijalankannya filtering, sehingga didapatkan hasil data yang berbeda. Query yang pertama melakukan pemilihan salary > 3000 terlebih dahulu sebelum kemudian dikelompokkan berdasarkan salary. Sedangkan query kedua melakukan pengelompokan terlebih dahulu terhadap salary. Pengelompokan tersebut menyebabkan data pertama untuk setiap kelompok saja yang terpilih. Ketika parameter HAVING dijalankan, query hanya akan menampilkan data pertama untuk setiap kelompok yang memiliki salary > 3000.

Penggunaan HAVING setelah ORDER BY memerlukan nama kolom yang akan difilter menggunakan HAVING ikut dipilih. Jika tidak akan muncul pesan error seperti di bawah ini:

```
mysql> SELECT city, COUNT(*) FROM employee GROUP BY city HAVING salary > 3000;
ERROR 1054 (42S22): Unknown column 'salary' in 'having clause'
```

Mengurutkan hasil query menggunakan ORDER BY

Hasil query dapat kita urutkan berdasarkan field/kolom tertentu menggunakan ORDER BY. Statement ASC dan DESC dapat kita gunakan untuk mendapatkan pengurutan naik atau turun. Berikut adalah contoh penggunaannya.

```
mysql> SELECT CONCAT(first_name," ",last_name) AS name
    -> FROM employee
    -> ORDER BY name;
+-----------------+
| name            |
+-----------------+
| Alison Mathews  |
| Celia Rice      |
| David Larry     |
| Hercule Poirot  |
| James Bond      |
| James Cat       |
| James Smith     |
| Jason Martin    |
| Lincoln Rhyme   |
| Linda Green     |
```

```
| Robert Black    |
| Sherlock Holmes |
+-----------------+
12 rows in set (0.00 sec)

mysql> SELECT CONCAT(first_name," ",last_name) AS name
    -> FROM employee
    -> ORDER BY name;
+-----------------+
| name            |
+-----------------+
| Sherlock Holmes |
| Robert Black    |
| Linda Green     |
| Lincoln Rhyme   |
| Jason Martin    |
| James Smith     |
| James Cat       |
| James Bond      |
| Hercule Poirot  |
| David Larry     |
| Celia Rice      |
| Alison Mathews  |
+-----------------+
12 rows in set (0.00 sec)
```

Contoh query yang pertama, kita mengurutkan employee berdasarkan nama. Secara default, ORDER BY menggunakan urutan naik. Kita bisa menggunakan DESC untuk mendapatkan urutan turun seperti yang ditunjukkan oleh contoh query yang kedua.

Mengurutkan hasil query berdasarkan lebih dari satu kolom

Pengurutan hasil query dapat dilakukan berdasarkan lebih dari satu kolom. Statement urutan (ASC dan DESC) melekat pada kolom yang mendahuluinya. Berikut adalah contohnya:

```
mysql> SELECT first_name, last_name, city FROM employee ORDER BY first_name, city;
+------------+-----------+-----------+
| first_name | last_name | city      |
+------------+-----------+-----------+
| Alison     | Mathews   | Vancouver |
| Celia      | Rice      | Vancouver |
| David      | Larry     | New York  |
| Hercule    | Poirot    | Brussels  |
| James      | Bond      | London    |
| James      | Smith     | Vancouver |
```

```
| James      | Cat       | Vancouver |
| Jason      | Martin    | Toronto   |
| Lincoln    | Rhyme     | New York  |
| Linda      | Green     | New York  |
| Robert     | Black     | Vancouver |
| Sherlock   | Holmes    | London    |
+------------+-----------+-----------+
12 rows in set (0.00 sec)
```

Contoh query di atas menunjukkan pengurutan berdasarkan first_name terlebih dahulu sebelum pengurutan berdasarkan city. Jika ada data dengan first_name yang sama, baru data tersebut diurutkan berdasarkan city. Untuk lebih jelasnya, mari kita bandingkan dengan contoh query di bawah ini:

```
mysql> SELECT first_name, last_name, city FROM employee ORDER BY first_name, city DESC;
+------------+-----------+-----------+
| first_name | last_name | city      |
+------------+-----------+-----------+
| Alison     | Mathews   | Vancouver |
| Celia      | Rice      | Vancouver |
| David      | Larry     | New York  |
| Hercule    | Poirot    | Brussels  |
| James      | Smith     | Vancouver |
| James      | Cat       | Vancouver |
| James      | Bond      | London    |
| Jason      | Martin    | Toronto   |
| Lincoln    | Rhyme     | New York  |
| Linda      | Green     | New York  |
| Robert     | Black     | Vancouver |
| Sherlock   | Holmes    | London    |
+------------+-----------+-----------+
12 rows in set (0.00 sec)
```

Contoh query di atas menunjukkan pengurutan berdasarkan first_name terlebih dahulu dilanjutkan pengurutan menurun terhadap city. Data employee yang sama, yaitu yang memiliki first_name James selanjutnya diurutkan menurun berdasarkan city. Akibatnya employee first_name yang berada di city Vancouver ditampilkan terlebih dahulu dibandingkan yang berada di London. Bandingkan dengan hasil query yang sebelumnya.

Kombinasi ORDER BY dengan LIMIT

Ketika statement ORDER BY dikombinasikan dengan LIMIT, maka statement ORDER BY yang dieksekusi terlebih dahulu, baru kemudian LIMIT dilakukan untuk membatasi jumlah hasil query yang ditampilkan.

```
mysql> SELECT first_name, last_name, city FROM employee ORDER BY city LIMIT 4;
+------------+-----------+----------+
| first_name | last_name | city     |
+------------+-----------+----------+
| Hercule    | Poirot    | Brussels |
```

```
| Sherlock   | Holmes    | London   |
| James      | Bond      | London   |
| Lincoln    | Rhyme     | New York |
+------------+-----------+----------+
4 rows in set (0.00 sec)
```

Operator BETWEEN

Operator BETWEEN digunakan untuk memfilter data yang bernilai di antara dua buah nilai yang dispesifikasikan. Berikut adalah contoh penggunaannya.

```
mysql> SELECT first_name, last_name, salary FROM employee
    -> WHERE salary BETWEEN 1000 and 3000;
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| Jason      | Martin    | 1235.56 |
| Celia      | Rice      | 2345.78 |
| Robert     | Black     | 2335.78 |
| James      | Cat       | 1233.78 |
| James      | Bond      | 1235.56 |
+------------+-----------+---------+
5 rows in set (0.00 sec)
```

Hasil query di atas menunjukkan employee dengan pendapatan antara 1000 dan 3000. Operator BETWEEN value1 AND value2 memiliki nilai yang ekivalen dengan value1 ≤ data < value2.

```
mysql> SELECT first_name, last_name, salary FROM employee WHERE salary BETWEEN 1233.78 and 1235.56;
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| James      | Cat       | 1233.78 |
+------------+-----------+---------+
1 row in set (0.00 sec)

mysql> SELECT first_name, last_name, salary FROM employee WHERE salary BETWEEN 1233.78 and 1235.57;
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| Jason      | Martin    | 1235.56 |
| James      | Cat       | 1233.78 |
| James      | Bond      | 1235.56 |
+------------+-----------+---------+
3 rows in set (0.00 sec)
```

BETWEEN juga dapat digunakan untuk memfilter nilai alfanumerik.

```
mysql> SELECT first_name,last_name FROM employee WHERE first_name BETWEEN "Elvis" and "James";
+------------+-----------+
| first_name | last_name |
```

```
+------------+-----------+
| James      | Smith     |
| James      | Cat       |
| James      | Bond      |
| Hercule    | Poirot    |
+------------+-----------+
4 rows in set (0.00 sec)
```

Negasi dari BETWEEN adalah NOT BETWEEN.

```
mysql> SELECT first_name,last_name FROM employee WHERE first_name NOT BETWEEN "Elvis" and "James";
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Jason      | Martin    |
| Alison     | Mathews   |
| Celia      | Rice      |
| Robert     | Black     |
| Linda      | Green     |
| David      | Larry     |
| Lincoln    | Rhyme     |
| Sherlock   | Holmes    |
+------------+-----------+
8 rows in set (0.00 sec)
```

SUB QUERY

Sub query adalah statement SELECT di dalam statement SELECT. Sebuah sub query dinyatakan di dalam tanda kurung (). Statement di luar statement sub query adalah salah satu dari statement SELECT, INSERT, UPDATE, DELETE, SET, atau DO.

```
mysql> SELECT first_name, last_name, salary
    -> FROM employee
    -> WHERE salary = (SELECT MAX(salary) FROM employee);
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| David      | Larry     | 7898.78 |
+------------+-----------+---------+
1 row in set (0.00 sec)
```

Operator penghubung "=" hanya dapat digunakan untuk menerima satu buah nilai, sehingga ketika hasil dari sub query berisi lebih dari satu data, maka akan terjadi pesan error.

```
mysql> SELECT first_name, last_name, city
    -> FROM employee
    -> WHERE city = (SELECT city FROM employee WHERE city LIKE "Vancouver");
ERROR 1242 (21000): Subquery returns more than 1 row
```

Sub query dengan ALL

Command ALL diikuti dengan operator perbandingan digunakan memiliki arti menampilkan nilai jika perbandingan bernilai benar untuk semua data. Berikut adalah contoh penggunaannya.

```
mysql> SELECT first_name, last_name, salary
    -> FROM employee
    -> WHERE salary > ALL(SELECT salary FROM employee WHERE salary <2000);
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| Alison     | Mathews   | 6662.78 |
| James      | Smith     | 6545.78 |
| Celia      | Rice      | 2345.78 |
| Robert     | Black     | 2335.78 |
| Linda      | Green     | 4323.78 |
| David      | Larry     | 7898.78 |
| Hercule    | Poirot    | 4313.98 |
| Lincoln    | Rhyme     | 3213.98 |
| Sherlock   | Holmes    | 4124.21 |
+------------+-----------+---------+
9 rows in set (0.00 sec)
```

Query di atas menghasilkan employee yang memiliki salary lebih besar dari semua employee yang

memiliki salary kurang dari 2000.

Sub query dengan ANY

Command ANY diikuti dengan operator perbandingan memiliki arti menampilkan nilai yang sesuai dengan apapun yang dihasilkan oleh sub query. Alias dari ANY adalah SOME. Berikut adalah contohnya:

```
mysql> SELECT first_name, last_name, salary FROM employee WHERE salary > ANY(SELECT
salary FROM employee WHERE salary <2000);
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| Jason      | Martin    | 1235.56 |
| Alison     | Mathews   | 6662.78 |
| James      | Smith     | 6545.78 |
| Celia      | Rice      | 2345.78 |
| Robert     | Black     | 2335.78 |
| Linda      | Green     | 4323.78 |
| David      | Larry     | 7898.78 |
| James      | Bond      | 1235.56 |
| Hercule    | Poirot    | 4313.98 |
| Lincoln    | Rhyme     | 3213.98 |
| Sherlock   | Holmes    | 4124.21 |
+------------+-----------+---------+
11 rows in set (0.00 sec)
```

Hasil sub query pada contoh di atas adalah semua employee yang memiliki salary kurang dari 2000. Hasil query keseluruhan menampilkan semua employee yang memiliki salary yang lebih besar dari hasil sub query. Employee dengan salary terendah, yaitu 1233.78 tidak ditampilkan karena tidak ada salary lain yang lebih kecil dari dia. Contoh query dengan SOME adalah sebagai berikut:

```
mysql> SELECT first_name, last_name, salary FROM employee WHERE salary > SOME(SELECT
salary FROM employee WHERE salary <2000);
+------------+-----------+---------+
| first_name | last_name | salary  |
+------------+-----------+---------+
| Jason      | Martin    | 1235.56 |
| Alison     | Mathews   | 6662.78 |
| James      | Smith     | 6545.78 |
| Celia      | Rice      | 2345.78 |
| Robert     | Black     | 2335.78 |
| Linda      | Green     | 4323.78 |
| David      | Larry     | 7898.78 |
| James      | Bond      | 1235.56 |
| Hercule    | Poirot    | 4313.98 |
| Lincoln    | Rhyme     | 3213.98 |
| Sherlock   | Holmes    | 4124.21 |
```

```
+------------+-----------+---------+
11 rows in set (0.00 sec)
```

