UGM | UNIVERSITAS GADJAH MADA

Modul

Praktikum Basis Data

MII212502

Laboratorium Komputer Dasar
DIKE FMIPA UGM
2023

Daftar Isi

Daftar Isi 1
BAB I ENTITY RELATIONSHIP 5
1.1. Capaian Pembelajaran 5
1.2. Dasar Teori 5
1.2.1. Himpunan Entitas 5
1.2.2. Atribut 5
1.2.3. Relasi 6
1.2.4. Key 7
1.2.5. Entity-Relationship Diagram 8
1.3. Aktivitas Mahasiswa 10
1.4. Latihan 10
BAB II DATA DEFINITION LANGUAGE (DDL): CREATE, ALTER, DROP 11
2.1. Capaian Pembelajaran 11
2.2. Dasar Teori 11
2.2.1. Create Database 12
2.2.2. Menampilkan Database 13
2.2.3. Memilih Database 13
2.2.4. Alter Database 13
2.2.5. Drop Database 14
2.2.6. Create Table 14
2.2.7. Alter Table 16
2.2.8. Drop Table 17
2.3. Aktivitas Mahasiswa 18
2.4. Pertanyaan 18
BAB III DATA MANIPULATION LANGUAGE (DML) 1: INSERT, UPDATE, DELETE 19
3.1. Capaian Pembelajaran 19
3.2. Dasar Teori 19
3.2.1. SELECT 20
3.2.2. INSERT 23
3.2.3. UPDATE 26
3.2.4. DELETE 29
3.3. Aktivitas Mahasiswa 31
3.4. Pertanyaan 31
BAB IV DATA MANIPULATION LANGUAGE (DML) 2 32

4.1. Capaian Pembelajaran 32
4.2. Dasar Teori 32
4.2.1. Pengurutan Data (Sorting Data) 32
4.2.2. Pemilihan Data (Filtering Data) 34
4.3. Aktivitas Mahasiswa 40
4.4. Pertanyaan 41
BAB V DATA MANIPULATION LANGUAGE (DML) 3 42
5.1. Capaian Pembelajaran 42
5.2. Dasar Teori 42
5.2.1. Fungsi Agregat 42
5.2.2. Pengelompokan Data (Grouping Data) 45
5.3. Aktivitas Mahasiswa 47
5.4. Pertanyaan 47
BAB VI SELECT JOIN 48
6.1. Capaian Pembelajaran 48
6.2. Dasar Teori 48
6.2.1. Inner Join 49
6.2.2. Left (Outer) Join 50
6.2.3. Right (Outer) Join 51
6.2.4. Self Join 51
6.2.5. Union 52
6.3. Aktivitas Mahasiswa 53
BAB VII SUBQUERY 54
7.1. Capaian Pembelajaran 54
7.2. Dasar Teori 54
7.2.1. Perbandingan menggunakan Subquery 55
7.2.2. Subquery menggunakan ANY, IN, SOME, dan ALL 56
7.2.3. Subquery menggunakan EXISTS 57
7.2.4. Subquery menggunakan klausa FROM 58
7.2.5. Subquery pada INSERT dan UPDATE 59
7.3. Aktivitas Mahasiswa 60
BAB VIII STORED PROCEDURE 61
8.1. Capaian Pembelajaran 61
8.2. Dasar Teori 61
8.2.1. Definisi Prosedur Tersimpan 61
8.2.2. Menulis itu pertama MySQL disimpan prosedur 62

8.2.3. Variabel Prosedur Tersimpan MySQL 63
8.2.4. Parameter Prosedur Tersimpan MySQL 65
8.2.5. MySQL IF Statement 68
8.2.6. MySQL CASE Statement 73
8.3. Aktivitas Mahasiswa 77
BAB IX FUNCTION DAN TRIGGER 78
9.1. Capaian Pembelajaran 78
9.2. Dasar Teori 78
9.2.1. MySQL Stored Function 78
9.2.2. MySQL Trigger 80
9.3. Aktivitas Mahasiswa 85
BAB X NORMALISASI 86
10.1. Capaian Pembelajaran 86
10.2. Dasar Teori 86
10.2.1. Proses Normalisasi 86
10.2.2. Anomali 86
10.2.3. Tahapan Normalisasi 88
10.2.4. Contoh Normalisasi 89
10.2.5. Ringkasan Aturan Normalisasi 95
10.3 Aktivitas Mahasiswa 95
REFERENSI 98

BAB I
ENTITY RELATIONSHIP

1.1. Capaian Pembelajaran

1. Mahasiswa dapat memahami simbol ERD
2. Mahasiswa dapat menggambar ERD berdasarkan kasus yang diberikan

1.2. Dasar Teori

Dalam ERD, ada tiga hal penting, yaitu: *entity set* , atribut , dan relasi.

1.2.1. Himpunan Entitas

Himpunan entitas *(entity set)* adalah objek yang ada di dunia nyata yang berbeda dengan objek lainnya, memiliki atribut penyusun dan sebagai pembangun suatu sistem. Misalnya, seorang pria yang bekerja di suatu perusahaan adalah sebuah *entity*. Entitas *(entity)* memiliki atribut bernilai *(values)*, misalnya: 000-11-3452 adalah sebuah nomor induk seorang pekerja (atribut nomor induk pekerja). Selain itu, seorang pekerja juga memiliki tanggal lahir. Di sini, tanggal lahir merupakan atribut dari *entity* pekerja yang berkedudukan sejajar dengan atribut nomor induk pekerja.

Entity memiliki beberapa tipe atribut sebagai berikut:

1. Atribut sederhana *(simple attribute)*: atribut dengan nilai sederhana, tidak dapat dibagi lagi menjadi atribut yang lebih kecil.
2. Atribut komposit *(composite attribute)*: atribut yang dapat dibagi menjadi atribut yang lebih kecil. Misalnya: atribut nama dapat dibagi menjadi nama depan, nama tengah, dan nama belakang.
3. Atribut bernilai tunggal *(single-valued attribute)*: atribut yang hanya dapat berisi satu nilai. Misalnya: nomor induk pekerja.
4. Atribut multi-nilai *(multi-valued attribute)*: atribut yang dapat berisi nol, satu atau lebih dari satu nilai. Misalnya: atribut nomor telepon, bisa saja seorang pekerja memiliki satu atau lebih nomor telepon.
5. Atribut turunan *(derivative attributes)*: atribut yang dapat diturunkan dari atribut lain. Misalnya: atribut umur, nilainya bisa diturunkan dari tanggal lahir dan sekarang tanggal.

1.2.2. Atribut

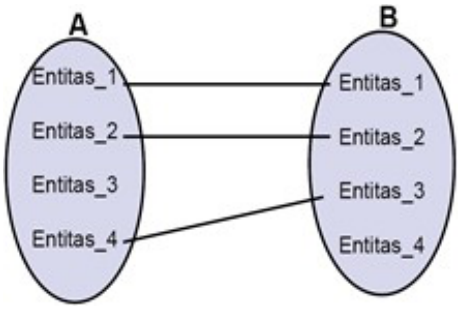
Atribut adalah deskripsi yang menjelaskan suatu entitas, misalnya departemen untuk entitas karyawan atau NIM, nama, fakultas, jurusan untuk entitas mahasiswa.

1.2.3. Relasi

Relasi *(relationship)* menjelaskan hubungan antar entitas. Misalnya pada contoh di atas entitas manusia memiliki relasi dengan alamat yaitu "tinggal di". Ketika merancang *database*, suatu entitas harus memiliki relasi dengan entitas lain, minimal dengan satu entitas. Jika ada entitas dalam *database* yang tidak memiliki relasi dengan entitas lain, maka akan terjadi kesalahan dalam desain. Biasanya entitas yang tidak terkait akan dihilangkan.

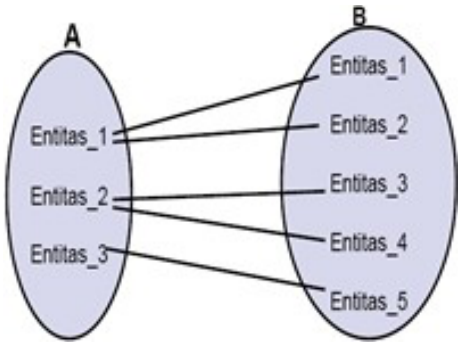
Terdapat empat jenis relasi antar entitas antara lain:

1. *One to one*



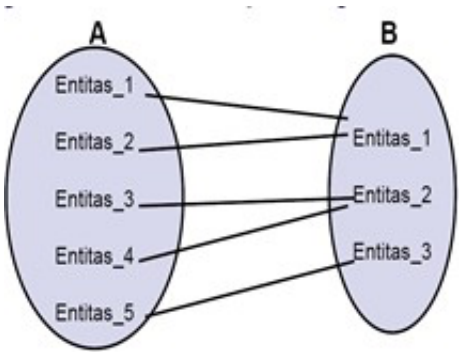
   Setiap entitas pada himpunan entitas A berelasi paling banyak dengan satu entitas pada himpunan entitas B, dan sebaliknya setiap entitas pada himpunan entitas B berelasi paling banyak dengan satu entitas pada himpunan entitas A.

2. *One to many*



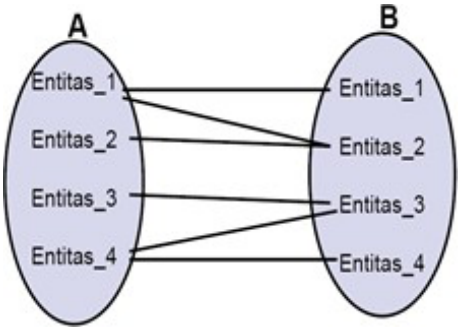
   Setiap entitas pada himpunan entitas A berelasi dengan banyak entitas pada himpunan entitas B, tetapi tidak sebaliknya, dimana setiap entitas pada himpunan entitas B paling banyak berhubungan dengan satu entitas pada himpunan entitas A.

3. *Many to one*



Setiap entitas pada himpunan entitas A berelasi paling banyak dengan satu entitas pada himpunan entitas B, tetapi tidak sebaliknya, dimana setiap entitas pada himpunan entitas B berhubungan dengan banyak entitas pada himpunan entitas A.

4. *Many to many*



Setiap entitas pada himpunan entitas A dapat berelasi dengan banyak entitas pada himpunan entitas B, begitu pula sebaliknya, dimana setiap entitas pada himpunan entitas B dapat berelasi dengan banyak entitas pada himpunan entitas A.

1.2.4. *Key*

Kunci *(key)* merupakan satu atau gabungan dari beberapa atribut yang dapat mengidentifikasi *record*/baris dalam suatu tabel secara unik. *Key* terdiri dari:

1. *Super key:* satu atau gabungan beberapa atribut (kolom) yang dapat membedakan setiap baris dalam suatu tabel secara unik.
2. *Candidate key:* kumpulan atribut minimal yang dapat dijadikan sebagai identitas untuk membedakan setiap *record* dalam sebuah tabel secara unik, bisa terdapat satu atau lebih *candidate key. Candidate key* disebut juga dengan *minimal super key,* yaitu *super key* yang tidak mengandung *super key* yang lain. Setiap *candidate key* pasti merupakan *super key,* namun tidak semua *super key* akan menjadi *candidate key.*
3. *Primary key: candidate key* yang dapat dijadikan sebagai acuan utama karena dapat mengidentifikasi setiap data dalam suatu tabel secara unik.
4. *Alternate key: candidate key* yang tidak dijadikan *primary key.*
5. *Composite key: key* yang terdiri dari 2 atribut atau lebih. Atribut tersebut apabila berdiri sendiri tidak menjadi identitas *record,* tetapi apabila digabungkan dapat mengidentifikasi secara unik.
6. *Foreign key:* atribut yang menjadi *primary key* pada suatu tabel tetapi pada tabel lain atribut tersebut hanya sebagai atribut biasa *(non-key).*

Primary Key

Seperti yang telah dijelaskan sebelumnya, *entity* adalah suatu objek yang berbeda dengan objek lain. Kemudian setiap *entity* harus memiliki atribut yang dapat membedakan suatu objek dengan objek

lainnya. Sebagai contoh entitas manusia, entitas manusia tidak memiliki atribut untuk membedakan antara objek manusia1 dengan objek manusia2. Namun, jika manusia menjadi seorang karyawan, kemudian objek manusia1 dan manusia2 sebagai anggota entitas karyawan akan memiliki atribut pembeda (nomor ID pekerja). Atribut itu disebut atribut kunci *(key attribute)*.

Key dapat terdiri atas dari beberapa atribut, umumnya disebut dengan *candidate key*. Namun, dalam praktiknya hampir merupakan kewajiban bahwa kunci atribut hanya terdiri dari satu atribut. Hal ini untuk memudahkan dalam pengolahan data. Untuk mendapatkan *key* dari *candidate key* (kombinasi beberapa atribut), pilih atribut yang paling unik dari entitas tersebut (tidak ada nilai yang sama di dalam atribut) dan dapat dibedakan antara anggota entitas dengan anggota entitas lainnya. Jika sudah mendapatkannya, *key* ini disebut dengan *primary key*.

Misalkan terdapat *candidate key* berupa nama, NIM, dan tanggal lahir. Kemudian, perlu diurutkan lagi mana yang merupakan atribut paling unik dari ketiga atribut tersebut. Atribut nama tidak unik; karena mungkin ada dua entitas pekerja memiliki nama yang identik. Sama halnya dengan atribut tanggal lahir, bisa saja terdapat dua entitas pekerja yang memiliki tanggal lahir yang sama. Hanya atribut NIP saja yang tentunya berbeda dari setiap entitas pekerja.