Sub query dengan EXISTS

Sebuah sub query dengan EXISTS memberikan nilai TRUE jika sub query tersebut memiliki hasil. Jika sub query bernilai TRUE, maka query utama akan dijalankan. Negasi dari sub query EXISTS adalah NOT EXISTS. Berikut adalah contohnya:

```
mysql> SELECT first_name, last_name, city FROM employee WHERE EXISTS (SELECT * FROM
employee WHERE city = "Toronto");
+------------+-----------+-----------+
| first_name | last_name | city      |
+------------+-----------+-----------+
| Jason      | Martin    | Toronto   |
| Alison     | Mathews   | Vancouver |
| James      | Smith     | Vancouver |
| Celia      | Rice      | Vancouver |
| Robert     | Black     | Vancouver |
| Linda      | Green     | New York  |
| David      | Larry     | New York  |
| James      | Cat       | Vancouver |
| James      | Bond      | London    |
| Hercule    | Poirot    | Brussels  |
| Lincoln    | Rhyme     | New York  |
| Sherlock   | Holmes    | London    |
+------------+-----------+-----------+
12 rows in set (0.00 sec)

mysql> SELECT first_name, last_name, city FROM employee WHERE NOT EXISTS (SELECT * FROM
employee WHERE city = "Toronto");
Empty set (0.00 sec)
```

Sub query dengan IN

Seperti yang telah disebutkan sebelumnya, operator “=” hanya dapat digunakan ketika hasil sub query memiliki hasil tepat satu. Jika hasil sub query berjumlah lebih dari satu data, maka kita dapat menggunakan IN. Berikut adalah contohnya:

```
mysql> CREATE TABLE job (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> title VARCHAR(20));
Query OK, 0 rows affected (0.14 sec)

mysql> INSERT into job VALUES
    -> (1, 'Tester'),
```

```
    -> (2, 'Accountant') ,
    -> (3, 'Programmer'),
    -> (4, 'Professor');
Query OK, 4 rows affected (0.05 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * from job;
+----+------------+
| id | title      |
+----+------------+
|  1 | Tester     |
|  2 | Accountant |
|  3 | Programmer |
|  4 | Professor  |
+----+------------+
4 rows in set (0.00 sec)
mysql> SELECT first_name, last_name, description
    -> FROM employee
    -> WHERE description IN
    -> (SELECT title FROM job
    -> );
+------------+-----------+-------------+
| first_name | last_name | description |
+------------+-----------+-------------+
| Jason      | Martin    | Programmer  |
| Alison     | Mathews   | Tester      |
| James      | Smith     | Tester      |
| Robert     | Black     | Tester      |
| Linda      | Green     | Tester      |
| James      | Cat       | Tester      |
+------------+-----------+-------------+
6 rows in set (0.00 sec)
```

Table Join

Dalam basis data relasional, kita mengenal relasi antar tabel. Untuk melakukan query terhadap dua atau lebih tabel yang memiliki relasi, kita bisa menggunakan fitur table join di MySQL. Sebelum memulai table join, kita persiapkan dulu tabel yang akan kita gunakan.

Pertama, kita membuat dulu tabel Jobs yang berisi daftar pekerjaan.

```
mysql> CREATE TABLE jobs (
    -> job_id INT PRIMARY KEY AUTO_INCREMENT,
    -> title VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.19 sec)
```

Kemudian kita isi tabel jobs tersebut denggan data sebagai berikut:

```
mysql> INSERT INTO jobs VALUES
    -> (1, 'Programmer'),
    -> (2, 'Tester'),
    -> (3, 'Manager'),
    -> (4, 'Spy'),
    -> (5, 'Detective'),
    -> (6, 'Forensics');
Query OK, 6 rows affected (0.36 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

Selanjutnya, kita membuat duplikat dari tabel employee, mari kita namakan employee_join sebagai berikut:

```
mysql> CREATE TABLE employee_join AS (SELECT * FROM employee);
Query OK, 12 rows affected (0.50 sec)
Records: 12  Duplicates: 0  Warnings: 0
```

Kemudian kita tambahkan kolom job_id untuk referensi ke tabel jobs sebagai berikut:

```
mysql> ALTER TABLE employee_join ADD job_id INT;
Query OK, 12 rows affected (0.62 sec)
Records: 12  Duplicates: 0  Warnings: 0

mysql> DESC employee_join;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | int(11)     | NO   |     | 0       |       |
| first_name  | varchar(15) | YES  |     | NULL    |       |
| last_name   | varchar(15) | YES  |     | NULL    |       |
| start_date  | date        | YES  |     | NULL    |       |
| end_date    | date        | YES  |     | NULL    |       |
| salary      | float(8,2)  | YES  |     | NULL    |       |
| city        | varchar(10) | YES  |     | NULL    |       |
```

```
| description | varchar(15) | YES  |     | NULL    |       |
| job_id         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
9 rows in set (0.01 sec)
```

Kemudian kita update job di tabel employee_join sesuai dengan referensi id pada tabel jobs.

```
mysql> UPDATE employee_join,jobs SET employee_join.job_id = jobs.job_id  WHERE
employee_join.description = jobs.title;
Query OK, 12 rows affected (0.06 sec)
Rows matched: 12  Changed: 12  Warnings: 0


mysql> SELECT * FROM employee_join;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+------+
| id | first_name | last_name | start_date | end_date   | salary  | city      |
description | job_id  |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer  |    1 |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver | Tester
|    2 |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver | Tester
|    2 |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver | Manager
|    3 |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver | Tester
|    2 |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  | Tester
|    2 |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  | Manager
|    3 |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver | Tester
|    2 |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    | Spy
|    4 |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  | Detective
|    5 |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  | Forensics
|    6 |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    | Detective
|    5 |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+------+
12 rows in set (0.00 sec)
```

Karena tabel employee_join sudah bereferensi dengan tabel jobs, maka kita hapus kolom description pada tabel employee_join.

```
mysql> ALTER TABLE employee_join DROP description;
Query OK, 12 rows affected (0.27 sec)
Records: 12  Duplicates: 0  Warnings: 0
```

Akhirnya, kita mendapatkan dua buah tabel yang saling bereferensi yaitu tabel employee_join dan tabel jobs.

```
mysql> SELECT * FROM employee_join;
+----+------------+-----------+------------+------------+---------+-----------+--------+
| id | first_name | last_name | start_date | end_date   | salary  | city      | job_id |
+----+------------+-----------+------------+------------+---------+-----------+--------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
+----+------------+-----------+------------+------------+---------+-----------+--------+
12 rows in set (0.00 sec)
mysql> SELECT * FROM jobs;
+--------+------------+
| job_id | title      |
+--------+------------+
|      1 | Programmer |
|      2 | Tester     |
|      3 | Manager    |
|      4 | Spy        |
|      5 | Detective  |
|      6 | Forensics  |
+--------+------------+
6 rows in set (0.00 sec)
```

Cross Join

Operasi tabel yang pertama adalah cross join atau disebut juga sebagai Cartesian join. Pada cross join, semua data dalam tabel yang pertama dipasangkan dengan semua data pada tabel yang kedua. Berikut adalah contohnya

```
mysql> SELECT *
```

```
    -> FROM employee_join, jobs;
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      | job_id |
job_id | title      |
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
1 | Programmer |
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
2 | Tester     |
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
3 | Manager    |
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
4 | Spy        |
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
5 | Detective  |
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
6 | Forensics  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
1 | Programmer |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
2 | Tester     |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
3 | Manager    |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
4 | Spy        |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
5 | Detective  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
6 | Forensics  |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
1 | Programmer |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
2 | Tester     |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
3 | Manager    |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
4 | Spy        |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
5 | Detective  |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
6 | Forensics  |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
1 | Programmer |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
2 | Tester     |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
```

```
3 | Manager    |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
4 | Spy        |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
5 | Detective  |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
6 | Forensics  |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
1 | Programmer |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
2 | Tester     |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
3 | Manager    |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
4 | Spy        |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
5 | Detective  |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
6 | Forensics  |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
1 | Programmer |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
2 | Tester     |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
3 | Manager    |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
4 | Spy        |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
5 | Detective  |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
6 | Forensics  |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
1 | Programmer |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
2 | Tester     |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
3 | Manager    |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
4 | Spy        |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
5 | Detective  |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
6 | Forensics  |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
1 | Programmer |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
2 | Tester     |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
```

```
3 | Manager    |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
4 | Spy        |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
5 | Detective  |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
6 | Forensics  |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
1 | Programmer |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
2 | Tester     |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
3 | Manager    |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
4 | Spy        |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
5 | Detective  |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
6 | Forensics  |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
1 | Programmer |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
2 | Tester     |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
3 | Manager    |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
4 | Spy        |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
5 | Detective  |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
6 | Forensics  |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
1 | Programmer |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
2 | Tester     |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
3 | Manager    |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
4 | Spy        |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
5 | Detective  |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
6 | Forensics  |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
1 | Programmer |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
2 | Tester     |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
```

```
3 | Manager    |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
4 | Spy        |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
5 | Detective  |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
6 | Forensics  |
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
72 rows in set (0.00 sec)
```

Tabel employee_join memiliki 12 data, sedangkan tabel jobs memiliki 6 data, hasil cross join memiliki 72 data.

Equi-Join atau Inner Join

Inner join menggabungkan tabel dengan membandingkan nilai yang sama antara dua buah kolom. Kolom yang dibandingkan dapat kita spesifikasikan.

```
mysql> SELECT * FROM
    -> employee_join, jobs
    -> WHERE employee_join.job_id = jobs.job_id;
+mysql> SELECT * FROM employee_join, jobs WHERE employee_join.job_id = jobs.job_id;
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      | job_id |
job_id | title      |
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
1 | Programmer |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
2 | Tester     |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
2 | Tester     |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
2 | Tester     |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
2 | Tester     |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
2 | Tester     |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
3 | Manager    |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
3 | Manager    |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
4 | Spy        |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
5 | Detective  |
```

```
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
5 | Detective  |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
6 | Forensics  |
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
12 rows in set (0.00 sec)
```

Selain menggunakan WHERE, kita juga dapat menggunakan INNER JOIN … ON … .

```
mysql> SELECT * FROM  employee_join t1 INNER JOIN jobs t2 ON t1.job_id = t2.job_id;
mysql> SELECT * FROM employee_join INNER JOIN jobs ON employee_join.job_id = jobs.job_id;
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
| id | first_name | last_name | start_date | end_date   | salary  | city      | job_id |
job_id | title      |
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
1 | Programmer |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
2 | Tester     |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
2 | Tester     |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
2 | Tester     |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
2 | Tester     |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
2 | Tester     |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
3 | Manager    |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
3 | Manager    |
|  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4 |
4 | Spy        |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
5 | Detective  |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
5 | Detective  |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
6 | Forensics  |
+----+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
12 rows in set (0.00 sec)
```

Selain menggunakan ON, kita juga bisa menggunakan USING dan memberikan satu nama kolom yang ingin kita bandingkan.

```
mysql> SELECT * FROM employee_join INNER JOIN jobs USING (job_id);
+--------+----+------------+-----------+------------+------------+---------+-----------
```

```
+------------+
| job_id | id | first_name | last_name | start_date | end_date   | salary  | city      |
title      |
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
|      1 |  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer |
|      2 |  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |
Tester     |
|      2 |  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |
Tester     |
|      3 |  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |
Manager    |
|      2 |  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |
Tester     |
|      2 |  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |
Tester     |
|      3 |  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |
Manager    |
|      2 |  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |
Tester     |
|      4 |  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |
Spy        |
|      5 | 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |
Detective  |
|      6 | 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |
Forensics  |
|      5 | 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |
Detective  |
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
12 rows in set (0.00 sec)
```

Natural Join

Natural join sebenarnya mirip dengan INNER JOIN. Namun kita tidak perlu menspesifikasikan kolom mana yang ingin kita bandingkan. Secara automatis, MySQL akan mencari kolom pada dua buah tabel yang memiliki nilai yang sama dan membandingkannya. Sebagai contoh, untuk tabel employee_join dan jobs, yang dibandingkan adalah kolom job_id yang ada di keduanya.