Foreign Key

Kita telah melihat bahwa entitas dalam *database* harus memiliki relasi dengan entitas lain. Oleh karena itu, entitas yang memiliki relasi dengan entitas lain harus memuat *primary key* dari entitas yang memiliki relasi dengannya. Misalnya, entitas Karyawan memiliki NIP sebagai *primary key*-nya. Entitas Karyawan ini memiliki relasi dengan entitas Pekerjaan, dengan *primary key* ID bagian/jabatan pekerjaan. Untuk dapat menghubungkan kedua entitas tersebut, entitas Karyawan tidak perlu memasukkan bagian/jabatan pekerjaan ke dalam salah satu atributnya, tetapi hanya perlu memasukkan *primary key* dari entitas Pekerjaan (ID bagian/jabatan pekerjaan). Proses menggunakan *primary key* dari entitas lain untuk menjadi atribut suatu entitas disebut juga dengan *foreign key*.

Terdapat beberapa dasar untuk memuat *primary key* dari suatu entitas ke entitas lain yang berhubungan. Landasan ini memakai karakteristik relasi yang digunakan.

1. *One to one*: Entitas A berhubungan dengan entitas B secara *one to one*, maka *primary key* entitas A dimuat ke dalam entitas B atau sebaliknya.
2. *Many to one*: Entitas A berhubungan dengan entitas B secara *many to one*, maka *primary key* entitas B dimuat ke dalam entitas A.
3. *Many to many*: Entitas A berhubungan dengan entitas B secara *many to many*, maka pemuatan *primary key* dari masing-masing entitas akan melibatkan suatu entitas baru.

1.2.5. *Entity-Relationship Diagram*

E-R Diagram digunakan untuk membuat suatu model *database*. Kemudian dari model tersebut dibuatlah sistem *database*. Adapun macam-macam komponen dalam *E-R Diagram* adalah :

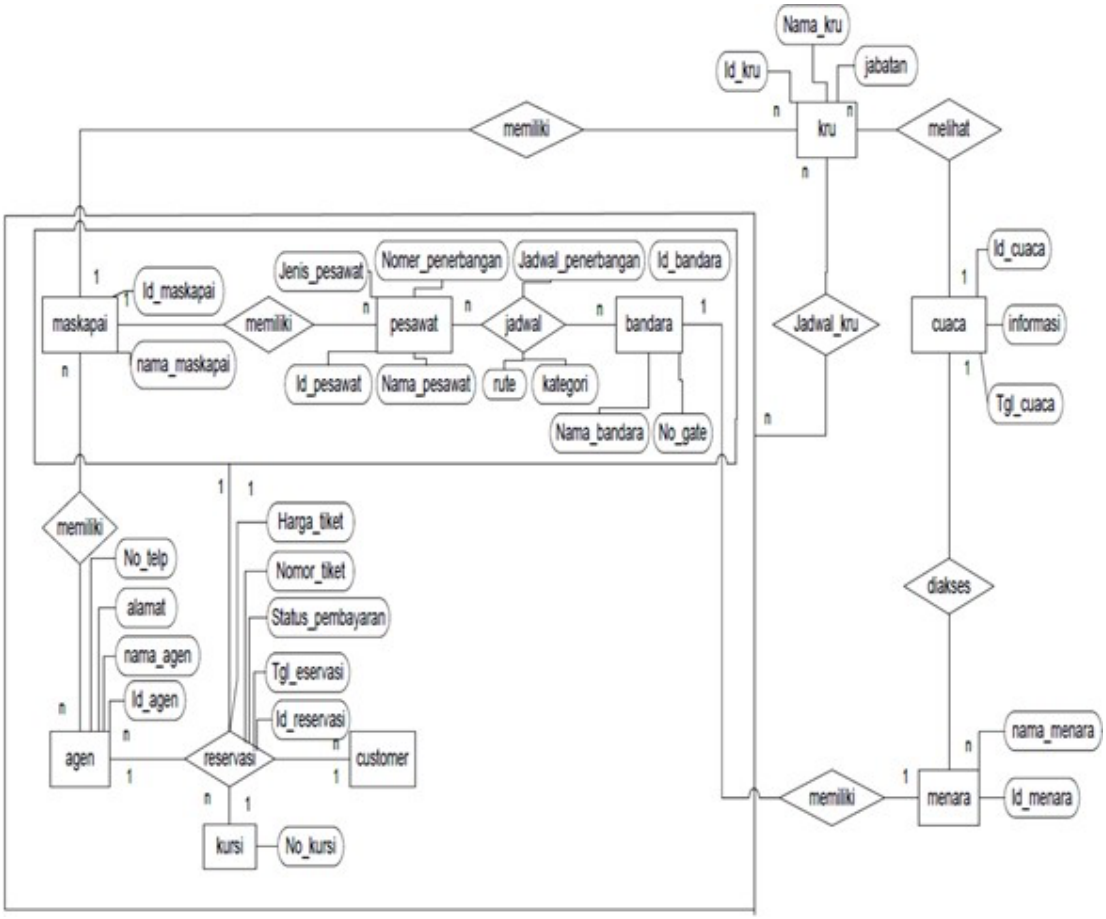
1. *Rectangle*: merepresentasikan entitas
2. *Elips*: merepresentasikan atribut
3. *Rhombus*: merepresentasikan relasi antara entitas
4. *Line*: merepresentasikan hubungan antara atribut
5. *Double Ellipse* : merepresentasikan atribut yang memiliki banyak nilai
6. *Ellipse Dashed*: merepresentasikan atribut turunan
7. *Double Lines*: merepresentasikan total partisipasi entitas dalam relasi
8. *Double Rectangle*: merepresentasikan entitas lemah

Terdapat dua jenis entitas; yang pertama adalah entitas yang kuat yaitu entitas yang memiliki *primary key,* yang kedua adalah entitas yang lemah yaitu entitas yang tidak memiliki *primary key*. Entitas kuat dan lemah akan dibahas lebih lanjut pada materi normalisasi.

Contoh *E-R Diagram*

Pada setiap bandara sudah memiliki jadwal penerbangan untuk semua maskapai, baik penerbangan domestik maupun internasional. Dimana pihak bandara telah melakukan penjadwalan terhadap pesawat dan rute penerbangan baik keberangkatan maupun kedatangan. Petugas pengatur lalu lintas udara yang biasa disebut *air traffic controllers (ATC)* memiliki peranan yang penting terutama pada pesawat untuk mencegah antar pesawat tidak terlalu dekat satu sama lain, untuk mencegah tabrakan antar pesawat dan menghindari hambatan di sekitar pesawat selama operasi. Selain itu *ATC* memiliki peran dalam mengatur kelancaran lalu lintas, membantu pilot dalam pengendalian keadaan darurat, memberikan informasi yang dibutuhkan pilot (seperti informasi cuaca, informasi navigasi penerbangan, dan informasi lalu lintas udara). *ATC* merupakan rekan terdekat pilot selama di udara, peran *ATC* sangat besar dalam tercapainya tujuan penerbangan. Dalam sistem pelayanan penerbangan memiliki beberapa layanan yang ditawarkan melalui *web* diantaranya adalah mengikuti:

1. Pihak bandara dapat melihat situasi cuaca saat ini, baik atau buruk untuk melakukan penerbangan, status penerbangan (*landing, takeoff, delay*), kedatangan, penerbangan dan tujuan pesawat.
2. Maskapai dapat melihat jadwal tugas penerbangan dari dan ke mana, siapa pilot, pramugari dan awak-awaknya.
3. Agen dapat menjual, memesan *(booking)*, memantau harga pesawat dan jadwal penerbangan.
4. *Customer* dapat melihat jadwal penerbangan, rute penerbangan, melihat harga tiket, dan memesan tiket.



1.3. Aktivitas Mahasiswa

1. Mahasiswa mempelajari dan mencoba membuat ERD pada Gambar 1.1.
2. Mahasiswa mengerjakan Latihan 1.4.

1.4. Latihan

Buatlah sebuah model *database* dari sistem perpustakaan. Tentukan entitas-entitas yang ada di dalamnya dan jelaskan atribut-atribut penyusun entitas. Tentukan *primary key,* entitas yang kuat dan lemah, dan relasi dari entitas tersebut. Setelah itu, gambarlah model *database* tersebut dalam ERD.

BAB II

DATA DEFINITION LANGUAGE (DDL): CREATE, ALTER, DROP

2.1. Capaian Pembelajaran

1. Mahasiswa dapat menjelaskan konsep *Data Definition Language (DDL)* .
2. Mahasiswa dapat menggunakan perintah dasar *(DDL)* untuk mendefinisikan objek dari *database*.
3. Mahasiswa dapat membuat *database*, mengubah *database* dan menghapus *database*.

2.2. Dasar Teori

Pada *MySQL*, suatu *database* adalah sekumpulan objek yang digunakan untuk menyimpan dan memanipulasi data seperti tabel-tabel, menampilkan *(view) database*, *trigger*, dan *stored procedure*. *Data Definition Language (DDL)* adalah standar untuk perintah yang mendefinisikan struktur yang berbeda dalam *database* . Perintah *DDL* digunakan untuk membuat, memodifikasi, dan menghapus objek *database* seperti sebagai tabel, indeks, dan *user*. Ketentuan ini juga dikenal sebagai bahasa deksripsi data *(data description language)* dalam beberapa konteks, *DDL* mendeskripsikan *fields* dan *records* dalan tabel *database.* Perintah *DDL* yang umum adalah *CREATE*, *ALTER*, dan *DROP*.

Implementasi

Industri *database* saat ini memasukkan DDL ke dalam *formal language* yang mendeskripsikan data. Namun, hal ini dianggap sebagai bagian dari *SQL (Structured Query Language)*. SQL sering menggunakan kata kerja imperatif dengan Bahasa Inggris normal seperti sebagai kalimat untuk mengimplementasikan modifikasi. Oleh karena itu, *DDL* tidak muncul sebagai bahasa yang berbeda dalam *database SQL*, tetapi mendefinisikan perubahan dalam skema *database*.

DDL yang umum digunakan dalam kueri *SQL* adalah:

- *CREATE*: Perintah ini membuat tabel baru dan memiliki sintaks yang sudah ditentukan sebelumnya. Sintaks perintah `CREATE TABLE [nama_tabel] ([definisi_kolom]) [parameter_tabel]`. Misalnya `CREATE TABLE Employee (Employee_ID INTEGER PRIMARY KEY, Firstname CHAR (50) NULL, Lastname CHAR (75) NOT NULL)`.
- *ALTER*: Perintah ini digunakan untuk mengubah/memodifikasi tabel yang sudah ada pada *database*. Perintah ini dapat digunakan untuk menambahkan kolom tambahan, menghapus kolom yang ada dan mengubah tipe data kolom yang ada dalam suatu tabel *database*. Sintaks perintah *alter* adalah `ALTER [tipe_objek] [nama objek] [parameter]`. Misalnya, `ALTER TABLE Karyawan ADD DOB Tanggal`.
- *DROP*: Perintah ini digunakan untuk menghapus tabel, indeks, atau tampilan *(view)*. Sintaks

perintah *drop* adalah `DROP [tipe_objek] [nama objek]`. Misalnya, `DROP TABLE Karyawan`.

2.2.1. *Create Database*

Sebelum melakukan hal lain dengan data, Anda perlu membuat *database*. *Database* adalah wadah *(container)* dari data. *Database* menyimpan kontak, vendor, pelanggan atau setiap jenis data yang dapat Anda pikirkan. Di dalam MySQL, suatu *database* adalah sekumpulan objek yang digunakan untuk menyimpan dan memanipulasi data seperti tabel-tabel, menampilkan *(view) database*, *trigger*, dan *stored procedure*

Untuk membuat *database* baru melalui program MySQL, Anda dapat menggunakan langkah-langkah berikut:

Pertama, masuk ke Server *MySQL* menggunakan *user* `root`

```
1 >mysql -u root -p
2 Enter password: ********
```

Catatan: jika *user* `root` Anda tidak memiliki kata sandi, Anda dapat menulis: `mysql –u root`

Untuk membuat database di *MySQL*, Anda menggunakan pernyataan *`CREATE DATABASE`* sebagai berikut:

```
1 CREATE DATABASE [IF NOT EXISTS] database_name;
```

Mari kita bahas perintah *`CREATE DATABASE`* secara lebih rinci:

- Perintah *`CREATE DATABASE`* diikuti oleh nama *database* yang hendak dibuat. Nama *database* sebaiknya merupakan nama yang bermakna dan deskriptif.
- *`IF NOT EXISTS`* merupakan klausa opsional dari perintah *`CREATE DATABASE`*. Klausa *`IF NOT EXISTS`* mencegah Anda dari suatu kesalahan ketika membuat suatu *database* baru yang sudah ada di *database server*. Anda tidak dapat memiliki 2 *database* dengan nama yang sama dalam database *MySQL server*.

Sebagai contoh, untuk membuat *database classicmodels*, Anda bisa menjalankan *statement* *`CREATE DATABASE`* sebagai berikut.

```
1 CREATE DATABASE classicmodels;
```

Setelah menjalankan *statement* tersebut ini, *MySQL* akan mengembalikan pesan *(message)* untuk memberi tahu apakah *database* baru telah berhasil dibuat atau tidak.

2.2.2. Menampilkan *Database*

Perintah `SHOW DATABASES` menampilkan semua *database* yang ada pada server *database MySQL*. Perintah `SHOW DATABASES` dapat digunakan untuk memeriksa *database* yang dibuat atau untuk melihat semua *database* di *server database* sebelum membuat *database* baru, misalnya:

```
mysql> SHOW DATABASES;

+--------------------+
| Database           |
+--------------------+
| classicmodels      |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Seperti yang ditunjukkan dengan jelas pada *output*, terdapat tiga *database* di *server database MySQL*. Pada output tersebut, `information_schema` dan `mysql` merupakan *database* bawaan yang tersedia ketika kita meng-install MySQL, dan `classicmodels` merupakan *database* baru yang dibuat.

2.2.3. Memilih *Database*

Sebelum menggunakan suatu *database* tertentu, Anda harus memberi tahu *MySQL* terkait *database* mana yang akan digunakan menggunakan perintah `USE`.