```
mysql> SELECT * FROM  employee_join NATURAL JOIN jobs;
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
| job_id | id | first_name | last_name | start_date | end_date   | salary  | city      |
title      |
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
|      1 |  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer |
```

```
|      2 |  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |
Tester     |
|      2 |  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |
Tester     |
|      2 |  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |
Tester     |
|      2 |  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |
Tester     |
|      2 |  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |
Tester     |
|      3 |  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |
Manager    |
|      3 |  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |
Manager    |
|      4 |  9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |
Spy        |
|      5 | 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |
Detective  |
|      5 | 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |
Detective  |
|      6 | 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |
Forensics  |
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
12 rows in set (0.00 sec)
```

Left Join dan Right Join

Left join dan right join digunakan untuk menghindari data yang hilang karena mungkin ada data yang belum direferensi oleh tabel yang lain. Sebagai contoh, kita tambahkan data di jobs tapi kita tidak referensikan di tabel employee_join sebagai berikut:

```
mysql> INSERT INTO jobs VALUES (7, 'Developer');
Query OK, 1 row affected (0.04 sec)
```

Kemudian jika kita lakukan LEFT JOIN dengan tabel jobs yang kita sebut terlebih dahulu (atau di sebelah kiri) maka akan muncul sebagai berikut:

```
mysql> SELECT * FROM  jobs t1 LEFT JOIN employee_join t2 ON t1.job_id = t2.job_id;
+--------+------------+------+------------+-----------+------------+------------
+---------+-----------+--------+
| job_id | title      | id   | first_name | last_name | start_date | end_date   | salary
| city      | job_id |
+--------+------------+------+------------+-----------+------------+------------
+---------+-----------+--------+
|      1 | Programmer |    1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56
| Toronto   |      1 |
|      2 | Tester     |    2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78
| Vancouver |      2 |
|      2 | Tester     |    3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78
```

```
| Vancouver |      2 |
|      2 | Tester     |    5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78
| Vancouver |      2 |
|      2 | Tester     |    6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78
| New York  |      2 |
|      2 | Tester     |    8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78
| Vancouver |      2 |
|      3 | Manager    |    4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78
| Vancouver |      3 |
|      3 | Manager    |    7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78
| New York  |      3 |
|      4 | Spy        |    9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56
| London    |      4 |
|      5 | Detective  |   10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98
| Brussels  |      5 |
|      5 | Detective  |   12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21
| London    |      5 |
|      6 | Forensics  |   11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98
| New York  |      6 |
|      7 | Developer  | NULL | NULL       | NULL      | NULL       | NULL       |    NULL
| NULL      |   NULL |
+--------+------------+------+------------+-----------+------------+------------
+---------+-----------+--------+

13 rows in set (0.00 sec)
```

Meskipun pada tabel jobs ada data yang belum direferensi, namun data tersebut tetap ditampilkan dan diberi nilai NULL untuk nilai di tabel employee_join yang belum diketahui. Hal yang sama juga berlaku untuk RIGHT JOIN, kali ini dengan tabel jobs berada di sebelah kanan.

```
mysql> SELECT * FROM  employee_join t1 RIGHT JOIN jobs t2 ON t1.job_id = t2.job_id;
+------+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
| id   | first_name | last_name | start_date | end_date   | salary  | city      | job_id
| job_id | title      |
+------+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
|    1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1
|      1 | Programmer |
|    2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2
|      2 | Tester     |
|    3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2
|      2 | Tester     |
|    5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2
|      2 | Tester     |
|    6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2
|      2 | Tester     |
|    8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2
|      2 | Tester     |
|    4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3
```

```
|      3 | Manager    |
|    7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3
|      3 | Manager    |
|    9 | James      | Bond      | 1982-04-21 | 2002-09-23 | 1235.56 | London    |      4
|      4 | Spy        |
|   10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5
|      5 | Detective  |
|   12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5
|      5 | Detective  |
|   11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6
|      6 | Forensics  |
| NULL | NULL       | NULL      | NULL       | NULL       |    NULL | NULL      |   NULL
|      7 | Developer  |
+------+------------+-----------+------------+------------+---------+-----------+--------
+--------+------------+
13 rows in set (0.00 sec)
```

Update menggunakan Join Table

Kita juga dapat melakukan update menggunakan Join table. Bentuk sintaksnya adalah

```
UPDATE <table1>, <table2>
SET <column_name> = 'new_value'
WHERE <conditions>
```

Misal, kita ingin merubah nama belakang dari employee yang pekerjaannya Spy menjadi 'Bono'. Kita dapat melakukannya sebagai berikut:

```
mysql> SELECT first_name, last_name, title FROM employee_join INNER JOIN jobs USING
(job_id);
+------------+-----------+------------+
| first_name | last_name | title      |
+------------+-----------+------------+
| Jason      | Martin    | Programmer |
| Alison     | Mathews   | Tester     |
| James      | Smith     | Tester     |
| Celia      | Rice      | Manager    |
| Robert     | Black     | Tester     |
| Linda      | Green     | Tester     |
| David      | Larry     | Manager    |
| James      | Cat       | Tester     |
| James      | Bond      | Spy        |
| Hercule    | Poirot    | Detective  |
| Lincoln    | Rhyme     | Forensics  |
| Sherlock   | Holmes    | Detective  |
+------------+-----------+------------+
12 rows in set (0.00 sec)
```

```
mysql> UPDATE employee_join, jobs
    -> SET last_name = "Bono"
    -> WHERE employee_join.job_id = jobs.job_id AND jobs.title = 'Spy';
Query OK, 1 row affected (0.37 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT first_name, last_name, title FROM employee_join INNER JOIN jobs USING
(job_id);
+------------+-----------+------------+
| first_name | last_name | title      |
+------------+-----------+------------+
| Jason      | Martin    | Programmer |
| Alison     | Mathews   | Tester     |
| James      | Smith     | Tester     |
| Celia      | Rice      | Manager    |
| Robert     | Black     | Tester     |
| Linda      | Green     | Tester     |
| David      | Larry     | Manager    |
| James      | Cat       | Tester     |
| James      | Bono      | Spy        |
| Hercule    | Poirot    | Detective  |
| Lincoln    | Rhyme     | Forensics  |
| Sherlock   | Holmes    | Detective  |
+------------+-----------+------------+
12 rows in set (0.00 sec)
```

Delete menggunakan join table

Kita juga dapat menghapus data menggunakan join table sebagai berikut:

```
mysql> DELETE employee_join FROM employee_join, jobs WHERE employee_join.job_id =
jobs.job_id AND jobs.title = 'Spy';
Query OK, 1 row affected (0.37 sec)

mysql> SELECT * FROM employee_join INNER JOIN jobs USING (job_id);
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
| job_id | id | first_name | last_name | start_date | end_date   | salary  | city      |
title      |
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
|      1 |  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |
Programmer |
|      2 |  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |
```

```
Tester     |
|      2 |  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |
Tester     |
|      3 |  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |
Manager    |
|      2 |  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |
Tester     |
|      2 |  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |
Tester     |
|      3 |  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |
Manager    |
|      2 |  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |
Tester     |
|      5 | 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |
Detective  |
|      6 | 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |
Forensics  |
|      5 | 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |
Detective  |
+--------+----+------------+-----------+------------+------------+---------+-----------
+------------+
11 rows in set (0.00 sec)
```

MODUL EMPAT

Pertemuan 4 View dan Trigger

View

View adalah query tersimpan yang menghasilkan result set ketika dipanggil. View bertindak sebagai tabel virtual. Beberapa hal yang tidak boleh digunakan pada query yang mendefinisikan view adalah sebagai berikut:

- definisi view tidak boleh memiliki sub query di klausa FROM dari statement SQL
- variabel user, sistem, atau lokal tidak boleh digunakan dalam sebuah SQL SELECT
- view tidak dapat merujuk ke tabel temporer
- trigger tidak dapat diasosiasikan terhadap sebuah view
- view yang dibuat di dalam stored procedure tidak dapat merujuk kepada parameter dalam stored procedure

Pembuatan view memerlukan pendefinisian nama view dan sebuah statement SQL. Setelah view dibuat, view dapat diquery seperti tabel biasa. Bentuk dasar pembuatan view adalah sebagai berikut:

```
CREATE [OR REPLACE] [<algorithm attributes>] VIEW [database.]< name> [(<columns>)]
AS <SELECT statement> [WITH <LOCAL | CASCADE> <check options>]
```

Pembuatan view dapat menggunakan OR REPLACE untuk mengganti view yang telah ada sebelumnya. Berikut adalah contoh pembuatan view sederhana.

```
mysql> CREATE VIEW view1 AS
    -> SELECT CONCAT(first_name, " ", last_name), city FROM employee;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM view1;
+-----------------------------------+-----------+
| CONCAT(first_name, " ", last_name) | city      |
+-----------------------------------+-----------+
| Jason Martin                      | Toronto   |
| Alison Mathews                    | Vancouver |
| James Smith                       | Vancouver |
| Celia Rice                        | Vancouver |
| Robert Black                      | Vancouver |
| Linda Green                       | New York  |
| David Larry                       | New York  |
| James Cat                         | Vancouver |
| James Bond                        | London    |
| Hercule Poirot                    | Brussels  |
| Lincoln Rhyme                     | New York  |
| Sherlock Holmes                   | London    |
+-----------------------------------+-----------+
```

```
12 rows in set (0.00 sec)
```

Kita juga bisa mendefinisikan nama kolom sendiri seperti contoh sebagai berikut:

```
mysql> CREATE VIEW view2 (name, place) AS
    -> SELECT CONCAT(first_name, " ", last_name), city FROM employee;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM view2;
+-----------------+-----------+
| name            | place     |
+-----------------+-----------+
| Jason Martin    | Toronto   |
| Alison Mathews  | Vancouver |
| James Smith     | Vancouver |
| Celia Rice      | Vancouver |
| Robert Black    | Vancouver |
| Linda Green     | New York  |
| David Larry     | New York  |
| James Cat       | Vancouver |
| James Bond      | London    |
| Hercule Poirot  | Brussels  |
| Lincoln Rhyme   | New York  |
| Sherlock Holmes | London    |
+-----------------+-----------+
12 rows in set (0.00 sec)
```

Algorithm Attributes

Algorithm attributes memberikan kendali bagaimana MySQL memproses view. Atribut ini bersifat opsional. Atribut algoritma menerima tiga buah nilai, yaitu MERGE, TEMPTABLE, dan UNDEFINED. Default algoritma yang dipilih adalah UNDEFINED.

Untuk algoritma MERGE, teks dari statement query yang merujuk ke sebuah view dan definisi view digabung sedemikian sehingga bagian dari definisi view menggantikan bagian yang bersesuaian dengan statement.
Untuk algoritma TEMPTABLE, hasil dari view diambil ke dalam temporary table, yang kemudian digunakan untuk menjalankan statement. TEMPTABLE dipilih karena lock pada table yang digunakan dapat langsung dilepas setelah temporary table telah dibuat. Akibatnya, penggunaan TEMPTABLE dapat mempercepat pelepasan lock pada table utama, sehingga klien lain yang akan menggunakan view tidak menunggu terlalu lama.

Untuk algoritma UNDEFINED, MySQL memilih sendiri algoritma mana yang akan digunakan. MySQL lebih memilih MERGE karena biasanya lebih efisien. Selain itu, view yang menggunakan TEMPTABLE tidak dapat diupdate karena menggunakan temporary table.

Berikut adalah contoh pembuatan view menggunakan atribut algoritma MERGE.

```
mysql> CREATE ALGORITHM = MERGE VIEW v_merge (name, salary)
    -> AS SELECT CONCAT(first_name," ",last_name), salary
    -> FROM employee WHERE salary > 2000;
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> SELECT * FROM v_merge;
+-----------------+---------+
| name            | salary  |
+-----------------+---------+
| Alison Mathews  | 6662.78 |
| James Smith     | 6545.78 |
| Celia Rice      | 2345.78 |
| Robert Black    | 2335.78 |
| Linda Green     | 4323.78 |
| David Larry     | 7898.78 |
| Hercule Poirot  | 4313.98 |
| Lincoln Rhyme   | 3213.98 |
| Sherlock Holmes | 4124.21 |
+-----------------+---------+
9 rows in set (0.00 sec)

mysql> SELECT * FROM v_merge WHERE salary < 5000;
+-----------------+---------+
| name            | salary  |
+-----------------+---------+
| Celia Rice      | 2345.78 |
| Robert Black    | 2335.78 |
| Linda Green     | 4323.78 |
| Hercule Poirot  | 4313.98 |
| Lincoln Rhyme   | 3213.98 |
| Sherlock Holmes | 4124.21 |
+-----------------+---------+
6 rows in set (0.00 sec)
```

Pada query di atas, kita membuat view dengan algoritma MERGE. Ketika query SELECT pada view dilakukan, MySQL akan menangani statement tersebut sebagai berikut:

```
SELECT * FROM v_merge;
```

- v_merge menjadi employee
- * menjadi name dan salary yang berkaitan dengan CONCAT(first_name, " ", last_name)
- Klausa WHERE kemudian ditambahkan

Statement query tersebut kemudian akan dieksekusi menjadi:

```
SELECT CONCAT(first_name," ",last_name), salary FROM employee WHERE salary > 2000;
```

Untuk statement query kedua:

```
SELECT * FROM v_merge WHERE salary < 5000;
```

Statement ini dijalankan seperti statement sebelumnya, namun klausa WHERE pada query ditambahkan pada view menggunakan konektor AND dan parentesis (kurung buka dan tutup) untuk memastikan urutan pengerjaan klausa). Statement query di atas akan dieksekusi sebagai:

```
SELECT CONCAT(first_name," ", last_name), salary FROM employee WHERE (salary > 2000) AND
(salary < 5000);
```

Algoritma MERGE tidak dapat digunakan ketika view memerlukan penggunaan konstruksi sebagai berikut:

- Fungsi aggregate, seperti SUM(), MIN(), MAX(), COUNT() dan lain-lain
- DISTINCT
- GROUP BY
- HAVING
- LIMIT
- UNION atau UNION ALL
- Subquery pada SELECT
- Referensi ke nilai literal (tidak ada tabel yang direferensi)

Ketika algoritma MERGE tidak dapat digunakan, kita bisa menggunakan algoritma TEMPTABLE.

```
mysql> CREATE ALGORITHM = MERGE VIEW myView (city, number_of_employee) AS
    -> SELECT city, count(id) FROM employee GROUP BY city;
Query OK, 0 rows affected, 1 warning (0.06 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1354
Message: View merge algorithm can't be used here for now (assumed undefined algorithm)
1 row in set (0.00 sec)
```

Seperti contoh di atas, ketika kita menggunakan MERGE untuk statement SELECT yang mengandung GROUP BY dan fungsi agregat COUNT(), MySQL akan memberikan pesan warning dan merubah algoritma secara automatis menjadi UNDEFINED.

```
mysql> CREATE ALGORITHM = TEMPTABLE VIEW tempView (city, number_of_employee) AS SELECT
city, count(id) FROM employee GROUP BY city;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM tempView;
+-----------+--------------------+
| city      | number_of_employee |
+-----------+--------------------+
| Brussels  |                  1 |
| London    |                  2 |
| New York  |                  3 |
| Toronto   |                  1 |
| Vancouver |                  5 |
+-----------+--------------------+
5 rows in set (0.00 sec)
```

Ketika kita mendefinisikan VIEW menggunakan algoritma TEMPTABLE tidak muncul pesan warning.

Updatable View

Ada beberapa view yang dapat diupdate menggunakan statement UPDATE, DELETE, atau INSERT. Update juga dilakukan pada tabel yang dirujuk view tersebut. Supaya sebuah view dapat diupdate, harus ada hubungan satu-satu antara baris dalam view dengan baris pada tabelnya. Selain view yang dapat diupdate, ada juga view yang tidak dapat diupdate, yaitu view yang memiliki:

- fungsi aggregat, seperti SUM(), MIN(), MAX(), COUNT().
- DISTINCT
- GROUP BY
- HAVING
- UNION atau UNION ALL
- sub query di SELECT
- Join-join tertentu
- Nonupdatable view di klausa FROM
- Sebuah subquery di klausa WHERE yang merujuk ke sebuah tabel dalam klausa FROM.
- Hanya merujuk ke nilai literal (tidak merujuk ke sebuah tabel)
- Menggunakan ALGORITHM = TEMPTABLE (penggunaan temporary table membuat view tidak dapat diupdate)
- Referensi berulangkali ke kolom manapun dari sebuah tabel.

```
mysql> select * from view2;
+-----------------+-----------+
| name            | place     |
+-----------------+-----------+
| Jason Martin    | Toronto   |
| Alison Mathews  | Vancouver |
| James Smith     | Vancouver |
| Celia Rice      | Vancouver |
| Robert Black    | Vancouver |
| Linda Green     | New York  |
| David Larry     | New York  |
| James Cat       | Vancouver |
| James Bond      | London    |
| Hercule Poirot  | Brussels  |
| Lincoln Rhyme   | New York  |
| Sherlock Holmes | London    |
+-----------------+-----------+
12 rows in set (0.00 sec)

mysql> UPDATE view2 SET place = "Canberra" WHERE place = "Toronto";
```

```
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from view2;
+-----------------+-----------+
| name            | place     |
+-----------------+-----------+
| Jason Martin    | Canberra  |
| Alison Mathews  | Vancouver |
| James Smith     | Vancouver |
| Celia Rice      | Vancouver |
| Robert Black    | Vancouver |
| Linda Green     | New York  |
| David Larry     | New York  |
| James Cat       | Vancouver |
| James Bond      | London    |
| Hercule Poirot  | Brussels  |
| Lincoln Rhyme   | New York  |
| Sherlock Holmes | London    |
+-----------------+-----------+
12 rows in set (0.00 sec)

mysql> select * from employee where city = 'Canberra';
+----+------------+-----------+------------+------------+---------+----------
+-------------+
|  id  | first_name  | last_name  | start_date  | end_date    | salary   | city       |
description |
+----+------------+-----------+------------+------------+---------+----------
+-------------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Canberra | Programmer
|
+----+------------+-----------+------------+------------+---------+----------
+-------------+
1 row in set (0.00 sec)
```

Contoh di atas menunjukkan update terhadap kota Canberra juga terjadi pada tabel yang direferensi oleh view.

View dapat diupdate dengan INSERT jika memenuhi syarat sebagai berikut:

- Tidak boleh ada duplikasi nama kolom
- View harus terdiri atas semua kolom pada tabel dasar yang tidak memiliki nilai default
- Kolom view harus merupakan referensi kolom dan bukan kolom turunan

Berikut adalah contoh kolom turunan:

```
3.1415
CONCAT(first_name, " ", last_name)
```

```
salary + 1
(subquery)
```

Berikut adalah contoh penggunaan INSERT untuk mengupdate view.

```
mysql> INSERT INTO view2 VALUES ('Miss Marple', 'London');
ERROR 1471 (HY000): The target table view2 of the INSERT is not insertable-into
```

Insert untuk view2 tidak dapat dilakukan karena memiliki kolom turunan yaitu name turunan dari CONCAT(first_name, " ", last_name). Meski demikian, view2 ini tetap dapat diupdate, namun hanya pada kolom yang bukan kolom turunan.

```
mysql> UPDATE view2 SET name = "John Doe" WHERE name = "James Bond";
ERROR 1348 (HY000): Column 'name' is not updatable

mysql> UPDATE view2 SET place = "Toronto" WHERE place = "Canberra";
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM view2;
+-----------------+-----------+
| name            | place     |
+-----------------+-----------+
| Jason Martin    | Toronto   |
| Alison Mathews  | Vancouver |
| James Smith     | Vancouver |
| Celia Rice      | Vancouver |
| Robert Black    | Vancouver |
| Linda Green     | New York  |
| David Larry     | New York  |
| James Cat       | Vancouver |
| James Bond      | London    |
| Hercule Poirot  | Brussels  |
| Lincoln Rhyme   | New York  |
| Sherlock Holmes | London    |
+-----------------+-----------+
12 rows in set (0.00 sec)
```

Berikut adalah contoh memasukkan data menggunakan INSERT ke dalam view. INSERT dapat dilakukan karena kolom-kolom didefinisikan memiliki nilai default NULL.

```
mysql> SELECT * FROM emp_high;
+------------+---------+-----------+
| first_name | salary  | city      |
+------------+---------+-----------+
| Jason      | 1235.56 | Canberra  |
| Celia      | 2345.78 | Vancouver |
| Robert     | 2335.78 | Vancouver |
| James      | 1233.78 | Vancouver |
```

```
| James      | 1235.56 | London    |
| Lincoln    | 3213.98 | New York  |
+------------+---------+-----------+
6 rows in set (0.00 sec)

mysql> INSERT INTO emp_high VALUE ('Marple', 3000, 'London');
Query OK, 1 row affected (0.41 sec)

mysql> SELECT * FROM emp_high;
+------------+---------+-----------+
| first_name | salary  | city      |
+------------+---------+-----------+
| Jason      | 1235.56 | Canberra  |
| Celia      | 2345.78 | Vancouver |
| Robert     | 2335.78 | Vancouver |
| James      | 1233.78 | Vancouver |
| James      | 1235.56 | London    |
| Lincoln    | 3213.98 | New York  |
| Marple     | 3000.00 | London    |
+------------+---------+-----------+
7 rows in set (0.00 sec)

mysql> INSERT INTO emp_high VALUE ('John', 4500, 'London');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM emp_high;
+------------+---------+-----------+
| first_name | salary  | city      |
+------------+---------+-----------+
| Jason      | 1235.56 | Canberra  |
| Celia      | 2345.78 | Vancouver |
| Robert     | 2335.78 | Vancouver |
| James      | 1233.78 | Vancouver |
| James      | 1235.56 | London    |
| Lincoln    | 3213.98 | New York  |
| Marple     | 3000.00 | London    |
+------------+---------+-----------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM employee;
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city        |
```

```
description |
+----+-----------+-----------+-----------+-----------+---------+-----------
+-------------+
|   1  | Jason        | Martin      | 1996-07-25  | 2006-07-25  | 1235.56  | Canberra   |
Programmer  |
|   2 | Alison      | Mathews    | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver | Tester
|
|   3 | James       | Smith      | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver | Tester
|
|   4 | Celia       | Rice       | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver | Manager
|
|   5 | Robert      | Black      | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver | Tester
|
|   6 | Linda       | Green      | 1987-07-30 | 1996-01-04 | 4323.78 | New York   | Tester
|
|   7 | David       | Larry      | 1990-12-31 | 1998-02-12 | 7898.78 | New York   | Manager
|
|   8 | James       | Cat        | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver | Tester
|
|   9 | James        | Bond        | 1982-04-21 | 2002-09-23 | 1235.56 | London     | Spy
|
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels   | Detective
|
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York   | Forensics
|
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    | Detective
|
| 13 | Marple      | NULL       | NULL        | NULL        | 3000.00 | London     | NULL
|
| 14 | John        | NULL       | NULL        | NULL        | 4500.00 | London     | NULL
|
+----+-----------+-----------+-----------+-----------+---------+-----------
+-------------+
14 rows in set (0.00 sec)
```

Klausa WITH CHECK OPTION

Klausa WITH CHECK OPTION dalam pembuatan view diberikan kepada updatable view untuk mencegah INSERT atau UPDATE pada baris data kecuali klausa WHERE pada statement SELECT bernilai benar (true). Parameter yang dapat digunakan untuk CHECK OPTION adalah LOCAL dan CASCADED. Parameter LOCAL berarti pengecekan hanya dilakukan pada view yang didefinisikan. Parameter CASCADED berarti pengecekan dilakukan pada view beserta tabel yang direferensi oleh view tersebut. Secara default, pendefinisian CHECK OPTION bernilai CASCADED.

Berikut adalah contoh definisi view menggunakan CHECK OPTION:

```
mysql> CREATE VIEW v1 AS
    -> SELECT first_name, salary, city FROM employee WHERE salary < 3000
    -> WITH CHECK OPTION;
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE salary > 0 WITH LOCAL CHECK OPTION;
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE salary > 0 WITH CASCADED CHECK OPTION;
Query OK, 0 rows affected (0.43 sec)

mysql> INSERT INTO v2 VALUES ("Doe", 3500, "Indonesia");
Query OK, 1 row affected (0.36 sec)

mysql> INSERT INTO v3 VALUES ("Hancock", 3500, "Jakarta");
ERROR 1369 (HY000): CHECK OPTION failed 'PRAK2.v3'
```

Pada contoh di atas, kita membuat view v1 yang berisi data dari employee yang memiliki salary < 3000. Kemudian kita mendefinisikan view v2 dengan tambahan batasan salary > 0 dan LOCAL CHECK OPTION. Selain itu, kita juga mendefinisikan view v3 dengan tambahan batasan salary > 0 namun dengan CASCADED CHECK OPTION.

Selanjutnya, kita mencoba memasukkan data ke dalam v2 dan v3. Pada v2, data bernilai salary 3500 dapat dimasukkan, karena pengecekan dilakukan secara LOCAL, yaitu hanya sebatas pada view v2 yang didefinisikan > 0. Sebaliknya, pada v3, data bernilai salary 3500 tidak dapat dimasukkan karena pengecekan dilakukan secara CASCADED, yaitu melibatkan pengecekan pada v1 yang direferensi oleh v3. Karena v1 memiliki batasan salary < 3000, maka masukan data salary 3500 akan bernilai salah.

Merubah View

View yang telah dibuat dapat diubah menggunakan ALTER VIEW. Berikut adalah contohnya.