```
USE database_name;
```

Misalnya untuk memilih *database classicmodels* menggunakan *statement* `USE` sebagai berikut.

```
USE classicmodels;
```

Mulai saat ini, semua operasi seperti membuat kueri data, membuat tabel baru atau pemanggilan *stored procedure* yang dilakukan akan berefek pada *database* yang dipilih saat ini, yaitu *database classicmodels*.

Untuk keluar dari program *MySQL*, ketik perintah `exit`

```
mysql> exit
Bye
```

2.2.4. *Alter Database*

Perintah `ALTER DATABASE` digunakan ke memodifikasi, memelihara, atau memulihkan suatu *database* yang sudah ada. *MySQL* menyediakan fungsi *ALTER* yang membantu kita menggabungkan perubahan pada desain *database* yang sudah ada. Perintah *ALTER* digunakan untuk memodifikasi database yang ada, tabel, *view* atau objek *database* lain yang mungkin perlu diubah selama *life cycle database*.

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | DEFAULT ENCRYPTION [=] {'Y' | 'N'}
```

2.2.5. *Drop Database*

Menghapus *database* berarti menghapus semua tabel yang terdapat dalam *database* dan *database* itu sendiri secara permanen. Oleh karena itu, sangat penting untuk menjalankan kueri ini dengan ekstra hati-hati.

```
DROP DATABASE [IF EXISTS] database_name;
```

Klausa `DROP DATABASE` diikuti dengan nama *database* yang hendak dihapus. Serupa dengan perintah `CREATE DATABASE`, *IF EXIST* merupakan bagian opsional dari *statement* ini untuk mencegah dari menghapus *database* yang tidak ada di *database server*.

Untuk mengimplementasikan *statement* `DROP DATABASE`, buatlah *database* baru, pastikan *database* tersebut berhasil dibuat, kemudian hapus *database* tersebut. Mari kita lihat kueri berikut:

```
CREATE DATABASE IF NOT EXISTS tempdb;
SHOW DATABASES;
DROP DATABASE IF EXISTS temp_database;
```

Urutan ketiga *statement* tersebut sebagai berikut:

1. Pertama, membuat *database* bernama `tempdb` menggunakan *statement* `CREATE DATABASE`.
2. Kedua, menampilkan semua *database* menggunakan *statement* `SHOW DATABASES`.
3. Ketiga, menghapus *database* `tempdb` menggunakan *statement* `DROP DATABASE`.

2.2.6. *Create Table*

Untuk membuat tabel baru di dalam *database*, Anda menggunakan perintah MySQL `CREATE TABLE`. *Statement* `CREATE TABLE` merupakan salah satu *statement* yang paling kompleks di MySQL. Perintah berikut ini mengilustrasikan sintaks *statement* MySQL `CREATE TABLE` dengan yang telah disederhanakan.

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_list
) ENGINE=storage_engine
```

Pertama, tentukan nama tabel yang akan dibuat setelah klausa `CREATE TABLE`. Nama tabel harus unik dalam suatu *database.* Klausa `IF NOT EXISTS` adalah klausa opsional yang memungkinkan Anda memeriksa apakah tabel yang akan dibuat sudah ada di dalam *database*. Apabila demikian, *MySQL* akan mengabaikan seluruh *statement* dan tidak akan membuat tabel baru.

Sangat disarankan untuk menggunakan klausa `IF NOT EXISTS` pada setiap *statement* `CREATE TABLE` untuk menghindari kesalahan membuat tabel baru yang sudah ada.

Kedua , tentukan daftar kolom untuk tabel pada bagian `column_list`, kolom-kolom dipisahkan dengan koma.

Ketiga, tentukan mesin penyimpanan *(storage engine)* untuk tabel pada klausa `engine` (opsional). Jika *storage engine* tidak dideklarasikan, *MySQL* akan menggunakan *InnoDB* secara *default*.

Untuk mendefinisikan suatu kolom pada tabel di dalam *statement* `CREATE TABLE`, gunakan sintaks berikut ini:

```
1 column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]
```

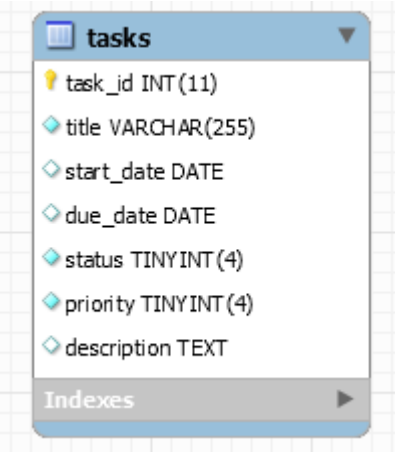
Komponen terpenting dari sintaks di atas adalah:

- `column_name` menentukan nama dari kolom tersebut. Setiap kolom memiliki tipe data panjang maksimum yang spesifik, misalnya, `VARCHAR (255)`
- `NOT NULL` menunjukkan bahwa kolom melakukan tidak diperbolehkan kosong *(NULL)*.
- `DEFAULT VALUE` digunakan untuk menspesifikan nilai *default* dari kolom.
- `AUTO_INCREMENT` menunjukkan bahwa nilai dari kolom tersebut dihasilkan secara otomatis setiap kali baris baru dimasukkan ke dalam tabel. Setiap tabel memiliki satu dan hanya satu kolom *AUTO_INCEREMENT*.

Untuk mengatur suatu kolom atau sekumpulan kolom sebagai *primary*, gunakan sintaks berikut:

```
1 PRIMARY KEY (col1,col2,...)
```

Statement berikut digunakan untuk membuat tabel baru yang bernama *tasks:*



```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    PRIMARY KEY (task_id)
)  ENGINE=INNODB;
```

2.2.7. *Alter Table*

Perintah *ALTER TABLE* memodifikasi itu struktur dari suatu tabel yang sudah ada. Perintah ini umumnya digunakan untuk menambahkan suatu kolom, menghapus suatu kolom, mengubah tipe data suatu kolom, menambahkan *primary key*, mengganti nama tabel, dan banyak lagi. Berikut ini mengilustrasikan sintaks *statement* `ALTER TABLE`:

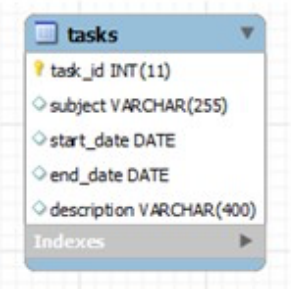
```
ALTER TABLE table_name action1[,action2,…]
```

Untuk mengubah struktur tabel yang sudah ada:

- Pertama, tentukan nama tabel yang akan diubah, setelah klausa `ALTER TABLE`.
- Kedua, tentukan daftar sekumpulan aksi *(action)* yang akan diterapkan ke tabel. Sebuah aksi dapat berupa apapun seperti menambahkan kolom baru, menambahkan *primary key*, atau mengganti nama tabel. *Statement* `ALTER TABLE` dapat mengaplikasikan beberapa aksi dalam satu *statement* `ALTER TABLE`, beberapa aksi tersebut dipisahkan dengan koma (,).

Statement berikut membuat tabel baru yang dinamai *tasks* dalam sampel *database.*

```
CREATE TABLE tasks (
    task_id INT NOT NULL,
    subject VARCHAR(45) NULL,
    start_date DATE NULL,
    end_date DATE NULL,
    description VARCHAR(200) NULL,
    PRIMARY KEY (task_id),
    UNIQUE INDEX task_id_unique (task_id ASC)
);
```



Misalkan kita menginginkan nilai kolom *task_id* akan bertambah secara otomatis setiap memasukkan baris baru ke dalam tabel *tasks*. Untuk melakukan ini, gunakan *statement* `ALTER TABLE` untuk mengatur atribut dari kolom *task_id* menjadi *AUTO_INCEREMENT* sebagai berikut:

```
ALTER TABLE tasks
CHANGE COLUMN task_id task_id INT(11) NOT NULL AUTO_INCREMENT;
```

Untuk menambahkan kolom baru yang disebut *complete* untuk mencatat persentase penyelesaian setiap *task* di dalam tabel *tasks.* Dalam hal ini, gunakan `ALTER TABLE` untuk menambahkan suatu kolom baru ke tabel *tasks* sebagai berikut:

```
ALTER TABLE tasks
ADD COLUMN complete DECIMAL(2,1) NULL
AFTER description;
```

Misalkan kita tidak ingin menyimpan deskripsi tugas di dalam tabel *tasks* dan kita hendak menghapus kolom tersebut. *Statement* berikut ini digunakan untuk menghapus kolom deskripsi pada tabel *tasks:*

```
ALTER TABLE tasks
DROP COLUMN description;
```

Catatan:

- Perintah untuk mengganti nama tabel di *database* menggunakan sintaks berikut:

```
RENAME TABLE old_table_name to new_table_name
```

- Perintah untuk menampilkan struktur tabel menggunakan sintaks berikut:

```
DESCRIBE table_name
```

- Perintah untuk mengganti nama kolom yang sudah ada pada suatu tabel menggunakan sintaks berikut:

```
ALTER TABLE tablename change old_column_name  new_column_name
new_data_type(size);
```

2.2.8. *Drop Table*

Untuk menghapus tabel yang ada, Anda menggunakan *statement* `DROP TABLE` pada MySQL. Pada contoh ini, kita akan menghapus tabel `tasks` yang sudah dibuat menggunakan `CREATE TABLE` pada tutorial sebelumnya.

Selain itu, kita mencoba menghapus tabel yang tidak ada di *database* untuk mendemonstrasikan *statement SHOW WARNING. Statement* untu menghapus tabel `tasks` dan tabel yang tidak ada yang dinamai `nonexistent_tabel` sebagai berikut:

```
1 DROP TABLE IF EXISTS tasks, nonexistent_table;
```

```
1 SHOW WARNINGS;
```

	Level	Code	Message
▶	Note	1051	Unknown table 'nonexistent_table'

2.3. Aktivitas Mahasiswa

1. Buatlah *database* akademik, kemudian buatlah tabel mahasiswa (dengan atribut: NIM, nama, jenis kelamin, tempat lahir, tanggal lahir, telepon dan pembimbing akademik), dosen (terdiri dari atribut, NIP, nama, jenis kelamin, jabatan, minat, dan telepon), mata kuliah (terdiri dari atribut: kode mata kuliah, nama mata kuliah, SKS, dosen pengampu, hari, jam, dan ruang kelas), KRS (terdiri dari atribut ID KRS, kode mata kuliah, NIM, tahun, semester, dan nilai). Setiap *field/*kolom sebagai *primary key* harus didefinisikan *auto increment* dan *not null*.
2. Tampilkan semua tabel yang ada di *database*.
3. Tampilkan struktur dari setiap tabel yang telah dibuat dibuat.
4. Buatlah *query* kueri untuk melakukan hal berikut hal-hal:
   - Hapus *(drop)* primary key pada "NIP".
   - Tambahkan kembali *primary key* pada "NIP".
   - Lihat kolom tabel "dosen".
   - Ubah nama tabel "dosen" menjadi "profesor"
   - Ubah nama atribut "nama" dengan " nama_dosen ".
   - Ubah tipe data "jenis_kelamin" menjadi enum {'laki-laki', 'perempuan'}.
   - Ubah tipe data "telepon" menjadi int.

2.4. Pertanyaan

1. Apa perbedaan antara *Create*, *Alter* dan *Drop* pada *Data Definition Language*?
2. Perintah apa yang seharusnya digunakan untuk mendefinisikan sebuah objek dari *database*?
3. Alih-alih menghapus suatu *database*, apa perintah yang sebaiknya digunakan untuk nama suatu *database*?

BAB III

DATA MANIPULATION LANGUAGE (DML) 1: INSERT, UPDATE, DELETE

3.1. Capaian Pembelajaran

1. Mahasiswa dapat mengimplementasikan perintah *Insert*, *Update*, dan *Delete.*
2. Mahasiswa dapat memodifikasi entri tabel.
3. Mahasiswa dapat memodifikasi struktur tabel.

Catatan: Selama praktikum kita akan menggunakan database *"classicmodels"* pada link berikut: http://ugm.id/classicmodels

3.2. Dasar Teori

Data Manipulation Language (DML) adalah bahasa yang digunakan *user* untuk mengakses dan memanipulasi data. Manipulasi data merujuk ke pengambilan, penambahan, penghapusan, dan modifikasi dari data atau informasi yang disimpan di dalam suatu *database*. Tujuan utamanya adalah untuk mencapai interaksi manusia dengan sistem secara efisien. *MySQL* menyediakan beberapa perintah untuk memodifikasi tabelnya: *Insert*, *Update* dan *Delete*.

a) **SELECT** – Perintah ini merupakan perintah *DML* yang paling banyak digunakan. Perintah ini mengambil informasi atau *record* dari *database*.

Sintaks:

```
SELECT [column name(s)] from [table name]
```

Contoh:

Untuk memilih semua *record* dari tabel *employee* yang memiliki *salary* lebih besar dari 2000.

```
SELECT * FROM EMPLOYEE WHERE SALARY > 2000
```

b) **UPDATE** – Perintah ini digunakan untuk modifikasi tabel. Perintah ini dapat memodifikasi data dari satu atau beberapa *record* sesuai dengan kondisi yang ditentukan.

Sintaks:

```
UPDATE [table name] SET [column name = value] where [condition]
```

Contoh:

Untuk memperbarui *salary* dari seorang karyawan (id = 101) yang telah dipromosikan dengan penambahan 5000.