```
mysql> SELECT * FROM view1;
+-----------------------------------+-----------+
| CONCAT(first_name, " ", last_name) | city      |
+-----------------------------------+-----------+
| Alison Mathews                    | Vancouver |
| James Smith                       | Vancouver |
| Celia Rice                        | Vancouver |
| Robert Black                      | Vancouver |
| Linda Green                       | New York  |
| David Larry                       | New York  |
| Hercule Poirot                    | Brussels  |
| Lincoln Rhyme                     | New York  |
| Sherlock Holmes                   | London    |
| NULL                              | London    |
| NULL                              | London    |
| NULL                              | London    |
| NULL                              | Indonesia |
| NULL                              | Jakarta   |
+-----------------------------------+-----------+
14 rows in set (0.00 sec)
```

```
mysql> ALTER VIEW view1 (name, city) AS
    -> SELECT CONCAT(first_name, " ", last_name), city FROM employee;
Query OK, 0 rows affected (0.38 sec)

mysql> SELECT * FROM view1;
+-----------------+-----------+
| name            | city      |
+-----------------+-----------+
| Alison Mathews  | Vancouver |
| James Smith     | Vancouver |
| Celia Rice      | Vancouver |
| Robert Black    | Vancouver |
| Linda Green     | New York  |
| David Larry     | New York  |
| Hercule Poirot  | Brussels  |
| Lincoln Rhyme   | New York  |
| Sherlock Holmes | London    |
| NULL            | London    |
| NULL            | London    |
| NULL            | London    |
| NULL            | Indonesia |
| NULL            | Jakarta   |
+-----------------+-----------+
14 rows in set (0.00 sec)
```

Melihat definisi pembuatan view

Kita dapat melihat kembali sintaks yang kita gunakan untuk membuat view menggunakan SHOW CREATE. Berikut adalah contohnya:

```
mysql> SHOW CREATE VIEW view1\G
*************************** 1. row ***************************
                View: view1
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
DEFINER VIEW `view1` AS select concat(`employee`.`first_name`,' ',`employee`.`last_name`)
AS `name`,`employee`.`city` AS `city` from `employee`
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (0.00 sec)
```

Selain itu, command DESC juga dapat digunakan untuk melihat deskripsi dari view yang telah kita buat.

```
mysql> DESC view1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(31) | YES  |     | NULL    |       |
```

```
| city  | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> DESC view2;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(31) | YES  |     | NULL    |       |
| place | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> DESC v_merge;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| name   | varchar(31) | YES  |     | NULL    |       |
| salary | float(8,2)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Menghapus View

View yang telah kita buat dapat kita hapus menggunakan statement DROP VIEW.

```
mysql> DROP VIEW view1;
Query OK, 0 rows affected (0.00 sec)
```

Trigger

Trigger adalah sebuah objek database yang diasosiasikan dengan sebuah tabel. Trigger diaktifkan ketika sebuah event terjadi pada tabel yang diasosiasikan. Tabel yang diasosiasikan dengan trigger haruslah sebuah tabel yang permanen dan bukan temporary tabel.

Membuat trigger

Trigger dibuat menggunakan sintaks sebagai berikut:

```
CREATE TRIGGER <trigger_name> <trigger_time> <trigger_event>
ON <table>
FOR EACH ROW <trigger_body statements>
```

Pada sintaks di atas, trigger_name adalah nama dari trigger yang akan kita buat.

Trigger_time adalah saat aktivasi trigger. Parameter trigger_time dapat berisi BEFORE atau AFTER, yang menandakan apakah aktivasi trigger dilakukan sebelum atau sesudah terjadi perubahan pada tabel.

Trigger_event menunjukkan jenis statement yang mengaktifkan trigger. Trigger_event dapat didefinisikan sebagai salah satu dari:

- INSERT: trigger diaktifkan ketika sebuah baris baru ditambahkan ke tabel, sebagai contoh melalui statement INSERT, LOAD DATA, atau REPLACE.
- UPDATE: trigger diaktifkan ketika sebuah baris dimodifikasi, sebagai contoh melalui statement UPDATE.
- DELETE: trigger diaktifkan ketika sebuah baris dihapus, melalui statement DELETE dan REPLACE. Namun demikian, DROP TABLE dan TRUNCATE TABLE tidak mengaktifkan trigger DELETE

Tidak boleh ada dua buah trigger yang sama pada sebuah tabel yang memiliki trigger_time dan trigger_event yang sama. Sebagai contoh, kita tidak dapat membuat dua buah BEFORE UPDATE trigger pada satu buah tabel yang sama, namun kita dapat membuat trigger BEFORE UPDATE dan AFTER UPDATE untuk satu tabel yang sama.

trigger_body merupakan definisi statement yang dijalankan ketika sebuah trigger diaktifkan. Jika ada beberapa statement yang ingin dijalankan, statement-statement tersebut dapat didefinisikan di antara BEGIN … AND.

Keyword OLD dan NEW dapat digunakan untuk mereferensi nilai sebelum dan sesudah trigger dilakukan. Sebagai contoh OLD.nama_kolom menunjukkan nilai kolom sebelum data tersebut dihapus atau diupdate, sedangkan NEW.nama_kolom menunjukkan nilai kolom sebuah data yang akan dimasukkan atau nilai kolom data setelah diupdate.

BEFORE TRIGGER

Berikut adalah contoh trigger yang dijalankan sebelum data dimasukkan ke dalam sebuah tabel.

```
mysql> DELIMITER //
mysql> CREATE TRIGGER before_insert BEFORE INSERT ON employee
    -> FOR EACH ROW
    -> BEGIN
    -> IF NEW.salary IS NULL OR NEW.salary = 0 THEN
    ->    SET NEW.salary = 1000;
```

```
    -> ELSE
    ->    SET NEW.salary  = NEW.salary + 100;
    -> END IF;
    -> END //
Query OK, 0 rows affected (0.13 sec)

mysql> DELIMITER ;
```

Pada contoh di atas, kita membuat trigger before insert yang akan dijalankan sebelum INSERT dilakukan ke tabel employee. Trigger ini akan mengganti nilai salary jika salary yang dimasukkan dalam INSERT bernilai NULL atau 0 dan menambahkan 100 jika selainnya. Berikut adalah contoh memasukkan data setelah didefinisikan trigger.

```
mysql> insert into employee(id,first_name, last_name, start_date, end_Date, salary,
City, Description)
    -> values (1,'John', 'Doe', '19960725',  '20060725', 0, 'Canberra',  'Programmer');
Query OK, 1 row affected (0.40 sec)

mysql> SELECT * FROM employee WHERE first_name = 'John';
+----+------------+-----------+------------+------------+---------+----------
+-------------+
| id | first_name | last_name | start_date | end_date    | salary   | city       |
description |
+----+------------+-----------+------------+------------+---------+----------
+-------------+
|  1 | John       | Doe       | 1996-07-25 | 2006-07-25 | 1000.00 | Canberra | Programmer
|
+----+------------+-----------+------------+------------+---------+----------
+-------------+
1 rows in set (0.00 sec)
```

Namun demikian, kita tidak dapat mengupdate tabel yang sama dengan tabel yang diasosiasikan dengan trigger menggunakan trigger.

```
mysql> DELIMITER &&
mysql> CREATE TRIGGER before_update BEFORE UPDATE ON employee
    -> FOR EACH ROW
    -> BEGIN
    ->    UPDATE employee SET salary = salary+(NEW.salary - OLD.salary)
    ->    WHERE id = NEW.id;
    -> END &&
Query OK, 0 rows affected (0.13 sec)

mysql> DELIMITER ;
mysql> UPDATE employee SET salary = 1000 WHERE id = 1;
ERROR 1442 (HY000): Can't update table 'employee' in stored function/trigger because it
is already used by statement which invoked this stored function/trigger.
```

AFTER TRIGGER

Berikut adalah contoh trigger yang dijalankan setelah update dilakukan terhadap tabel.

```
mysql> CREATE TABLE trans_log(
    ->   user_id VARCHAR(15),
    ->   description VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.16 sec)
```

Query di atas digunakan untuk membuat tabel trans_log. Tabel ini akan digunakan untuk mencatat perubahan yang terjadi pada kolom salary pada tabel employee.

```
mysql> DELIMITER $$
mysql> CREATE TRIGGER log_salary AFTER UPDATE
    -> ON employee
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO trans_log
      ->       VALUES (user(), CONCAT('merubah akun ',NEW.id,' dari ',OLD.salary, ' to
',NEW.salary));
    -> END $$
Query OK, 0 rows affected (0.12 sec)

mysql> DELIMITER ;
```

Pada query di atas, kita membuat trigger setelah update. Ketika update pada tabel employee dijalankan, perubahan nilai salary akan dicatat dalam tabel trans_log.

```
mysql> UPDATE employee SET salary = salary + 1000;
Query OK, 15 rows affected (0.13 sec)
Rows matched: 15  Changed: 15  Warnings: 0

mysql> SELECT * FROM trans_log;
+----------------+-----------------------------------------+
| user_id        | description                             |
+----------------+-----------------------------------------+
| root@localhost | merubah akun 1 dari 1000.00 to 2000.00  |
| root@localhost | merubah akun 2 dari 6662.78 to 7662.78  |
| root@localhost | merubah akun 3 dari 6545.78 to 7545.78  |
| root@localhost | merubah akun 4 dari 2345.78 to 3345.78  |
| root@localhost | merubah akun 5 dari 2335.78 to 3335.78  |
| root@localhost | merubah akun 6 dari 4323.78 to 5323.78  |
| root@localhost | merubah akun 7 dari 7898.78 to 8898.78  |
| root@localhost | merubah akun 10 dari 4313.98 to 5313.98 |
| root@localhost | merubah akun 11 dari 3213.98 to 4213.98 |
| root@localhost | merubah akun 12 dari 4124.21 to 5124.21 |
| root@localhost | merubah akun 13 dari 3000.00 to 4000.00 |
```

```
| root@localhost | merubah akun 14 dari 4500.00 to 5500.00 |
| root@localhost | merubah akun 15 dari 4500.00 to 5500.00 |
| root@localhost | merubah akun 16 dari 3500.00 to 4500.00 |
| root@localhost | merubah akun 17 dari 3500.00 to 4500.00 |
+----------------+-----------------------------------------+
15 rows in set (0.00 sec)
```

Contoh di atas mencoba melakukan update terhadap salary tabel employee. Ketika update terjadi, perubahan akan dicatat dalam tabel trans_log.

Melihat trigger yang sudah dibuat

Trigger yang telah dibuat dapat dilihat menggunakan command SHOW TRIGGERS. Contoh berikut menunjukkan bagaimana melihat trigger pada database PRAK2.

```
mysql> SHOW TRIGGERS IN PRAK2\G
*************************** 1. row ***************************
             Trigger: before_insert
               Event: INSERT
               Table: employee
               Statement: BEGIN IF NEW.salary IS NULL OR NEW.salary = 0 THEN       SET
NEW.salary = 1000; ELSE    SET NEW.salary = NEW.salary + 100; END IF; END
              Timing: BEFORE
             Created: NULL
            sql_mode:
             Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
*************************** 2. row ***************************
             Trigger: log_salary
               Event: UPDATE
               Table: employee
                 Statement: BEGIN        INSERT INTO trans_log          VALUES (user(),
CONCAT('merubah akun ',NEW.id,' dari ',OLD.salary, ' to ',NEW.salary)); END
              Timing: AFTER
             Created: NULL
            sql_mode:
             Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
2 rows in set (0.00 sec)
```

Menghapus Trigger

Trigger yang telah dibuat dapat dihapus dengan menggunakan command DROP TRIGGER, seperti yang

ditunjukkan oleh contoh berikut:

```
mysql> DROP TRIGGER before_insert;
Query OK, 0 rows affected (0.08 sec)

mysql> SHOW TRIGGERS IN PRAK2\G
*************************** 1. row ***************************
             Trigger: log_salary
               Event: UPDATE
               Table: employee
                 Statement: BEGIN       INSERT INTO trans_log         VALUES (user(),
CONCAT('merubah akun ',NEW.id,' dari ',OLD.salary, ' to ',NEW.salary)); END
              Timing: AFTER
             Created: NULL
            sql_mode:
             Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Tugas

Tabel employee_join

```
+----+------------+-----------+------------+------------+---------+-----------+--------+
| id | first_name | last_name | start_date | end_date   | salary  | city      | job_id |
+----+------------+-----------+------------+------------+---------+-----------+--------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London    |      5 |
+----+------------+-----------+------------+------------+---------+-----------+--------+
```

Tabel job

```
+--------+------------+
| job_id | title      |
+--------+------------+
|      1 | Programmer |
|      2 | Tester     |
|      3 | Manager    |
|      4 | Spy        |
|      5 | Detective  |
|      6 | Forensics  |
|      7 | Developer  |
+--------+------------+
7 rows in set (0.00 sec)
```

1. Buatlah view yang berisi name (gabungan first_name dan last_name), salary, city, dan job_description dari kedua tabel di atas. Lihat contoh hasil di bawah ini:

   ```
   +-----------------+---------+-----------+------------+
   | name            | salary  | city      | job_desc   |
   +-----------------+---------+-----------+------------+
   | Jason Martin    | 1235.56 | Toronto   | Programmer |
   | Alison Mathews  | 6662.78 | Vancouver | Tester     |
   | James Smith     | 6545.78 | Vancouver | Tester     |
   ```

```
| Celia Rice      | 2345.78 | Vancouver | Manager    |
| Robert Black    | 2335.78 | Vancouver | Tester     |
| Linda Green     | 4323.78 | New York  | Tester     |
| David Larry     | 7898.78 | New York  | Manager    |
| James Cat       | 1233.78 | Vancouver | Tester     |
| Hercule Poirot  | 4313.98 | Brussels  | Detective  |
| Lincoln Rhyme   | 3213.98 | New York  | Forensics  |
| Sherlock Holmes | 4124.21 | London    | Detective  |
+-----------------+---------+-----------+------------+
```

2. Buatlah view untuk menampilkan job_description dan jumlah employee untuk masing-masing job. Lihat contoh hasil di bawah ini:

```
+------------+-----------+
| job_desc   | emp_count |
+------------+-----------+
| Detective  |         2 |
| Forensics  |         1 |
| Manager    |         2 |
| Programmer |         1 |
| Tester     |         5 |
+------------+-----------+
```

3. Buatlah sebuah trigger untuk menyimpan data yang dihapus dalam tabel employee_join. Data yang dihapus tersebut disimpan dalam tabel baru bernama employee_bak. Lihat contoh hasil di bawah ini:

```
mysql> desc employee_bak;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   |     | 0       |       |
| first_name | varchar(15) | YES  |     | NULL    |       |
| last_name  | varchar(15) | YES  |     | NULL    |       |
| start_date | date        | YES  |     | NULL    |       |
| end_date   | date        | YES  |     | NULL    |       |
| salary     | float(8,2)  | YES  |     | NULL    |       |
| city       | varchar(10) | YES  |     | NULL    |       |
| job_id     | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

mysql> DELETE FROM employee_join WHERE id = 12;
Query OK, 1 row affected (0.38 sec)
```

```
mysql> SELECT * FROM employee_join;
+----+------------+-----------+------------+------------+---------+-----------
+--------+
| id | first_name | last_name | start_date | end_date    | salary   | city       |
job_id |
+----+------------+-----------+------------+------------+---------+-----------
+--------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto    |
1 |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver  |
2 |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver  |
2 |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver  |
3 |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver  |
2 |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York   |
2 |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York   |
3 |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver  |
2 |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels   |
5 |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York   |
6 |
+----+------------+-----------+------------+------------+---------+-----------
+--------+
10 rows in set (0.00 sec)

mysql> SELECT * FROM employee_bak;
+----+------------+-----------+------------+------------+---------+--------+--------+
| id | first_name | last_name | start_date | end_date   | salary  | city   | job_id |
+----+------------+-----------+------------+------------+---------+--------+--------+
| 12 | Sherlock   | Holmes    | 1923-08-12 | 1945-07-21 | 4124.21 | London |      5 |
+----+------------+-----------+------------+------------+---------+--------+--------+
1 row in set (0.00 sec)
```

MODUL LIMA

Pertemuan 5 Function dan Procedure

1 Intro Stored Routine

Stored routine (function dan procedure) merupakan sekumpulan statement SQL yang dapat disimpan dalam server. Setelah routine disimpan, client tidak perlu memanggil statement individual terus menerus, namun cukup dengan memanggil stored routine.

Stored routine dapat bermanfaat pada situasi berikut:

- Ketika beberapa aplikasi client ditulis menggunakan bahasa yang berbeda-beda atau bekerja pada platform yang berbeda, namun memerlukan operasi database yang sama.
- Ketika keamanan diutamakan. Function dan stored procedure dapat digunakan untuk operasi-operasi database yang umum digunakan. Routine dapat memastikan setiap operasi dicatat lognya. Aplikasi dan user dapat diatur tidak memiliki akses langsung ke database, tapi hanya dapat mengakses routine tertentu.

Stored routine dapat meningkatkan performa, karena dapat mengurangi informasi yang perlu dikirimkan antara server dan client. Kelemahan yang ditimbulkan oleh stored routine adalah peningkatan kinerja server, karena server melakukan lebih banyak pekerjaan.

Stored routine dalam MySQL dapat didefinisikan menjadi dua jenis, yaitu function dan procedure. Sebuah function dapat digunakan dalam statement apapun layaknya function yang telah disediakan oleh MySQL. Berbeda dengan function, sebuah stored procedure dapat dipanggil menggunakan statement CALL. Stored function tidak dapat rekursif, sedangkan stored procedure memperbolehkan penggunaan rekursif namun secara default dinonaktifkan (disable).

2 Function

Sebuah function dapat digunakan secara langsung dalam statement `SELECT`, `UPDATE`, dan `DELETE`. Hasil dari function dapat dikembalikan sebagai output. Sebuah function hanya dapat mengembalikan **sebuah** nilai saja.

2.1 Membuat function

Sintaks yang digunakan untuk membuat function adalah sebagai berikut:

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

Function menerima parameter (`func_parameter`) yang didefinisikan dengan nama parameter beserta tipe datanya. Pembuatan function juga perlu pendefinisian tipe data nilai yang dikembalikan oleh fungsi tersebut menggunakan statement `RETURNS`. Definisi apa yang dilakukan oleh sebuah fungsi diletakkan pada `routine_body` diapit oleh `BEGIN` dan `END`.

Berikut adalah contoh pembuatan fungsi.

```
mysql> DELIMITER //
mysql> CREATE FUNCTION full_name( in_first_name VARCHAR(15), in_last_name VARCHAR(15))
    -> RETURNS VARCHAR(35)
```

```
    -> BEGIN
    -> RETURN CONCAT(in_first_name,' ',in_last_name);
    -> END//
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;

mysql> SELECT full_name(first_name, last_name) FROM employee;
+---------------------------------+
| full_name(first_name, last_name) |
+---------------------------------+
| John Doe                        |
| Alison Mathews                  |
| James Smith                     |
| Celia Rice                      |
| Robert Black                    |
| Linda Green                     |
| David Larry                     |
| Hercule Poirot                  |
| Lincoln Rhyme                   |
| Sherlock Holmes                 |
+---------------------------------+
10 rows in set (0.00 sec)
```

2.2 Melihat function yang telah dibuat

Function apa saja yang telah kita buat dapat dilihat menggunakan statement `SHOW FUNCTION STATUS`.

```
mysql> SHOW FUNCTION STATUS;
+-------+-----------+----------+----------------+---------------------
+---------------------+---------------+---------+----------------------
+----------------------+--------------------+
| Db    | Name      | Type     | Definer        | Modified            | Created
| Security_type | Comment | character_set_client | collation_connection | Database
Collation |
+-------+-----------+----------+----------------+---------------------
+---------------------+---------------+---------+----------------------
+----------------------+--------------------+
| PRAK2 | full_name | FUNCTION | root@localhost | 2012-06-26 14:45:08 | 2012-06-26
14:45:08 | DEFINER       |         | utf8                 | utf8_general_ci      |
latin1_swedish_ci  |
+-------+-----------+----------+----------------+---------------------
+---------------------+---------------+---------+----------------------
+----------------------+--------------------+
1 row in set (0.00 sec)
```

2.3 Menghapus function

Function yang telah dibuat dapat dihapus menggunakan `DROP FUNCTION nama_function`.

```
mysql> DROP FUNCTION full_name;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW FUNCTION STATUS;
Empty set (0.00 sec)
```

3 Procedure

Procedure dapat berisi statement SQL (INSERT, UPDATE, DELETE, SELECT) atau operasi lain yang disimpan dalam database. Sebuah procedure dapat dipanggil menggunakan statement `CALL nama_procedure` disertai parameter yang diperlukan.

3.1 Membuat procedure

Sintaks untuk membuat procedure adalah sebagai berikut:

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

Berikut adalah contoh pembuatan stored procedure:

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE show_employees()
    -> BEGIN
    -> SELECT * FROM employee;
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> CALL show_employees();
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date   | salary  | city        |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  1 | John       | Doe       | 1996-07-25 | 2006-07-25 | 2100.00 | Canberra    |
Programmer  |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 7762.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 7645.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 3445.78 | Vancouver | Manager
```

```
|
|   5 | Robert      | Black      | 1984-01-15 | 1998-08-08 | 3435.78 | Vancouver | Tester
|
|   6 | Linda       | Green      | 1987-07-30 | 1996-01-04 | 5423.78 | New York  | Tester
|
|   7 | David       | Larry      | 1990-12-31 | 1998-02-12 | 8998.78 | New York  | Manager
|
| 10 | Hercule     | Poirot     | 1973-05-23 | 2001-08-09 | 5413.98 | Brussels  | Detective
|
| 11 | Lincoln     | Rhyme      | 1999-05-25 | 2011-07-13 | 4313.98 | New York  | Forensics
|
| 12 | Sherlock    | Holmes     | 1923-08-12 | 1945-07-21 | 5224.21 | London    | Detective
|
+----+-------------+------------+------------+------------+---------+-----------
+-------------+
10 rows in set (0.00 sec)


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

3.2 Parameter dalam procedure

Procedure menerima parameter (`proc_parameter`) yang didefinisikan sebagai berikut:

```
proc_parameter:
     [ IN | OUT | INOUT ] param_name type
```

Dalam parameter procedure, didefinisikan jenis parameter, yaitu `IN`, `OUT`, atau `INOUT`. Bila tidak dispesifikasikan saat pembuatan, maka default jenis parameter yang dipilih adalah `IN`.

- Parameter `IN` mengirimkan sebuah nilai ke dalam procedure. Procedure bisa saja merubah nilai parameter ini, namun perubahan tersebut tidak visibel terhadap pemanggil procedure ketika procedure tersebut selesai.
- Parameter `OUT` mengirimkan nilai dari procedure ke pemanggil. Nilai inisial dari parameter ini adalah `NULL` dan nilainya visibel terhadap pemanggil.
- Parameter `INOUT` diinisialisasi oleh pemanggil, dapat dimodifikasi oleh procedure, dan perubahan nilai parameter visibel terhadap pemanggil ketika procedure selesai.

3.2.1 Parameter IN

Berikut adalah contoh penggunaan parameter `IN`:

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE getEmployeeByCity (IN cityName VARCHAR(255))
    -> BEGIN
    -> SELECT * FROM employee WHERE city LIKE cityName;
    -> END $$
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELIMITER ;
```

Pada contoh di atas, dibuat procedure `getEmployeeByCity` dengan satu parameter masukan berjenis `IN` bernama `cityName`. Procedure ini digunakan untuk menampilkan data pada tabel employee dengan nama kota sesuai parameter masukan.

Pemanggilan procedure ini dapat dilihat pada contoh di bawah ini. Pada contoh ini, kita memasukkan "`Vancouver`" sebagai parameter masukan procedure.

```
mysql> CALL getEmployeeByCity("Vancouver");
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
| id | first_name | last_name | start_date | end_date    | salary   | city        |
description |
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 7762.78 | Vancouver | Tester
|
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 7645.78 | Vancouver | Tester
|
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 3445.78 | Vancouver | Manager
|
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 3435.78 | Vancouver | Tester
|
+----+------------+-----------+------------+------------+---------+-----------
+-------------+
4 rows in set (0.00 sec)

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

3.2.2 Parameter OUT

Berikut adalah contoh penggunaan parameter OUT.

```
mysql> DELIMITER :)
mysql> CREATE PROCEDURE getNumEmployee (OUT numEmployee INT)
    -> BEGIN
    -> SELECT COUNT(*) INTO numEmployee FROM employee;
    -> END :)
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

Pada contoh di atas, dibuat procedure untuk menampilkan jumlah employee dalam tabel. Hasil query tersebut disimpan dalam variabel `numEmployee` dengan statement `INTO numEmployee`. Pemanggilan employee dengan parameter `OUT` dilakukan dengan menggunakan variabel session yang diawali dengan karakter `@`. Pemanggilan procedure `getNumEmployee` ditunjukkan sebagai berikut:

```
mysql> CALL getNumEmployee(@num);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @num;
+------+
| @num |
+------+
|   10 |
+------+
1 row in set (0.00 sec)
```

3.2.3 Parameter INOUT

Berikut adalah contoh penggunaan parameter INOUT.

```
mysql> DELIMITER ^^
mysql> CREATE PROCEDURE increase(INOUT number INT)
    -> BEGIN
    -> SET number = number + 15;
    -> END ^^
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

Pada contoh di atas, kita membuat procedure `increase` untuk menambahkan input dengan nilai 15. Memodifikasi nilai parameter input dilakukan dengan menggunakan `SET`. Contoh berikut memperlihatkan bagaimana memanggil procedure `increase`. Kita mendefinisikan terlebih dahulu variabel session `@num` dengan nilai 100. Kemudian setelah pemanggilan increase, nilai `@num` menjadi 115.