```
UPDATE EMPLOYEE SET SALARY = SALARY + 5000 WHERE EMPLOYEE_ID =101
```

c) **INSERT** – Perintah ini digunakan untuk menambahkan satu atau beberapa *record* ke suatu *database*.

Sintaks:

```
INSERT INTO [table name] [column(s)] VALUES [value(s)]
```

Contoh:

Masukkan detail karyawan baru yang telah bergabung dengan organisasi.

```
INSERT INTO EMPLOYEE (ID, NAME, SALARY) VALUES ('102','MAYANK','10000')
```

d) **DELETE** - Perintah ini digunakan untuk menghapus satu atau beberapa *record* dari suatu *database* sesuai dengan kondisi yang ditentukan.

Sintaks:

```
DELETE FROM [table name] where [condition]
```

Contoh:

Mengapus *record* karyawan (id = 100) yang telah keluar dari perusahaan.

```
DELETE FROM EMPLOYEE WHERE EMPLOYEE_ID = 100
```

Data manipulation language pada awalnya digunakan hanya di dalam komputer program tetapi dengan hadirnya *SQL*, *database administrator* menggunakannya karena lebih baik secara interaktif.

3.2.1. *SELECT*

Statement SELECT mengambil data dari satu atau lebih tabel. Sebuah tabel terdiri dari baris dan kolom seperti suatu *spreadsheet*. Hasil dari itu *statement SELECT* disebut sebagai himpunan hasil *(result set)* yang merupakan daftar baris *(row)*, dimana setiap baris terdiri dari jumlah kolom yang sama. Perhatikan tabel *employees* pada contoh sampel *database* berikut. Tabel tersebut memiliki delapan kolom: *employee number*, *last name, first name, extension, email, office code, reports to, job title* dan banyak baris. Untuk memilik SEMUA kolom, gunakan `ALL` atau `(*)`:

Sekarang, mari kita lihat sintaks *statement* `SELECT`:

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees
```

Meskipun tabel *employees* memiliki banyak kolom, *statement SELECT* hanya mengembalikan dari dari tiga kolom dari semua baris yang ada pada tabel seperti yang ditunjukkan pada gambar berikut.

employeeNumb	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143	Sales Rep
1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep
1337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1102	Sales Rep
1370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1102	Sales Rep
1401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1102	Sales Rep
1501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1102	Sales Rep

Hasilnya sebagai berikut:

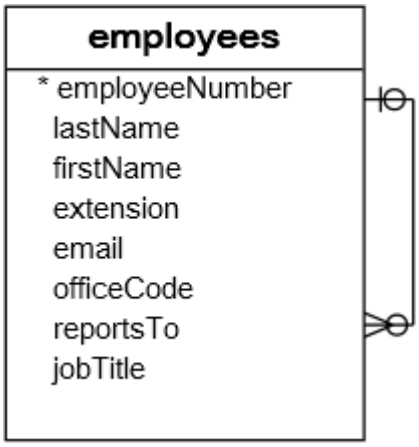
lastname	firstname	jobtitle
Murphy	Diane	President
Patterson	Mary	VP Sales
Firrelli	Jeff	VP Marketing
Patterson	William	Sales Manager (APAC)
Bondur	Gerard	Sale Manager (EMEA)
Bow	Anthony	Sales Manager (NA)
Jennings	Leslie	Sales Rep
Thompson	Leslie	Sales Rep
Firrelli	Julie	Sales Rep
Patterson	Steve	Sales Rep
Tseng	Foon Yue	Sales Rep
Vanauf	George	Sales Rep
Bondur	Loui	Sales Rep
Hernandez	Gerard	Sales Rep

Klausa SELECT DISTINCT

Ketika meminta data dari tabel, kami mungkin mendapatkan baris duplikat. Untuk menghapus baris yang duplikat ini gunakan klausa `DISTINCT` pada *statement* `SELECT`. Sintaks yang menggunakan klasua `DISTINCT` sebagai berikut:

```
SELECT DISTINCT
    select_list
FROM
    table_name;
```

Perhatikan contoh sederhana dari penggunaan klausa `DISTINCT` untuk memilih nama terakhir *(lastname)* unik pegawai dari tabel *employees.*



Pertama, buat kueri untuk mendapatkan *lastname* pegawai dari tabel *employees* menggunakan *statement* `SELECT` berikut:

```
SELECT
    lastname
FROM
    employees
ORDER BY
    lastname;
```

Hasilnya sebagai berikut:

	lastname
	Bondur
	Bondur
	Bott
	Bow
	Castillo
	Firrelli
	Firrelli
	Fixter
	Gerard
	Hernandez
	Jennings

Untuk menghapus nama belakang yang duplikat, tambahkan klausa *DISTINCT* pada *statement SELECT* sebagai berikut:

```
SELECT
    DISTINCT lastname
FROM
    employees
ORDER BY
    lastname;
```

Sekarang, hasilnya hanya akan memuat nilai unik saja :

	lastname
	Bondur
	Bott
	Bow
	Castillo
	Firrelli
	Fixter
	Gerard
	Hernandez
	Jennings
	Jones

3.2.2. *INSERT*

Statement `INSERT` menambahkan/menyisipkan satu atau lebih baris ke dalam tabel. Berikut ini ilustrasi sintaks dari *statement* `INSERT`:

```
1 INSERT INTO table(c1,c2,...)
2 VALUES (v1,v2,...);
```

Dalam sintaks ini,

- Pertama, tentukan nama tabel dan daftar kolom yang dipisahkan koma di dalam tanda kurung setelah klausa `INSERT INTO`.
- Lalu, letakkan daftar nilai yang dipisahkan koma sesuai dengan urutan kolom di dalam tanda kurung setelah klausa `VALUES`

Jumlah kolom dan nilai-nilai harus sama. Sebagai tambahan, posisi dari kolom harus sesuai dengan posisi nilai-nilainya.

Untuk menambahkan beberapa baris ke dalam satu tabel dengan satu *statement* INSERT, gunakan sintaks berikut:

```
INSERT INTO table(c1,c2,...)
VALUES
   (v11,v12,...),
   (v21,v22,...),
    ...
   (vnn,vn2,...);
```

Pada sintaks ini, baris dipisahkan dengan koma pada klausa `VALUES`.

Mari buat tabel baru bernama *tasks* untuk mempraktekkan *statement* `INSERT`.

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    priority TINYINT NOT NULL DEFAULT 3,
    description TEXT,
    PRIMARY KEY (task_id)
);
```

Statement berikut menambahkan baris baru ke tabel *tasks*:

```
INSERT INTO
 tasks(title,priority)
VALUES
 ('Learn MySQL INSERT Statement',1);
```

MySQL mengembalikan pesan berikut setelah *statement* dieksekusi:

```
1 row(s) affected
```

Anda dapat memverifikasinya dengan menggunakan kueri berikut:

```
SELECT
    *
FROM
    tasks;
```

Berikut ini hasilnya:

	task_id	title	start_date	due_date	priority	description
▶	1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL

Menambahkan baris menggunakan nilai *default*

Untuk menambahkan nilai default ke dalam kolom, terdapat dua cara sebagai berikut:

- Pertama, abaikan nama kolom dan nilainya dalam *statement* `INSERT`.
- Kedua, tentukan nama kolom pada klausa `INSERT INTO` dan gunakan *keyword* `DEFAULT` di klausa `VALUES`.

Contoh berikut menunjukkan demonstrasi menggunakan cara kedua:

```
INSERT INTO
 tasks(title,priority)
VALUES
 ('Understanding DEFAULT keyword in INSERT statement',DEFAULT);
```

Contoh ini menspesifikasikan kolom *priority* dan *keyword* `DEFAULT`. Karena nilai *default* untuk kolom *priority* adalah 3 seperti yang dideklarasikan dalam tabel definisi:

```
priority TINYINT NOT NULL DEFAULT 3
```

MySQL menggunakan angka 3 untuk memasukkan ke dalam kolom *priority*. Berikut ini menunjukkan isi tabel *tasks* setelah *insert*:

```
SELECT
    *
FROM
    tasks;
```

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT	NULL	NULL	3	NULL

Menambakan nilai tanggal *(date)* ke dalam tabel

Untuk memasukkan nilai tanggal literal ke dalam suatu kolom, Anda dapat menggunakan format berikut:

```
'YYYY-MM-DD'
```

Pada format ini:

- YYYY merepresentasikan tahun dalam empat digit misalnya, 2018.
- MM merepresentasikan bulan dalam dua digit, misalnya 01, 02, dan 12.
- DD merepresentasikan hari dalam dua digit, misalnya 01, 02, 30.

Contoh berikut menambahkan baris baru ke tabel *tasks* dengan nilai *start_date* dan *due_date*:

```
INSERT INTO tasks(title, start_date, due_date)
VALUES('Insert date into table','2018-01-09','2018-09-15');
```

Gambar berikut menunjukkan isinya dari tabel *tasks* setelah *insert*:

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL

Dimungkinkan pula untuk menggunakan ekspresi pada klausa `VALUES`. Sebagai contoh, *statement* berikut menambahkan sebuah *task* baru menggunakan *current date* untuk kolom *start_date* dan *due_date*:

```
INSERT INTO tasks(title,start_date,due_date)
VALUES
 ('Use current date for the task',CURRENT_DATE(),CURRENT_DATE())
```

Pada contoh ini, fungsi `CURRENT_DATE()` mengatur nilai untuk kolom *start_date* dan *due_date.* Sebagai catatan, fungsi `CURRENT_DATE()` merupakan fungsi tanggal yang mengembalikan tanggal pada sistem saat ini. Berikut isi dari tabel *tasks* setelah *insert*:

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL
4	Use current date for the task	2018-09-02	2018-09-02	3	NULL

Menambahkan beberapa baris

Statement berikut menambahkan tiga baris ke tabel *tasks*:

```
INSERT INTO tasks(title, priority)
VALUES
 ('My first task', 1),
 ('It is the second task',2),
 ('This is the third task of the week',3);
```

Tabel *tasks* tersebut memiliki data berikut setelah *insert:*

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL
4	Use current date for the task	2018-09-02	2018-09-02	3	NULL
5	My first task	NULL	NULL	1	NULL
6	It is the second task	NULL	NULL	2	NULL
7	This is the third task of the week	NULL	NULL	3	NULL

3.2.3. *UPDATE*

Statement `UPDATE` akan memperbarui data yang ada di dalam suatu tabel atau merubah nilai-nilai kolom dari satu baris, sekelompok baris, atau semua baris dalam tabel. Berikut ini ilustrasi sintaks *statement* MySQL `UPDATE`:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
    column_name1 = expr1,
    column_name2 = expr2,
    ...
[WHERE
    condition];
```

Di dalam *statement* `UPDATE`:

- Pertama, tentukan nama tabel untuk memperbarui data setelah *keyword* `UPDATE`.
- Kedua, klausa `SET` menentukan kolom mana yang hendak diubah dan nilai baru. Untuk memperbarui banyak kolom, gunakan daftar *assignment* yang dipisahkan koma. Masukkan nilai pada setiap kolom *assignment* dalam bentuk nilai literal, *expression* , atau *subquery*.
- Ketiga, tentukan baris mana yang akan diperbarui menggunakan kondisi tertentu dalam klausa `WHERE`. Klausa `WHERE` opsional.

MySQL mendukung dua *modifier* pada *statement* `UPDATE`:

- *Modifier* `LOW_PRIORITY` menginstruksikan *statement* `UPDATE` untuk menunda pembaruan sampai tidak ada koneksi membaca data dari tabel.
- *Modifier* `IGNORE` memungkinkan *statement* `UPDATE` untuk terus memperbarui baris meskipun terjadi kesalahan. Baris yang menyebabkan munculnya *error* seperti konflik kunci duplikat yang tidak diperbarui.

Memodifikasi nilai-nilai pada satu kolom

Dalam contoh ini, kita akan memperbarui *email* dari Mary Patterson menjadi email baru mary.patterso@classicmodelcars.com.

Pertama, untuk memastikan *update* berhasil, kita akan menulis kueri *email* Mary dari tabel *employees* menggunakan *statement* `SELECT` berikut:

```
SELECT
    firstname, lastname, email
FROM
    employees
WHERE
    employeeNumber = 1056;
```

	firstname	lastname	email
▶	Mary	Patterson	mpatterso@classicmodelcars.com

Sekarang gunakan *statement* `UPDATE` seperti yang ditunjukkan kueri berikut:

```
1 UPDATE employees
2 SET
3     email = 'mary.patterson@classicmodelcars.com'
4 WHERE
5     employeeNumber = 1056;
```

Jalankan *statement* `UPDATE` kembali untuk memverifikasi perubahan.

```
1 SELECT
2     firstname, lastname, email
3 FROM
4     employees
5 WHERE
6     employeeNumber = 1056;
```

	firstname	lastname	email
▶	Mary	Patterson	mary.patterson@classicmodelcars.com

Memodifikasi nilai-nilai pada beberapa kolom

Untuk memperbarui nilai pada beberapa kolom, tentukan *assignment* pada klausa `SET`. Misalnya, *statement* berikut digunakan untuk memperbarui kolom nama belakang dan email dari pegawai dengan *employee number* 1056:

```
1 UPDATE employees
2 SET
3     lastname = 'Hill',
4     email = 'mary.hill@classicmodelcars.com'
5 WHERE
6     employeeNumber = 1056;
```

Mari kita periksa perubahannya:

```
1 SELECT
2     firstname, lastname, email
3 FROM
4     employees
5 WHERE
6     employeeNumber = 1056;
```

	firstname	lastname	email
▶	Mary	Hill	mary.hill@classicmodelcars.com

3.2.4. *DELETE*

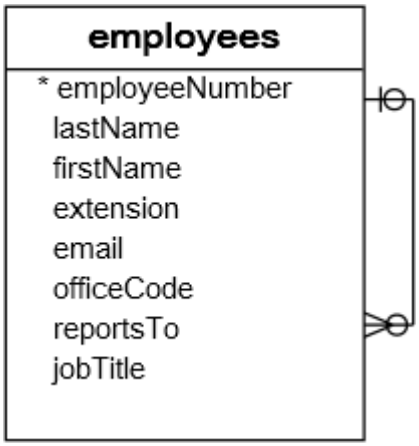
Untuk menghapus data dari suatu tabel, gunakan *statement MySQL* `DELETE`. Berikut ini ilustrasinya sintaks dari *statement* `DELETE`:

```
DELETE FROM table_name
WHERE condition;
```

Dalam *statement* ini:

- Pertama, tentukan tabel dari data mana yang akan dihapus.
- Kedua, gunakan kondisi untuk menentukan baris mana yang akan dihapus dalam klausa *WHERE*. Jika baris cocok dengan kondisi tersebut, makan baris tersebut akan dihapus.