```
mysql> SET @num = 100;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL increase(@num);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @num;
+------+
| @num |
+------+
|  115 |
+------+
1 row in set (0.00 sec)
```

3.2.4 Melihat procedure yang telah dibuat

Procedure yang telah kita buat dapat dilihat menggunakan statement SHOW PROCEDURE STATUS sebagai berikut:

```
mysql> SHOW PROCEDURE STATUS;
+-------+------------------+-----------+----------------+---------------------
+---------------------+---------------+---------+----------------------
+----------------------+--------------------+

| Db    | Name               | Type      | Definer        | Modified            | Created
|  Security_type  |  Comment  |  character_set_client  |  collation_connection  |  Database
Collation |
+-------+------------------+-----------+----------------+---------------------
+---------------------+---------------+---------+----------------------
+----------------------+--------------------+

| PRAK2 | getEmployeeByCity | PROCEDURE | root@localhost | 2012-06-26 18:37:14 | 2012-06-
26 18:37:14 | DEFINER        |          | utf8                   | utf8_general_ci       |
latin1_swedish_ci  |

| PRAK2 | getNumEmployee    | PROCEDURE | root@localhost | 2012-06-26 18:53:41 | 2012-06-
26 18:53:41 | DEFINER        |          | utf8                   | utf8_general_ci       |
latin1_swedish_ci  |

| PRAK2 | increase          | PROCEDURE | root@localhost | 2012-06-26 19:09:53 | 2012-06-
26 19:09:53 | DEFINER        |          | utf8                   | utf8_general_ci       |
latin1_swedish_ci  |

| PRAK2 | increaseSalary    | PROCEDURE | root@localhost | 2012-06-24 04:21:51 | 2012-06-
24 04:21:51 | DEFINER        |          | utf8                   | utf8_general_ci       |
latin1_swedish_ci  |

| PRAK2 | show_employees    | PROCEDURE | root@localhost | 2012-06-26 15:19:46 | 2012-06-
26 15:19:46 | DEFINER        |          | utf8                   | utf8_general_ci       |
latin1_swedish_ci  |

| prak  | increaseSalary    | PROCEDURE | root@localhost | 2012-06-26 13:57:23 | 2012-06-
26 13:57:23 | DEFINER        |          | utf8                   | utf8_general_ci       |
latin1_swedish_ci  |

+-------+------------------+-----------+----------------+---------------------
+---------------------+---------------+---------+----------------------
+----------------------+--------------------+

6 rows in set (0.01 sec)
```

3.2.5 Menghapus procedure

Procedure yang telah kita buat dapat dihapus menggunakan DROP PROCEDURE.

```
mysql> DROP PROCEDURE increaseSalary;

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

4 Pemrograman di Function dan Procedure

Di dalam function dan procedure, kita bisa memasukkan logika pemrograman. Ada beberapa karakteristik pemrograman yang didukung oleh MySQL. Beberapa di antaranya adalah penggunaan variabel, kendali kondisional, dan perulangan.

4.1 Variabel

Seperti pada pemrograman pada umumnya, kita bisa menggunakan variabel lokal pada function dan procedure. Pendeklarasian variabel memiliki sintaks sebagai berikut:

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

Nilai inisialisasi variabel dapat dilakukan menggunakan statement `DEFAULT`. Jika statement `DEFAULT` tidak digunakan, maka nilai inisialisasi variabel adalah `NULL`. Penamaan variabel lokal bersifat *case insensitive*. Berikut adalah beberapa contoh deklarasi variabel:

```
DECLARE total_sale INT
DECLARE x, y INT DEFAULT 0
```

Pemberian nilai ke sebuah variabel dilakukan dengan menggunakan statement SET. Hasil dari query juga dapat dimasukkan ke dalam variabel menggunakan SELECT ... INTO. Berikut adalah beberapa contoh pemberian nilai ke variabel.

```
SET total_sale = 50;
SELECT COUNT(*) INTO numEmployee FROM employee;
```

Ruang lingkup variabel adalah di antara blok `BEGIN ... END` di mana variabel tersebut didefinisikan. Variabel dapat diakses dari blok yang berada dalam blok di mana ia didefinisikan, kecuali pada blok yang memiliki deklarasi nama variabel yang sama. Berikut adalah contoh penggunaan variabel dalam function dan stored procedure.

```
mysql> CREATE FUNCTION addTax(salary FLOAT(8,2))
    -> RETURNS FLOAT (8,2)
    -> BEGIN
    -> DECLARE tax FLOAT DEFAULT 0.05;
    -> RETURN salary * (1 - tax);
    -> END ^_^
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

Pada contoh di atas, dibuat sebuah function dengan variabel bernama `tax`. Variabel ini diset memiliki nilai `default 0.05` dan digunakan untuk mengubah nilai `salary`. Contoh di bawah ini menunjukkan penggunaan function `addTax`.

```
mysql> SELECT first_name, addTax(salary) FROM employee;
+------------+----------------+
| first_name | addTax(salary) |
```

```
+------------+----------------+
| John       |        1995.00 |
| Alison     |        7374.64 |
| James      |        7263.49 |
| Celia      |        3273.49 |
| Robert     |        3263.99 |
| Linda      |        5152.59 |
| David      |        8548.84 |
| Hercule    |        5143.28 |
| Lincoln    |        4098.28 |
| Sherlock   |        4963.00 |
+------------+----------------+
10 rows in set (0.00 sec)
```

Nama variabel lokal seharusnya tidak sama dengan nama kolom dalam tabel database. Jika pada statement SQL seperti SELECT terdapat referensi ke kolom tabel dengan nama yang sama, MySQL mereferensikannya sebagai nama variabel. Berikut adalah contohnya.

```
mysql> DELIMITER **
mysql> CREATE PROCEDURE checkScope()
    -> BEGIN
    -> DECLARE first_name VARCHAR(15) DEFAULT 'bob';
    -> SELECT id, first_name FROM employee WHERE first_name = first_name;
    -> END **
Query OK, 0 rows affected (0.54 sec)

mysql> DELIMITER ;
mysql> CALL checkScope();
+------------+
| first_name |
+------------+
| bob        |
| bob        |
| bob        |
| bob        |
| bob        |
| bob        |
| bob        |
| bob        |
| bob        |
| bob        |
+------------+
```

```
10 rows in set (0.06 sec)


Query OK, 0 rows affected (0.06 sec)
```

Pada contoh di atas, ketika kita melakukan pemilihan `SELECT` untuk `first_name`, nilai yang ditampilkan adalah nilai default dari variable `first_name`, yaitu `'bob'`.

4.2 Kendali Kondisional

Seperti layaknya bahasa pemrograman, kita juga bisa mendefinisikan kendali kondisional di dalam function dan procedure. Kendali kondisional yang disediakan dalam MySQL adalah `IF` dan `CASE`.

4.2.1 Kendali IF

Sintaks dasar dari IF adalah sebagai berikut:

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] …
    [ELSE statement_list]
END IF;
```

Nilai `search_condition` dievaluasi. Jika bernilai true, maka `statement_list` setelah `THEN` dijalankan. Namun jika bernilai false, maka `statement_list` pada `ELSE` yang dijalankan. Penggunaan banyak kondisi dapat dilakukan dengan statement `ELSEIF`.

Berikut adalah contoh penggunaan `IF`:

```
mysql> DELIMITER &&
mysql> CREATE FUNCTION hideSalary(salary FLOAT(8,2))
    -> RETURNS VARCHAR(20)
    -> BEGIN
    -> DECLARE sal VARCHAR(20);
    -> IF salary < 4000 THEN SET sal = 'Low Salary';
    -> ELSE SET sal = 'High Salary';
    -> END IF;
    -> RETURN sal;
    -> END &&
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT first_name, last_name, hideSalary(salary)
    -> FROM employee;
+------------+-----------+--------------------+
| first_name | last_name | hideSalary(salary) |
+------------+-----------+--------------------+
| John       | Doe       | Low Salary         |
| Alison     | Mathews   | High Salary        |
```

```
| James      | Smith     | High Salary        |
| Celia      | Rice      | Low Salary         |
| Robert     | Black     | Low Salary         |
| Linda      | Green     | High Salary        |
| David      | Larry     | High Salary        |
| Hercule    | Poirot    | High Salary        |
| Lincoln    | Rhyme     | High Salary        |
| Sherlock   | Holmes    | High Salary        |
+------------+-----------+--------------------+
10 rows in set (0.03 sec)
```

4.2.2 Kendali CASE

Sintaks dari kendali CASE adalah sebagai berikut:

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Pada sintaks di atas, `case_value` dibandingkan dengan semua nilai `when_value` sampai ditemukan yang sesuai. Jika ditemukan, maka `statement_list` pada `WHEN` yang bersesuaian akan dijalankan. Jika tidak ada nilai `when_value` yang sesuai, maka `statement_list` pada `ELSE` yang dijalankan (jika ada).

Berikut adalah contoh penggunaan CASE:

```
mysql> DELIMITER ##
mysql> CREATE FUNCTION calcTax(job VARCHAR (20)))
    -> RETURNS FLOAT(3,2)
    -> BEGIN
    -> DECLARE tax FLOAT(3,2) DEFAULT 0.05;
    -> CASE job
    -> WHEN 'Manager' THEN SET tax = 0.1;
    -> WHEN 'Programmer' THEN set tax = 0.07;
    -> WHEN 'Tester' THEN set tax = 0.06;
    -> ELSE SET tax = 0.05;
    -> END CASE;
    -> RETURN tax;
    -> END ##
Query OK, 0 rows affected (0.06 sec)

mysql> delimiter ;
mysql> SELECT first_name, last_name, calcTax(description) FROM employee;
+------------+-----------+----------------------+
```

```
| first_name | last_name | calcTax(description) |
+------------+-----------+----------------------+
| John       | Doe       |                 0.07 |
| Alison     | Mathews   |                 0.06 |
| James      | Smith     |                 0.06 |
| Celia      | Rice      |                 0.10 |
| Robert     | Black     |                 0.06 |
| Linda      | Green     |                 0.06 |
| David      | Larry     |                 0.10 |
| Hercule    | Poirot    |                 0.05 |
| Lincoln    | Rhyme     |                 0.05 |
| Sherlock   | Holmes    |                 0.05 |
+------------+-----------+----------------------+
10 rows in set (0.00 sec)
```

Bentuk sintaks dari CASE yang lain adalah sebagai berikut:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Pada sintaks di atas, `search_condition` di setiap klausa `WHEN` dievaluasi hingga ditemukan klausa `WHEN` yang sesuai. Jika tidak ada klausa `WHEN` yang sesuai, maka klausa `ELSE` yang dijalankan. Jika tidak ada klausa `ELSE` ketika semua klausa `WHEN` tidak sesuai, maka akan terjadi `Case not found for CASE statement` error. Berikut adalah contoh penggunaan sintaks `CASE … WHEN` tersebut:

```
mysql> DELIMITER >>
mysql> CREATE FUNCTION calcTax2(job VARCHAR(20))
    -> RETURNS FLOAT(3,2)
    -> BEGIN
    -> DECLARE tax FLOAT(3,2);
    -> CASE
    -> WHEN job = 'Manager' THEN SET tax = 0.1;
    -> WHEN job = 'Programmer' THEN SET tax = 0.07;
    -> WHEN job = 'Tester' THEN SET tax = 0.06;
    -> ELSE SET tax = 0.05;
    -> END CASE;
    -> RETURN tax;
    -> END >>
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER ;
mysql> SELECT first_name, last_name, calcTax2(description) FROM employee;
```

```
+------------+-----------+-----------------------+
| first_name | last_name | calcTax2(description) |
+------------+-----------+-----------------------+
| John       | Doe       |                  0.07 |
| Alison     | Mathews   |                  0.06 |
| James      | Smith     |                  0.06 |
| Celia      | Rice      |                  0.10 |
| Robert     | Black     |                  0.06 |
| Linda      | Green     |                  0.06 |
| David      | Larry     |                  0.10 |
| Hercule    | Poirot    |                  0.05 |
| Lincoln    | Rhyme     |                  0.05 |
| Sherlock   | Holmes    |                  0.05 |
+------------+-----------+-----------------------+
10 rows in set (0.34 sec)
```

4.3 Perulangan

Pada function dan procedure juga disediakan perulangan. Beberapa bentuk perulangan yang disediakan dalam MySQL adalah `WHILE`, `REPEAT ... UNTIL`, dan `LOOP`.

4.3.1 Perulangan WHILE

Bentuk sintaks untuk perulangan WHILE adalah sebagai berikut:

```
WHILE search_condition DO
    statement_list
END WHILE
```

`Statement_list` yang terdapat dalam `WHILE` diulang selama `search_condition` bernilai true. `statement_list` terdiri atas satu atau lebih statement SQL, setiap statementnya dipisahkan dengan delimiter titik koma (;). Berikut adalah contoh penggunaan `WHILE`.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE mod12(IN number INT(10))
    -> BEGIN
    -> WHILE number MOD 12 > 0 DO
    -> SET number = number + 1;
    -> END WHILE;
    -> SELECT number;
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```

```
mysql> CALL mod12(10);
+--------+
| number |
+--------+
|     12 |
+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL mod12(24);
+--------+
| number |
+--------+
|     24 |
+--------+
1 row in set (0.00 sec)

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

4.3.2 Perulangan REPEAT ... UNTIL

Sintaks dari REPEAT ... UNTIL adalah sebagai berikut:

```
REPEAT
    statement_list
UNTIL search_condition
END REPEAT
```

Statement_list di dalam REPEAT dilakukan secara berulang hingga ekspresi search_condition bernilai true. Oleh karena itu, sebuah REPEAT memasuki perulangan paling sedikit sebanyak satu kali. statment_list terdiri atas satu atau lebih statement, masing-masing dipisah dengan delimiter titik koma (;). Berikut adalah contoh penggunaan REPEAT ... UNTIL.

```
mysql> DELIMITER /.
mysql> CREATE PROCEDURE repeatDemo(IN number INT(10))
    -> BEGIN
    -> REPEAT
    -> SET number = number +1;
    -> UNTIL number MOD 12 = 0
    -> END REPEAT;
    -> SELECT number;
    -> END /.
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELIMITER ;
mysql> CALL repeatDemo(10);
+--------+
| number |
+--------+
|     12 |
+--------+
1 row in set (0.00 sec)

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

4.3.3 Perulangan LOOP

Sintaks dari perulangan LOOP adalah sebagai berikut:

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

`LOOP` merupakan bentuk perulangan sederhana. Perulangan dilakukan terhadap `statement_list`, yang terdiri atas beberapa statement dengan dipisahkan oleh tanda titik koma (;). Statement di dalam `LOOP` diulang sampai `LOOP` berakhir. Cara mengakhiri `LOOP` biasanya dilakukan dengan statement `LEAVE`. Tanda perulangan dilakukan menggunakan `ITERATE`. Berikut adalah contoh penggunaan `LOOP`.

```
mysql> DELIMITER /?
mysql> CREATE PROCEDURE iterateDemo(number INT)
    -> BEGIN
    -> label1: LOOP
    ->       SET number = number + 1;
    ->       IF number MOD 12 > 0 THEN
    ->             ITERATE label1;
    ->       END IF;
    ->       LEAVE label1;
    -> END LOOP label1;
    -> SELECT number;
    -> END /?
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> CALL iterateDemo(10);
+--------+
| number |
+--------+
|     12 |
```

```
+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL iterateDemo(20);
+--------+
| number |
+--------+
|     24 |
+--------+
1 row in set (0.00 sec)

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

5 LATIHAN

Tabel employee_join

```
+----+------------+-----------+------------+------------+---------+-----------+--------+
| id | first_name | last_name | start_date | end_date   | salary  | city      | job_id |
+----+------------+-----------+------------+------------+---------+-----------+--------+
|  1 | Jason      | Martin    | 1996-07-25 | 2006-07-25 | 1235.56 | Toronto   |      1 |
|  2 | Alison     | Mathews   | 1976-03-21 | 1986-02-21 | 6662.78 | Vancouver |      2 |
|  3 | James      | Smith     | 1978-12-12 | 1990-03-15 | 6545.78 | Vancouver |      2 |
|  4 | Celia      | Rice      | 1982-10-24 | 1999-04-21 | 2345.78 | Vancouver |      3 |
|  5 | Robert     | Black     | 1984-01-15 | 1998-08-08 | 2335.78 | Vancouver |      2 |
|  6 | Linda      | Green     | 1987-07-30 | 1996-01-04 | 4323.78 | New York  |      2 |
|  7 | David      | Larry     | 1990-12-31 | 1998-02-12 | 7898.78 | New York  |      3 |
|  8 | James      | Cat       | 1996-09-17 | 2002-04-15 | 1233.78 | Vancouver |      2 |
| 10 | Hercule    | Poirot    | 1973-05-23 | 2001-08-09 | 4313.98 | Brussels  |      5 |
| 11 | Lincoln    | Rhyme     | 1999-05-25 | 2011-07-13 | 3213.98 | New York  |      6 |
+----+------------+-----------+------------+------------+---------+-----------+--------+
```

Tabel job

```
+--------+------------+
| job_id | title      |
+--------+------------+
|      1 | Programmer |
|      2 | Tester     |
|      3 | Manager    |
|      4 | Spy        |
|      5 | Detective  |
|      6 | Forensics  |
|      7 | Developer  |
+--------+------------+
7 rows in set (0.00 sec)
```

1. Buatlah function untuk menampilkan gabungan first_name dan last_name dengan bentuk “last_name, first_name”. Lihat contoh hasil di bawah ini:

```
mysql> SELECT revName(first_name, last_name) FROM employee;
+--------------------------------+
| revName(first_name, last_name) |
+--------------------------------+
```

```
| Doe, John                      |
| Mathews, Alison                |
| Smith, James                   |
| Rice, Celia                    |
| Black, Robert                  |
| Green, Linda                   |
| Larry, David                   |
| Poirot, Hercule                |
| Rhyme, Lincoln                 |
| Holmes, Sherlock               |
+--------------------------------+
```

2. Buatlah procedure untuk menampilkan job_description dari masukan sebuah id employee. Berikut adalah contoh outputnya.

```
mysql> CALL empJob(4);
+------------+-----------+---------+
| first_name | last_name | title   |
+------------+-----------+---------+
| Celia      | Rice      | Manager |
+------------+-----------+---------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL empJob(2);
+------------+-----------+--------+
| first_name | last_name | title  |
+------------+-----------+--------+
| Alison     | Mathews   | Tester |
+------------+-----------+--------+
1 row in set (0.00 sec)

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

MODUL ENAM

Pertemuan 6 Latihan Soal

Petunjuk pengerjaan

- Jawaban ditulis menggunakan pengolah kata dan disimpan dengan ekstensi *.doc
- File jawaban diberi nama dengan format **`DB06-nomor-nama.doc`**
- Setiap soal dijawab dengan **menuliskan query** saja, **tidak perlu** menggunakan **screenshoot**.
- Jawaban yang telah selesai dikirim

  ke email: **`kuliah@arifn.web.id`**

  subject: **`[DB06]nomor-nama`**

Sebuah database universitas memiliki 3 buah tabel sebagai berikut:

`instruktur (nip, namains, jurusan, asalkota)`

`matakuliah (nomk, namamk, sks)`

`kuliah (nip, nomk, ruangan, jmlmhs)`

Soal

1. Tuliskan query untuk membuat ketiga tabel tersebut. Tipe data menyesuaikan dengan nilai setiap kolom, seperti contoh data sebagai berikut:

Tabel `instruktur`

nip	namains	jurusan	asalkota
1	Steve Wozniak	Ilmu Komputer	Bantul
2	Steve Jobs	Seni Rupa	Solo
3	James Gosling	Ilmu Komputer	Klaten
4	Bill Gates	Ilmu Komputer	Magelang

Tabel `matakuliah`

nomk	namamk	sks
KOM101	Algoritma dan Pemrograman	3
KOM102	Basis Data	3
SR101	Desain Elementer	3
KOM201	Pemrograman Berorientasi Objek	3

Tabel `kuliah`

nip	nomk	ruangan	jmlmhs
1	KOM101	101	50
1	KOM102	102	35

nip	nomk	ruangan	jmlmhs
2	SR101	101	45
3	KOM201	101	55

2. Tuliskan **query** untuk mendapatkan data-data di bawah ini. Tambahkan data pada tabel sesuai dengan kebutuhan.
   a) Instruktur-instruktur jurusan 'Ilmu Komputer'
   b) Nomor mata kuliah yang pesertanya lebih dari 40 orang
   c) Nomor dan mata kuliah yang pesertanya lebih dari 40 orang
   d) nip instruktur yang mengampu mata kuliah dengan nomor 'KOM102'
   e) nip instruktur yang mengampu mata kuliah 'Basis Data'
   f) nip dan nama instruktur yang mengampu mata kuliah 'Basis Data'
   g) Nama mata kuliah dan ruangan yang diampu oleh 'Steve Jobs'
   h) Jumlah total mahasiswa yang diampu oleh 'Steve Wozniak'
   i) Nomor dan nama instruktur yang mengampu mahasiswa terbanyak
   j) Nomor dan nama instruktur yang belum mengampu mata kuliah apapun

3. Buatlah **view** untuk mendapatkan data berikut ini:
   a) Nomor dan nama instruktur yang belum mengampu mata kuliah apapun
   b) Jumlah mata kuliah yang diampu oleh setiap instruktur

4. Buatlah Trigger untuk pencatatan perubahan ruangan untuk sebuah mata kuliah. Catatan perubahan disimpan dalam tabel berikut:

Nama kolom	Tipe data	Keterangan
user_id	VARCHAR(15)	User MySQL yang melakukan perubahan. Didapatkan dari fungsi user()
deskripsi	VARCHAR(100)	Dituliskan seperti contoh sebagai berikut: 'Merubah ruangan KOM121 dari ruang 101 ke ruang 102'

Contoh hasil

```
mysql> update kuliah set ruangan=102 where nomk='KOM101';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from roomChanges;
+----------------+-----------------------------------------------------+
| user_id        | deskripsi                                           |
+----------------+-----------------------------------------------------+
| root@localhost | Merubah ruangan KOM101 dari ruang 101 ke ruang 102 |
+----------------+-----------------------------------------------------+
1 row in set (0.00 sec)
```

5. Buatlah fungsi atau prosedur sesuai kasus berikut ini:

   a) Fungsi untuk menampilkan jumlah kuliah yang diadakan di sebuah ruangan (nama ruangan dimasukkan sebagai input)

   contoh pemanggilan

```
mysql> select countRoom('102');
+------------------+
| countRoom('102') |
+------------------+
|                2 |
+------------------+
1 row in set (0.03 sec)

mysql> select countRoom('103');
+------------------+
| countRoom('103') |
+------------------+
|                0 |
+------------------+
1 row in set (0.00 sec)
```

   b) Fungsi untuk mendapatkan nama ruangan tempat sebuah mata kuliah diadakan (nomor mata kuliah dimasukkan sebagai input). Berikan nilai 'not found' jika sebuah mata kuliah belum diberi ruangan.

   Contoh pemanggilan

```
mysql> select getRoom('KOM102');
+-------------------+
| getRoom('KOM102') |
+-------------------+
| 102               |
+-------------------+
1 row in set (0.00 sec)

mysql> select getRoom('KOM103');
+-------------------+
| getRoom('KOM103') |
+-------------------+
| not found         |
+-------------------+
1 row in set, 1 warning (0.00 sec)
```

c) Prosedur untuk menampilkan nama mata kuliah dan ruangan yang diampu oleh seorang instruktur (nama instruktur dimasukkan sebagai input)

Contoh hasil:

```
mysql> call showRoom('Steve Jobs');
+------------------+---------+
| namamk           | ruangan |
+------------------+---------+
| Desain Elementer | 101     |
+------------------+---------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call showRoom('Steve Wozniak');
+---------------------------+---------+
| namamk                    | ruangan |
+---------------------------+---------+
| Algoritma dan Pemrograman | 102     |
| Basis Data                | 102     |
+---------------------------+---------+
2 rows in set (0.00 sec)

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

d) Prosedur untuk menampilkan jumlah SKS yang diampu oleh seorang instruktur (nama instruktur dimasukkan sebagai input)

Contoh hasil

```
mysql> call getSks('Steve Jobs');
+------------+----------+
| namains    | sum(sks) |
+------------+----------+
| Steve Jobs |        3 |
+------------+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> call getSks('Steve Wozniak');
+---------------+----------+
| namains       | sum(sks) |
+---------------+----------+
```

```
| Steve Wozniak |        6 |
+---------------+----------+
1 row in set (0.00 sec)

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