Selain menghapus data dari tabel, *statement* `DELETE` juga mengembalikan jumlah baris yang dihapus. Kita akan menggunakan tabel *employees* untuk demonstrasi.



Misalkan Anda ingin menghapus karyawan yang memiliki *officeNumber* 4, gunakan *statement* `DELETE` dengan menggunakan klausa `WHERE` seperti yang ditunjukkan dengan kueri berikut:

```
DELETE FROM employees
WHERE
    officeCode = 4;
```

Untuk menghapus semua baris dari tabel *employees,* Anda dapat menggunakan *statement* `DELETE` tanpa menggunakan klausa `WHERE` seperti berikut:

```
DELETE FROM employees;
```

Klausa `DELETE` dan `LIMIT`

Untuk membatasi jumlah baris yang akan dihapus, Anda dapat menggunakan klausa `LIMIT` sebagai berikut:

```
DELETE FROM table
LIMIT row_count;
```

Perhatikan bahwa urutan baris dalam tabel tidak ditentukan, oleh karena itu, saat Anda menggunakan klausa `LIMIT`, Anda harus selalu menggunakan klausa `ORDER BY`.

```
DELETE FROM table_name
ORDER BY c1, c2, ...
LIMIT row_count;
```

Misalnya, *statement* berikut mengurutkan pelanggan berdasarkan nama pelanggan menurut abjad dan menghapus 10 pelanggan pertama:

```
DELETE FROM customers
ORDER BY customerName
LIMIT 10;
```

Demikian pula, *statement* DELETE berikut memilih pelanggan di Prancis, mengurutkannya berdasarkan *credit limit* dari rendah ke tinggi, dan menghapus 5 pelanggan pertama:

```
DELETE FROM customers
WHERE country = 'France'
ORDER BY creditLimit
LIMIT 5;
```

3.3. Aktivitas Mahasiswa

Gambar berikut adalah *Invoice* Pelanggan *Pine Valley Furniture Company (PVFC)*. Berdasarkan *invoice* tersebut, buatlah databasenya dengan setidaknya 5 baris data *(tupple)* untuk setiap tabel!

PVFC Customer Invoice

Customer ID	2	**Order ID**	1006
Customer Name	Value Furniture	**Order Date**	10/24/2006
Address	15145 S.W. 17th St. Plano TX 750		

Product ID	Product Description	Finish	Quantity	Unit Price	Extended Price
7	Dining Table	Natural Ash	2	$800.00	$1,600.00
5	Writers Desk	Cherry	2	$325.00	$650.00
4	Entertainment Center	Natural Maple	1	$650.00	$650.00
				Total	$2,900.00

3.4. Pertanyaan

1. Jelaskan cara menggunakan perintah *Insert*, *Update*, dan *Delete*!
2. Perintah apa yang digunakan untuk mengubah nilai beberapa kolom dari suatu tabel?

BAB IV

DATA MANIPULATION LANGUAGE (DML) 2

4.1. Capaian Pembelajaran

1. Mahasiswa dapat mengenal *DML* pada MySQL seperti *SELECT, DISTINCT, WHERE, ORDER, LIMIT, AND, OR, IN, BETWEEN, LIKE, IS NULL*
2. Mahasiswa dapat menggunakan *DML* pada MySQL seperti *SELECT, DISTINCT, WHERE, ORDER, LIMIT, AND, OR, IN, BETWEEN, LIKE, IS NULL*

4.2. Dasar Teori

4.2.1. Pengurutan Data *(Sorting Data)*

ORDER BY

Ketika Anda menggunakan *statement* `SELECT` untuk kueri data dari suatu tabel, himpunan hasil tidak diurutkan dengan urutan apapun. Untuk mengurutkan himpunan hasil, Anda dapat menggunakan klausa `ORDER BY`. Klausa `ORDER BY` memungkinkan Anda untuk:

- Mengurutkan himpunan hasil berdasarkan satu kolom atau beberapa kolom.
- Mengurutkan himpunan hasil berdasarkan kolom yang berbeda dalam urutan menaik *(ascending)* atau menurun *(descending)*.

Berikut ini ilustrasi sintaks dari klausa `ORDER BY`:

```
1 SELECT column1, column2,...
2 FROM tbl
3 ORDER BY column1 [ASC|DESC], column2 [ASC|DESC],...
```

`ASC` merupakan singkatan *ascending* dan `DESC` merupakan singkatan *descending*. Secara *default*, klausa `ORDER BY` mengatur himpunan hasil dalam urutan menaik *(ascending order)* jika Anda tidak menentukan `ASC` atau `DESC` secara eksplisit.

<u>Contoh</u>

Data Tabel

customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

Sintaks

```
1 SELECT
2  contactLastname,
3  contactFirstname
4 FROM
5  customers
6 ORDER BY
7  contactLastname;
```

Hasil

	contactLastname	contactFirstname
▶	Accorti	Paolo
	Altagar,G M	Raanan
	Andersen	Mel
	Anton	Carmen
	Ashworth	Rachel
	Barajas	Miguel
	Benitez	Violeta
	Bennett	Helen
	Berglund	Christina

Sintaks lain

```
1 SELECT
2  contactLastname,
3  contactFirstname
4 FROM
5  customers
6 ORDER BY
7  contactLastname DESC;
```

Hasil

contactLastname	contactFirstname
Young	Jeff
Young	Julie
Young	Mary
Young	Dorothy
Yoshido	Juri
Walker	Brydey
Victorino	Wendy
Urs	Braun
Tseng	Jerry

4.2.2. Pemilihan Data *(Filtering Data)*

WHERE

Klausa `WHERE` memungkinkan Anda menentukan kondisi pencarian untuk mendapatkan baris yang dikembalikan dengan kueri. Berikut ini menunjukkan sintaks klausa `WHERE`.

Berikut ini menunjukkan sintaks dari klausa `WHERE`:

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

<u>Contoh</u>

Sintaks `WHERE`

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees
WHERE
    jobtitle = 'Sales Rep';
```

LIMIT

Klausa `LIMIT` digunakan pada *statement* `SELECT` untuk membatasi jumlah baris pada himpunan hasil. Klausa `LIMIT` menerima satu atau dua argumen. Nilai-nilai dari kedua argumen harus nol atau bilangan bulat positif.

Berikut ini mengilusttrasikan sintaks klausa `LIMIT` dengan dua argumen:

```
SELECT
    column1,column2,...
FROM
    table
LIMIT offset , count;
```

Mari kita periksa parameter klausa `LIMIT`:

- `offset` menentukan *offset* dari baris pertama yang akan dikembalikan. *Offset* dari baris pertama adalah 0, bukan 1.
- `count` menentukan jumlah maksimum baris yang akan dikembalikan.

Contoh

Sintaks `LIMIT`:

```
SELECT
 customernumber,
 customername,
 creditlimit
FROM
 customers
LIMIT 10;
```

AND

Operator `AND` adalah sebuah operator logika yang menggabungkan dua atau lebih ekspresi *Boolean* dan mengembalikan *true* hanya jika kedua ekspresi bernilai *true*. Operator `AND` mengembalikan *false* jika salah satu dari dua ekspresi tersebut bernilai *false*.

Berikut ini merupakan sintaks dari operator `AND`:

```
boolean_expression_1 AND boolean_expression_2
```

Contoh

Sintaks `AND`:

```
SELECT
    customername,
    country,
    state
FROM
    customers
WHERE
    country = 'USA' AND state = 'CA';
```

OR

Operator MySQL `OR` menggabungkan dua atau lebih ekspresi *Boolean* dan mengembalikan *true* ketika salah satu dari kedua kondisi bernilai *true.*

Berikut ini ilustrasi sintaks dari operator `OR`.

```
boolean_expression_1 OR boolean_expression_2
```

Contoh

Sintaks `OR`:

```
SELECT
 customername,
 country
FROM
 customers
WHERE country = 'USA'
 OR country = 'France';
```

IN

Operator *IN* memungkinkan Anda untuk menentukan apakah nilai yang ditentukan cocok dengan nilai apapun yang adalah dalam kumpulan nilai atau dikembalikan oleh *subquery*.

Berikut ini ilustrasi sintaks dari operator *IN*.

```
SELECT
    column1,column2,...
FROM
    table_name
WHERE
 (expr|column_1) IN ('value1','value2',...);
```

Contoh

Sintaks *IN*:

```
SELECT
    officeCode,
    city,
    phone,
    country
FROM
    offices
WHERE
    country IN ('USA' , 'France');
```

Sintaks *NOT IN*:

```
SELECT
    officeCode,
    city,
    phone
FROM
    offices
WHERE
    country NOT IN ('USA' , 'France');
```

BETWEEN

Operator `BETWEEN` adalah suatu operator logika yang memungkinkan Anda untuk menentukan apakah sebuah nilai berapa dalam jangkauan *(range)* tertentu atau tidak. Operator `BETWEEN` sering digunakan dalam klausa `WHERE` pada *statement SELECT*, *UPDATE*, dan *DELETE*.

Berikut ini ilustrasi sintaks dari operator `BETWEEN`.

```
expr [NOT] BETWEEN begin_expr AND end_expr;
```

Contoh

Sintaks `BETWEEN`:

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
WHERE
    buyPrice BETWEEN 90 AND 100;
```

Sintaks `NOT BETWEEN`:

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
WHERE
    buyPrice NOT BETWEEN 20 AND 100;
```

Sintaks `BETWEEN` dengan tanggal:

```
SELECT
   orderNumber,
   requiredDate,
   status
FROM
   orders
WHERE
   requireddate BETWEEN
     CAST('2003-01-01' AS DATE) AND
     CAST('2003-01-31' AS DATE);
```

LIKE

Operator `LIKE` merupakan operator logika yang mengevaluasi apakah suatu *string* memuat pola yang spesifik atau tidak. Berikut ini merupakan sintaks dari operator `LIKE`.

```
expression LIKE pattern ESCAPE escape_character
```

Operator `LIKE` digunakan dalam klausa `WHERE` pada *statement SELECT, UPDATE,* dan *DELETE* untuk memilih data berdasarkan pola.

MySQL menyediakan dua karakter *wildcard* untuk membangun pola: *percentage %* dan *underscore _*.

- *Wildcard percentage (%)* cocok dengan string apa pun yang berisi nol atau lebih karakter.
- *Wildcard underscore (_)* cocok dengan karakter tunggal apapun.

Misalnya, `s%` cocok dengan string apa pun yang dimulai dengan karakter `s`, seperti `sun` dan `six`. Pola `se_` cocok dengan string apapun yang dimulai dengan `se` dan diikuti dengan karakter apapun, seperti `see` dan `sea` .

<u>Contoh</u>

Sintaks `LIKE` pada awalan *(prefix)*:

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    firstName LIKE 'a%';
```

	employeeNumber	lastName	firstName
	1143	Bow	Anthony
	1611	Fixter	Andy

Sintaks `LIKE` pada akhiran (*afiks*):

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    lastName LIKE '%on';
```

employeeNumber	lastName	firstName
1056	Patterson	Mary
1088	Patterson	William
1166	Thompson	Leslie
1216	Patterson	Steve

Sintaks `LIKE` pada sisipan:

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    firstname LIKE 'T_m';
```

employeeNumber	lastName	firstName
1619	King	Tom

IS NULL

Untuk menguji apakah suatu nilai bernilai *NULL* atau tidak, Anda dapat menggunakan operator IS NULL. Berikut ini menunjukkan sintaks operator *`IS NULL`*.

```
value IS NULL
```

Apabila nilainya *NULL*, ekspresi mengembalikan *true*. Jika tidak, itu mengembalikan *false*.

<u>Contoh</u>

Sintaks *`IS NULL`*:

```
SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NULL
ORDER BY customerName;
```

customerName	country	salesrepemployeenumber
ANG Resellers	Spain	NULL
Anton Designs, Ltd.	Spain	NULL
Asian Shopping Network, Co	Singapore	NULL
Asian Treasures, Inc.	Ireland	NULL
BG&E Collectables	Switzerland	NULL
Cramer Spezialit?ten, Ltd	Germany	NULL
Der Hund Imports	Germany	NULL
Feuer Online Stores, Inc	Germany	NULL
Franken Gifts, Co	Germany	NULL

4.3. Aktivitas Mahasiswa

1. Buat *database* dengan data tabel *student* seperti berikut:

No	NIM	Name	City Address	Age	IPK	Departement
1	12345	Adi	Jakarta	17	2,5	Math
2	12346	Ani	Yogyakarta	20	2,1	Math
3	12347	Ari	Surabaya	18	2,5	Computer
4	12348	Ali	Banjarmasin	20	3,5	Computer
5	12349	Abi	Medan	17	3,7	Computer
6	12350	Budi	Jakarta	19	3,8	Computer
7	12351	Boni	Yogyakarta	20	3,2	Computer
8	12352	Bobi	Surabaya	17	2,7	Computer
9	12353	Beni	Banjarmasin	18	2,3	Computer
10	12354	Cepi	Jakarta	20	2,2	Null
11	12355	Coni	Yogyakarta	22	2,6	Null
12	12356	Ceki	Surabaya	21	2,5	Math
13	12357	Dodi	Jakarta	20	3,1	Math
14	12358	Didi	Yogyakarta	19	3,2	Physics
15	12359	Deri	Surabaya	19	3,3	Physics
16	12360	Eli	Jakarta	20	2,9	Physics
17	12361	Endah	Yogyakarta	18	2,8	Physics
18	12362	Feni	Jakarta	17	2,7	Null
19	12363	Farah	Yogyakarta	18	3,5	Null
20	12364	Fandi	Surabaya	19	3,4	Null

2. Tampilkan kolom *city address* tanpa ada duplikat.
3. Tampilkan data nama mahasiswa dari departemen komputer.
4. Tampilkan data NIM, nama, usia, dan *city address* yang diurutkan dari mahasiswa tertua.
5. Tampilkan nama dan usia 3 mahasiswa termuda dari Jakarta.
6. Tampilkan nama dan IPK mahasiswa asal Jakarta dengan IPK dibawah 2,5.
7. Tampilkan setiap nama mahasiswa dari Yogyakarta atau memiliki usia lebih tua dari 20.
8. Tampilkan nama dan alamat mahasiswa yang bukan dari Jakarta dan Surabaya.
9. Tampilkan nama, umur dan IPK mahasiswa dengan IPK dari 2,5 hingga 3,5.
10. Tampilkan nama mahasiswa yang memiliki huruf pada namanya.
11. Tampilkan NIM mahasiswa dengan data Null.

4.4. Pertanyaan

1. Mengapa kita perlu menggunakan *DML*?
2. Apa pentingnya *data sorting*?
3. Apa perbedaan antara *query* dan *filtering data*?

BAB V

DATA MANIPULATION LANGUAGE (DML) 3

5.1. Capaian Pembelajaran

1. Mahasiswa dapat mengenal *DML* pada *MySQL* seperti *MAX, MIN, AVG, SUM, COUNT, GROUP BY, HAVING*
2. Mahasiswa dapat *DML* pada *MySQL* seperti *MAX, MIN, AVG, SUM, COUNT, GROUP BY, HAVING*

5.2. Dasar Teori

5.2.1. Fungsi Agregat

MAX

Fungsi *MySQL* `MAX` mengembalikan nilai maksimum dalam sekumpulan nilai. Fungsi `MAX` berguna di dalam banyak kueri seperti untuk menemukan angka terbesar, produk paling mahal, dan pembayaran terbesar dari pelanggan.

Sintaks fungsi `MAX` adalah sebagai berikut:

```
MAX(DISTINCT expression)
```

Apabila Anda menambahkan operator `DISTINCT`, fungsi `MAX` mengembalikan nilai maksimum dari nilai-nilai yang berbeda, dimana sama dengan nilai maksimum dari semua nilai. Hal ini berarti operator `DISTINCT` tidak memberikan efek apapun pada fungsi `MAX`.

Contoh

```
SELECT
    MAX(amount)
FROM
    payments;
```

MIN

Fungsi `MIN` mengembalikan nilai minimum dalam sekumpulan nilai. Fungsi `MIN` sangat berguna dalam beberapa skenario seperti menemukan angka terkecil, memilih produk yang paling murah, mendapatkan *credit limit* terendah, dll.

Berikut ini ilustrasi sintaks fungsi `MIN`:

```
MIN(DISTINCT expression);
```

Apabila Anda menambahkan operator `DISTINCT`, fungsi `MIN` mengembalikan nilai minimum dari nilai-nilai yang berbeda, dimana sama dengan tanpa operator `DISTINCT`. Dengan kata lain, operator `DISTINCT` tidak memberikan efek apapun pada fungsi `MIN`. Hal itu hanya untuk kompatibilitas ISO.

Contoh

```
SELECT
    MIN(buyPrice)
FROM
    products;
```

AVG

Fungsi *MySQL* `AVG` merupakan suatu fungsi agregat yang memungkinkan Anda untuk menghitung itu nilai rata-rata dari sekumpulan nilai atau ekspresi.

Sintaks dari fungsi `AVG` adalah sebagai berikut:

```
AVG(DISTINCT expression)
```

Anda dapat menggunakan operator DISTINCT pada fungsi `AVG` untuk menghitung nilai rata-rata dari nilai-nilai berbeda. Sebagai contoh, jika Anda memiliki sekumpulan nilai-nilai 1,1,2,3, fungsi `AVG` dengan operator `DISTINCT` akan mengembalikan nilai dua, yaitu hasil dari (1 + 2 + 3) / 2 .

Contoh

```
SELECT
    AVG(buyprice) 'Average Price'
FROM
    products;
```

SUM

Fungsi `SUM` merupakan sebuah fungsi agregat yang memungkinkan Anda untuk menghitung jumlah dari sekumpulan nilai-nilai atau ekspresi.

Sintaks dari fungsi `SUM` adalah sebagai berikut:

```
SUM(DISTINCT expression)
```

Bagaimana fungsi `SUM` bekerja:

- Jika Anda menggunakan fungsi `SUM` dalam *statement* `SELECT` yang mengembalikan tidak ada baris yang cocok, fungsi `SUM` akan mengembalikan `NULL` , bukan nol.
- Operator `DISTINCT` memungkinkan Anda untuk menghitung nilai yang berbeda dari sekumpulan nilai.
- Fungsi `SUM` mengabaikan nilai `NULL` dalam perhitungan.

Contoh

```
SELECT
    FORMAT(SUM(quantityOrdered * priceEach),
        2) total
FROM
    orderdetails
WHERE
    orderNumber = 10100;
```

COUNT

Fungsi `COUNT` mengembalikan jumlah baris dalam suatu tabel. Fungsi `COUNT` memungkinkan Anda menghitung semua baris dalam tabel atau hanya baris yang cocok dengan kondisi tertentu.

Sintaks fungsi `COUNT` adalah sebagai berikut.

```
COUNT(expression)
```

Tipe kembalian dari fungsi `COUNT` adalah *`BIGINT`*. Fungsi `COUNT` akan mengembalikan nol jika tidak ada baris yang cocok ditemukan.

Terdapat beberapa bentuk fungsi `COUNT`: `COUNT(*)`, `COUNT(expression)`, dan `COUNT (DISTINCT expression)`.

Fungsi `COUNT(*)` mengembalikan jumlah baris dalam himpunan hasil yang dikembalikan dari *statement* `SELECT`. Fungsi `COUNT(*)` menghitung baris yang berisi nilai non-NULL dan NULL.

Jika Anda menggunakan fungsi `COUNT(*)` untuk menghitung baris dalam suatu tabel tanpa menggunakan klausa `WHERE` dan memilik kolom lain, hal itu akan bekerja sangat cepat.

Fungsi `COUNT(expression)` mengembalikan jumlah baris yang tidak memuat nilai NULL.

Fungsi `COUNT (DISTINCT expression)` mengembalikan jumlah baris unuk yang tidak memuat nilai NULL.

Contoh

id	val
1	1
2	1
3	2
4	2
5	NULL
6	3
7	4
8	NULL
9	5

```
SELECT COUNT(*)
FROM demos;
```

COUNT(*)
9

```
SELECT COUNT(*)
FROM demos
WHERE val = 2;
```

COUNT(*)
2

```
SELECT COUNT(DISTINCT val)
FROM demos;
```

	COUNT(DISTINCT val)
▶	5

5.2.2. Pengelompokan Data *(Grouping Data)*

GROUP BY

Klausa `GROUP BY` mengelompokkan sekumpulan baris ke satu kumpulan rangkuman baris berdasarkan nilai-nilai dari kolom atau ekspresi. Klausa `GROUP BY` mengembalikan satu baris untuk setiap kelompok. Dengan kata lain, klausa ini mengurangi jumlah baris dalam himpunan hasil.

Anda sering menggunakan klausa `GROUP BY` dengan fungsi agregat seperti sebagai *SUM, AVG, MAX, MIN*, dan *COUNT*. Fungsi agregat yang muncul dalam klausa `SELECT` memberikan informasi tentang setiap grup.

Klausa `GROUP BY` adalah klausa opsional dari *statement* `SELECT`. Berikut ini ilustrasi sintaks klausa `GROUP BY`:

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```

Klausa `GROUP BY` harus muncul setelah klausa `FROM` atau `WHERE`. *Keyword* `GROUP BY` diikuti dengan dafrar kolom yang dipisahkan dengan koma atau ekspresi yang Anda ingin gunakan sebagai kriteria untuk mengelompokkan baris

<u>Contoh</u>

```
SELECT
    status
FROM
    orders
GROUP BY status;
```

	status
▶	Cancelled
	Disputed
	In Process
	On Hold
	Resolved
	Shipped

```sql
SELECT
    status, COUNT(*)
FROM
    orders
GROUP BY status;
```

status	COUNT(*)
Cancelled	6
Disputed	3
In Process	6
On Hold	4
Resolved	4
Shipped	303

HAVING

Klausa `HAVING` digunakan dalam *statement* `SELECT` untuk menentukan kondisi *filter* untuk sekelompok baris atau agregat.

Klausa `HAVING` sering digunakan dengan klausa `GROUP BY` untuk menyaring grup berdasarkan suatu kondisi spesifik tertentu. Apabila klausa `GROUP BY` dihilangkan, maka klausa `HAVING` berperilaku seperti klausa `WHERE`.

Perhatikan bahwa klausa `HAVING` menerapkan kondisi filter untuk setiap grup baris, sedangkan klausa `WHERE` menerapkan kondisi *filter* untuk setiap baris individu.

Contoh

```sql
SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY ordernumber
HAVING total > 1000;
```

ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62

5.3. Aktivitas Mahasiswa

1. Buat *database* dengan data tabel *student* seperti berikut:

No	NIM	Name	City Address	Age	IPK	Departement
1	12345	Adi	Jakarta	17	2,5	Math
2	12346	Ani	Yogyakarta	20	2,1	Math
3	12347	Ari	Surabaya	18	2,5	Computer
4	12348	Ali	Banjarmasin	20	3,5	Computer
5	12349	Abi	Medan	17	3,7	Computer
6	12350	Budi	Jakarta	19	3,8	Computer
7	12351	Boni	Yogyakarta	20	3,2	Computer
8	12352	Bobi	Surabaya	17	2,7	Computer
9	12353	Beni	Banjarmasin	18	2,3	Computer
10	12354	Cepi	Jakarta	20	2,2	Null
11	12355	Coni	Yogyakarta	22	2,6	Null
12	12356	Ceki	Surabaya	21	2,5	Math
13	12357	Dodi	Jakarta	20	3,1	Math
14	12358	Didi	Yogyakarta	19	3,2	Physics
15	12359	Deri	Surabaya	19	3,3	Physics
16	12360	Eli	Jakarta	20	2,9	Physics
17	12361	Endah	Yogyakarta	18	2,8	Physics
18	12362	Feni	Jakarta	17	2,7	Null
19	12363	Farah	Yogyakarta	18	3,5	Null
20	12364	Fandi	Surabaya	19	3,4	Null

2. Tampilkan kolom *city address* tanpa ada duplikat.
3. Tampilkan nilai IPK Maksimum IPK dari mahasiswa dari Jakarta.
4. Tampilkan nilai IPK Minimum dari mahasiswa departemen Komputer.
5. Tampilkan nilai IPK Rata-rata dari mahasiswa departemen Matematika.
6. Tampilkan total pembayaran mahasiswa dari departemen Fisika
7. Tampilkan jumlah mahasiswa dari departemen Komputer.
8. Tampilkan jumlah mahasiswa dari setiap departemen.
9. Tampilkan jumlah mahasiswa yang dikelompokkan berdasarkan *city address* dengan usia dibawah 20 tahun.

5.4. Pertanyaan

1. Mengapa kita membutuhkan *agregate DML*?
2. Apa pentingnya pengelompokan data?
3. Apa perbedaan antara `WHERE` dan `HAVING`?

BAB VI
SELECT JOIN

6.1. Capaian Pembelajaran

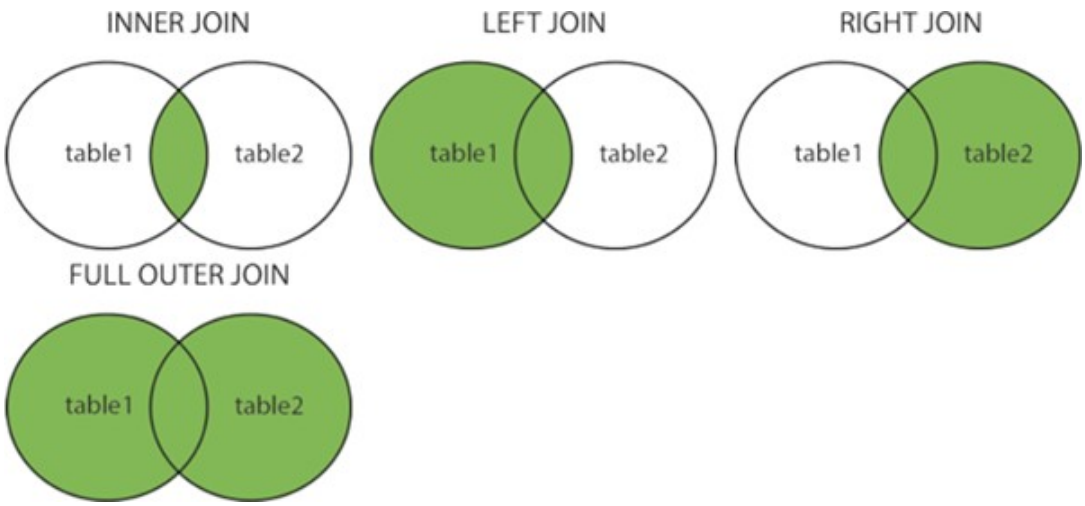
1. Mahasiswa dapat memilih *record* dari beberapa tabel.
2. Mahasiswa dapat mengimplementasikan berbagai *keyword Join* untuk memilih *record.*

6.2. Dasar Teori

Primary key dan *foreign key* dimaksudkan untuk menghubungkan *record* antar tabel. Data antar tabel perlu dilihat seluruhnya untuk melengkapi informasi. Praktisnya, kita perlu memilih *record* dari banyak tabel.

Terdapat berbagai tipe *Join* untuk memilih *record* dari dua tabel:

1. *(Inner) Join*: mencocokkan pada kedua tabel.
2. *Left (Outer) Join*: semua *record* dari tabel kiri dan *record* yang cocok dari tabel kanan.
3. *Right (Outer) Join*: semua *record* dari tabel kanan dan *record* yang cocok dari tabel meja.
4. *Full (Outer) Join:* mencocokkan nilai dari tabel kiri atau tabel kanan.
5. *Self Join*: *Inner join* dari suatu tabel dengan tabel itu sendiri.
6. *Union*: *record* merupakan gabungan dari dua atau lebih *statement select.*



Gunakan tabel berikut ini sebagai contoh:

jobID	jobTitle
1	Producer
2	Director
3	Writer
4	Actor
5	Fans
6	Executive Producer
7	Stunt Organizer
8	VFX Artist
9	Stuntman

employeeID	lastName	firstName	email	reportsTo	jobID
1	Carter	John	carterj@mail.co		1
2	Travolta	John	travj@mail.co		2
3	Carter	Peggy	carterp@mail.co		3
4	Connor	John	connj@mail.co		2
5	Johnson	Rian	jrian@mail.co		4
6	Rogers	Steve	cap@mail.co	3	2
7	Evans	Chris	evanc@mail.co	3	1
8	Pratt	Chris	prc@mail.co	1	5
9	Hemsworth	Chris	lebowski@mail.co	2	3
10	Hemsworth	Liam	hliam@mail.co		4
11	Feige	Kevin	kfeige@mail.co		6
12	Miller	George	travj@mail.co		

6.2.1. *Inner Join*

Inner join dari 2 *database* akan mengembalikan *record* yang memiliki nilai yang cocok di kedua tabel.

```
-- INNER JOIN
SELECT column1, column2
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Contoh

```
-- INNER JOIN
SELECT em.firstName, em.lastName, jo.jobTitle
FROM employees as em
INNER JOIN jobs as jo
ON em.jobID = jo.jobID;
```

Hasil:

firstName	lastName	jobTitle
John	Carter	Producer
John	Travolta	Director
Peggy	Carter	Writer
John	Connor	Director
Rian	Johnson	Actor
Steve	Rogers	Director
Chris	Evans	Producer
Chris	Pratt	Fans
Chris	Hemsworth	Writer
Liam	Hemsworth	Actor
Kevin	Feige	Executi...

6.2.2. *Left (Outer) Join*

Left join dari 2 *database* akan mengembalikan semua *record* dari tabel kiri dan *record* yang cocok dari tabel kanan.

```
1   -- LEFT JOIN
2   SELECT column1, column2
3   FROM table1
4   LEFT JOIN table2
5   ON table1.column_name = table2.column_name;
```

Contoh

```
1   -- LEFT JOIN
2 • SELECT em.firstName, em.lastName, jo.jobTitle
3   FROM employees as em
4   LEFT JOIN jobs as jo
5   ON em.jobID = jo.jobID;
```

Hasil:

firstName	lastName	jobTitle
John	Carter	Producer
John	Travolta	Director
Peggy	Carter	Writer
John	Connor	Director
Rian	Johnson	Actor
Steve	Rogers	Director
Chris	Evans	Producer
Chris	Pratt	Fans
Chris	Hemsworth	Writer
Liam	Hemsworth	Actor
Kevin	Feige	Executi...
George	Miller	NULL

6.2.3. *Right (Outer) Join*

Right join dari 2 *database* akan mengembalikan semua *record* dari tabel kanan dan *record* yang cocok dari tabel kiri.

```
-- RIGHT JOIN
SELECT column1, column2,...
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Contoh

```
1     -- RIGHT JOIN
2 •   SELECT em.firstName, em.lastName, jo.jobTitle
3     FROM employees as em
4     RIGHT JOIN jobs as jo
5     ON em.jobID = jo.jobID;
```

Hasil:

firstName	lastName	jobTitle
John	Carter	Producer
Chris	Evans	Producer
John	Travolta	Director
John	Connor	Director
Steve	Rogers	Director
Peggy	Carter	Writer
Chris	Hemsworth	Writer
Rian	Johnson	Actor
Liam	Hemsworth	Actor
Chris	Pratt	Fans
Kevin	Feige	Executi...
NULL	NULL	Stunt O...
NULL	NULL	VFX Artist
NULL	NULL	Stuntman

6.2.4. *Self Join*

Self join akan mengembalikan semua *record* dari tabel yang digabungkan dengan dirinya sendiri.

```
1     -- SELF JOIN
2 •   SELECT column1, column2
3     FROM table1 as T1, table2 as T2
4     WHERE conditions;
```

Contoh

```
1 •  SELECT
2        em1.employeeID, em1.firstName, em1.lastName,
3        em1.reportsTo, em2.firstName, em2.lastName
4    FROM
5        employees as em1, employees as em2
6    WHERE
7        em1.reportsTo = em2.employeeID;
```

Hasil:

	employeeID	firstName	lastName	reportsTo	firstName	lastName
▶	6	Steve	Rogers	3	Peggy	Carter
	7	Chris	Evans	3	Peggy	Carter
	8	Chris	Pratt	1	John	Carter
	9	Chris	Hemsworth	2	John	Travolta

6.2.5. *Union*

Statement Union akan mengembalikan *record* gabungan dari dua atau lebih *statement select*.

```
1     -- UNION
2 •  SELECT column1, column2 FROM table1
3    UNION
4    SELECT column1, column2 FROM table2;
```

Contoh 1

```
1     -- UNION
2 •  SELECT jobID
3    FROM employees
4    WHERE jobID IS NOT NULL
5    UNION
6    SELECT jobID FROM jobs;
```

Hasil:

	jobID
▶	1
	2
	3
	4
	5
	6
	7
	8
	9

Contoh 2

```
1    -- UNION ALL
2 •  SELECT jobID
3    FROM employees
4    WHERE jobID IS NOT NULL AND jobID < 6
5    UNION ALL
6    SELECT jobID FROM jobs;
```

Hasil:

jobID
1
1
2
2
2
3
3
4
4
5
1
2
3
4
5
6
7
8
9

6.3. Aktivitas Mahasiswa

Buatlah tabel untuk sistem perpustakaan yang terdiri dari tiga tabel yaitu *books*, *user*, dan *flow* dengan kolom sebagai berikut:

1. *Books*: *bookID, bookTitle, authorName, borrowedStatus*
2. *Flow*: *flowID, userIDBorrowing ,bookIDBorrowed, borrowDate , returnDate*
3. *User*: *userID, userName, numberOfBorrowing, numberOfReturning*

Dan gunakan *statement join* untuk

1. Menampilkan semua judul buku dimana statusnya dipinjam *(borrowed)* dan tanggal peeminjammnya kemarin.
2. Menampilkan semua judul buku bahkan buku yang tidak dipinjam dan *userID* peminjamnya untuk buku yang dipinjam.
3. Menampilkan semua buku yang dipinjam dan semua *userID* apakah dia meminjam atau tidak.
4. Dengan menggunakan satu kueri, tampilkan daftar semua judul buku dan nama *user* yang meminjam buku tersebut dan *user* yang meminjam lebih dari 3 buku.

BAB VII
SUBQUERY

7.1. Capaian Pembelajaran

1. Mahasiswa mampu menulis *query* yang kompleks.
2. Mampu mengimplementasikan *sub-query* pada *insert, update, update,* dan *delete.*

7.2. Dasar Teori

Subquery adalah *query* di dalam *query*, atau bisa juga disebut nested query. Secara teknis, *subquery* adalah *statement SELECT* di dalam *statement* lain seperti:

1. *SELECT*
2. *INSERT*
3. *UPDATE*
4. *DELETE*

Implementasi dari *subquery*

1. *WHERE*
2. *IN* dan *NOT IN*
3. *FROM*
4. *EXIST* dan *NOT EXIST*

Gunakan tabel berikut ini sebagai contoh:

jobID	jobTitle	basicSalary
1	Producer	20.000.000
2	Director	15.000.000
3	Writer	10.000.000
4	Actor	9.000.000
5	Fans	-
6	Executive Producer	22.500.000
7	Stunt Organizer	7.500.000
8	VFX Artist	7.500.000
9	Stuntman	6.000.000

employeeID	lastName	firstName	email	reportsTo	jobID	salary
1	Carter	John	carterj@mail.co		1	21.500.000
2	Travolta	John	travj@mail.co		2	19.000.000
3	Carter	Peggy	carterp@mail.co		3	10.000.000
4	Connor	John	connj@mail.co		2	16.000.000
5	Johnson	Rian	jrian@mail.co		4	15.000.000
6	Rogers	Steve	cap@mail.co	3	2	17.500.000
7	Evans	Chris	evanc@mail.co	3	1	25.000.000
8	Pratt	Chris	prc@mail.co	1	5	100.000
9	Hemsworth	Chris	lebowski@mail.co	2	3	12.000.000
10	Hemsworth	Liam	hliam@mail.co		4	9.500.000
11	Feige	Kevin	kfeige@mail.co		6	27.000.000
12	Miller	George	travj@mail.co			-

7.2.1. Perbandingan menggunakan *Subquery*

Perbandingan biasanya terjadi di dalam klausa *WHERE* untuk membandingkan dua nilai. *Subquery* dapat digunakan untuk menghasilkan nilai menggunakan *statement SELECT* untuk dibandingkan dengan suatu.

```
non_subquery_operand comparison_operator (subquery)
```

Operator perbandingan adalah salah satu dari operator berikut:

```
=  >  <  >=  <=  <>  !=  <=>
```

Contoh 1

```
1   -- COMPARISON
2   SELECT lastName, firstName, salary
3   FROM employees
4   WHERE salary >
5       (SELECT MAX(basicSalary)
6       FROM jobs);
```

Hasil

lastName	firstName	salary
Evans	Chris	25000000
Feige	Kevin	27000000

Contoh 2

```
1   -- COMPARISON
2   SELECT *
3   FROM employees as em
4   WHERE 3 =
5       (SELECT count(*)
6       FROM employees
7       WHERE em.jobId = employees.jobID)
```

Hasil

employeeID	lastName	firstName	email	reportsTo	jobID	Salary
2	Travolta	John	travj@mail.co	NULL	2	19000000
4	Connor	John	connj@mail.co	NULL	2	16000000
6	Rogers	Steve	cap@mail.co	3	2	17500000

7.2.2. *Subquery* menggunakan *ANY, IN, SOME,* dan *ALL*

Subquery juga dapat digunakan sebagai sebuah *list* dimana suatu operan memeriksa kemunculannya.

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
operand comparison_operator ALL (subquery)
```

Contoh 1

```
1    -- ANY, IN, SOME
2 •  SELECT jobTitle, basicSalary FROM jobs
3    WHERE jobID = ANY
4        (SELECT jobID FROM employees
5        WHERE salary>10000000);
```

Hasil

jobTitle	basicSalary
Producer	20000000
Director	15000000
Actor	9000000
Writer	10000000
Executive Producer	22500000

Contoh 2

```
1    -- ANY, IN, SOME
2 •  SELECT jobTitle, basicSalary FROM jobs
3    WHERE jobID NOT IN
4        (SELECT jobID FROM employees
5        WHERE salary>10000000);
```

Hasil

jobTitle	basicSalary
Fans	0
Stunt Organizer	7500000
VFX Artist	7500000
Stuntman	6000000

Contoh 3

```
1    -- ANY, IN, SOME
2 •  SELECT jobTitle, basicSalary FROM jobs
3    WHERE jobID <> ANY
4        (SELECT jobID FROM employees
5        WHERE salary>10000000);
```

Hasil

jobTitle	basicSalary
Producer	20000000
Director	15000000
Writer	10000000
Actor	9000000
Fans	0
Executive Producer	22500000
Stunt Organizer	7500000
VFX Artist	7500000
Stuntman	6000000

Contoh 4

```
-- ANY, IN, SOME
SELECT * FROM jobs
WHERE jobID > ALL
    (SELECT jobID FROM employees
    WHERE salary>10000000);
```

Hasil

jobID	jobTitle	basicSalary
7	Stunt Organizer	7500000
8	VFX Artist	7500000
9	Stuntman	6000000

7.2.3. *Subquery* menggunakan *EXISTS*

Selft join akan mengembalikan semua *records* dari tabel yang digabungkan dengan dirinya sendiri.

```
-- EXISTS
SELECT column1,column2
FROM table1
WHERE EXISTS
    (SELECT column1,column2
    FROM table2
    WHERE conditions);
```

Contoh 1

```
-- EXISTS
SELECT firstName,lastName,jobID,salary
FROM employees
WHERE EXISTS
    (SELECT jobID
    FROM jobs
    WHERE employees.jobID=jobs.jobID AND employees.salary < 1.1*jobs.basicSalary);
```

Hasil

firstName	lastName	jobID	salary
John	Carter	1	21500000
Peggy	Carter	3	10000000
John	Connor	2	16000000
Liam	Hemsworth	4	9500000

Contoh 2

```
1    -- EXISTS
2    SELECT firstName,lastName,jobID
3    FROM employees
4    WHERE NOT EXISTS
5        (SELECT jobID
6        FROM jobs
7        WHERE employees.jobID=jobs.jobID);
```

Hasil

firstName	lastName	jobID
George	Miller	NULL

7.2.4. *Subquery* menggunakan klausa *FROM*

Klausa *FROM* biasanya merujuk ke suatu nama tabel. Seperti yang kita pahami bahwa suatu *query* menggunakan *SELECT* dapat menghasilkan suatu tabel. Kita dapat menggunakan hasil dari *subquery SELECT* sebagai suatu tabel pada klausa *FROM.*

Contoh 1

```
1    -- Subquery in FROM
2    SELECT basic,recent,title
3    FROM
4        (SELECT jo.basicSalary as basic,em.salary as recent, jo.jobTitle as title
5        FROM jobs as jo
6        INNER JOIN employees as em
7        WHERE jo.jobID = em.jobID
8        ) as salaryJob
9    WHERE recent>10000000;
```

Hasil

basic	recent	title
20000000	21500000	Producer
15000000	19000000	Director
15000000	16000000	Director
9000000	15000000	Actor
15000000	17500000	Director
20000000	25000000	Producer
10000000	12000000	Writer
22500000	27000000	Executive Producer

7.2.5. *Subquery* pada *INSERT* dan *UPDATE*

Hasil *subquery* dapat digunakan sebagai parameter pada klausa *INSERT* dan *UPDATE*.

```
1   -- Subquery in INSERT
2   INSERT INTO table_name(column1,column2)
3   VALUES
4       (subquery);
```

Contoh 1

```
1   -- Subquery in INSERT
2   INSERT INTO jobs(jobTitle,basicSalary)
3   VALUES(
4       'Vice President',
5       (SELECT 1.25*max(salary) FROM employees)
6       );
```

Hasil

jobID	jobTitle	basicSalary
1	Producer	20000000
2	Director	15000000
3	Writer	10000000
4	Actor	9000000
5	Fans	0
6	Executive Producer	22500000
7	Stunt Organizer	7500000
8	VFX Artist	7500000
9	Stuntman	6000000
10	Vice President	33750000

Contoh 2

```
1   -- Subquery in UPDATE
2   UPDATE jobs
3   SET basicSalary = (
4       SELECT 0.1*MIN(salary)
5       FROM employees
6       )
7   WHERE basicSalary = 0 AND
8       jobID IN (SELECT DISTINCT jobID FROM employees);
9   SELECT * FROM jobs;
```

Hasil

jobID	jobTitle	basicSalary
1	Producer	20000000
2	Director	15000000
3	Writer	10000000
4	Actor	9000000
5	Fans	10000
6	Executive Producer	22500000
7	Stunt Organizer	7500000
8	VFX Artist	7500000
9	Stuntman	6000000
10	Vice President	33750000

7.3. Aktivitas Mahasiswa

1. Buat tabel sistem perpustakaan yang terdiri dari tiga tabel yaitu *book*, *user*, dan *flow* dengan kolom :
   a. *Book: bookID, bookTitle, authorName, borrowedStatus*
   b. *Flow: flowID, userIDBorrowing, bookIDBorrowed, borrowDate, returnDate*
   c. *User: userID, userName, numberOfBorrowing, numberOfReturning*
2. Gunakan *subquery* untuk
   a. Menampilkan semua judul buku yang mempunyai status dipinjam dan tanggal peminjamannya kemarin
   b. Menampilkan semua judul buku baik buku yang tidak dipinjam dan *userID* peminjam untuk buku yang dipinjam.
   c. Menampilkan semua buku yang dipinjam dan semua *userID* baik dia sebagai peminjaman atau tidak.
   d. Menggunakan satu *query*, tampilkan daftar semua judul buku dan *userName* yang meminjam buku dan *user* yang meminjam lebih dari 3 buku.

BAB VIII
STORED PROCEDURE

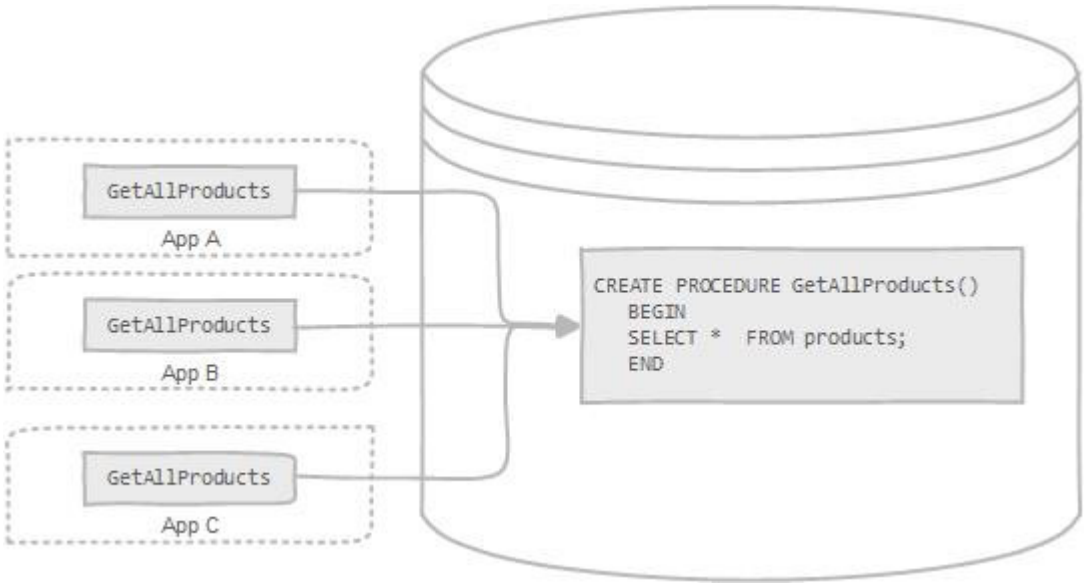
8.1. Capaian Pembelajaran

1. Mahasiswa mampu membuat prosedur sederhana menggunakan MySQL
2. Mahasiswa mampu membuat prosedur dengan variabel menggunakan MySQL
3. Mahasiswa mampu membuat prosedur dengan parameter menggunakan MySQL
4. Mahasiswa mampu mengimplementasikan *statement IF* dan *CASE* menggunakan MySQL

8.2. Dasar Teori

8.2.1. Definisi Prosedur Tersimpan

Prosedur tersimpan *(stored procesure)* adalah suatu segmen *statement* SQL deklaratif yang disimpan di dalam katalog *database*.



Keuntungan Prosedur Tersimpan MySQL

- Biasanya, prosedur tersimpan membantu meningkatkan kinerja aplikasi. Setelah dibuat, prosedur tersimpan dijalankan dan disimpan dalam *database*. Namun, MySQL mengimplementasikan prosedur tersimpan yang sedikit berbeda. Prosedur tersimpan MySQL dikompilasi sesuai permintaan *(request)*. Setelah mengkompilasi prosedur tersimpan, MySQL memasukkannya ke dalam *cache* dan mengatur *cache* prosedur tersimpannya sendiri untuk setiap koneksi. Jika suatu aplikasi menggunakan prosedur tersimpan beberapa kali dalam satu koneksi, maka *compiled version* akan digunakan, jika tidak, prosedur tersimpan berfungsi seperti *query*.
- Prosedur tersimpan membantu mengurangi *traffic* antara aplikasi dan *server* karena aplikasi

hanya mengirim nama dan parameter dari prosedur tersimpan dan tidak perlu mengirimkan *statement SQL* yang panjang.

- Prosedur tersimpan dapat digunakan kembali dan transparan untuk aplikasi apa pun. Prosedur tersimpan memperlihatkan *database interface* ke semua aplikasi sehingga *developer* tidak harus mengembangkan fungsi yang sudah didukung dalam prosedur tersimpan.
- Prosedur tersimpan aman. Administrator *database* dapat memberikan izin yang sesuai untuk aplikasi yang mengakses prosedur tersimpan dalam *database* tanpa memberikan izin apa pun pada tabel *database* yang menjadi sumbernya.

Di samping keuntungan yang sudah dijelaskan sebelumnya, prosedur tersimpan juga memiliki kekurangannya sendiri dimana kita harus memahami kekurangan tersebut sebelum kita menggunakannya di *database* kita.

Kerugian Prosedur Tersimpan MySQL

- Penggunaan memori setiap koneksi akan meningkat secara substansial apabila menggunakan banyak prosedur tersimpan. Selain itu, apabila terlalu sering menggunakan operasi logika di dalam prosedur tersimpan, penggunaan CPU akan meningkat karena *server database* tidak dirancang dengan baik untuk mengembangakan operasi logika yang kompleks dan fleksibel.
- Susunan prosedur tersimpan tidak dirancang untuk mengembangkan logika bisnis yang kompleks dan fleksibel.
- Sulit men-*debug* prosedur tersimpan. Hanya beberapa sistem manajemen *database* yang memungkinkan kita untuk men-*debug* prosedur tersimpan. Sayangnya, MySQL tidak menyediakan fasilitas untuk *debugging* prosedur tersimpan.
- Tidak mudah mengembangkan dan mengatur prosedur disimpan Prosedur. Untuk mengembangkan dan memelihara prosedur tersimpan seringkali membutuhkan keahlian khusus yang tidak dimiliki oleh semua pengembang aplikasi. Hal ini dapat menyebabkan masalah dalam fase pengembangan dan pemeliharaan aplikasi.

Prosedur tersimpan MySQL memiliki kelebihan dan kekurangannya masing-masing. Ketika mengembangkan suatu aplikasi, *developer* harus memutuskan apakah akan menggunakan prosedur tersimpan atau tidak berdasarkan arsitektur aplikasi yang akan dibuat.

8.2.2. Menulis itu pertama MySQL disimpan prosedur

Kita akan membuat prosedur tersimpan sederhana yang bernama `GetAllProducst()`. Prosedur tersimpan `GetAllProduct()` digunakan untuk memilih *(select)* semua produk yang ada pada tabel produk.

Gunakan *mysql client tool* dan ketikkan perintah berikut ini:

```
DELIMITER //
CREATE PROCEDURE GetAllProducts()
  BEGIN
  SELECT *  FROM products;
  END //
DELIMITER ;
```

Mari kita pelajari prosedur tersimpan tersebut secara lebih rinci:

- *Command* pertama adalah `DELIMITER //` *(double slash)* yang tidak terkait dengan sintaks prosedur tersimpan. *Statement* `DELIMITER` ini hanya digunakan untuk mengubah *delimiter* standar, yaitu titik koma (;) menjadi *delimiter* lain. Pada kasus ini, *delimiter* dirubah dari titik koma *(semicolon)* (;) menjadi garis miring ganda *(double-slashes)* (//),

  Mengapa kita harus mengubah *delimiter? Delimiter* diubah karena kita hendak mengirimkan prosedur tersimpan ke *server* secara keseluruhan sebagai satu kesatuan utuh daripada membiarkan *mysql tool* menjalankan setiap *statement* setiap waktu (*command* dijalankan baris per baris). Setelah *keyword* `END`, kita menggunakan *delimiter* untuk menunjukkan akhir dari prosedut tersimpan. Perintah terakhir (`DELIMITER;`) mengubah delimiter kembali menjadi titik koma (;).
- Kita menggunakan *statement* `CREATE PROCEDURE` untuk membuat suatu prosedur tersimpan baru. Kita menetapkan nama prosedur tersimpan setelah *statement* `CREATE PROCEDURE`. Pada kasus ini, nama prosedur tersimpan adalah `GetAllProducts`. Kita letakkan tanda kurung (*parentheses)* (()) setelah nama prosedur tersimpan
- Bagian antara `BEGIN` dan END disebut *body* prosedur tersimpan. *Statement SQL* ditempatkan antara *body* prosedur tersimpan untuk menganani *business logic*. Pada prosedur tersimpan ini, kita menggunakan statement *SELECT* yang sederhana untuk mendapatkan data dari tabel `products`.

8.2.3. Variabel Prosedur Tersimpan MySQL

Variabel adalah objek data bernama yang nilainya dapat berubah selama prosedur tersimpan dieksekusi. Variabel biasanya digunakan dalam prosedur tersimpan untuk menyimpan hasil langsung. Variabel-variabel ini bersifat lokal untuk prosedur tersimpan. Anda harus mendeklarasikan variabel sebelum menggunakannya.

Mendeklarasikan Variabel

Untuk mendeklarasikan suatu variabel pada prosedur tersimpan, kita gunakan *statement* *`DECLARE`* seperti dibawah ini:

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

Mari kita pelajari *satatement* diatas secara lebih rinci:

- Pertama, tentukan nama variabel setelah keyword `DECLARE`. Nama variabel harus mengikuti aturan penamaan nama kolom pada tabel MySQL.
- Kemudian tentukan tipe data dari variabel tersebut dan ukurannya. Suatu variabel dapat memiliki tipe data MySQL apa pun seperti `INT`, `VARCHAR`, dan `DATETIME`.
- Ketiga, ketika mendeklarasikan suatu variabel, nilai awalnya adalah `NULL`. Untuk meng-*assign* suatu nilai *default* pada variabel, gunakan *keyword* `DEFAULT` kemudian diikuti degan nilai default nya.

Sebagai contoh, misalnya kita akan mendeklarasikan suatu variable `total_sale` dengan tipe data `INT` dan nilai default 0, maka kita dapat menggunakan *statement* seperti berikut ini:

```
DECLARE total_sale INT DEFAULT 0;
```

MySQL juga dapat mendeklarasikan 2 variabel atau lebih yang mempunyai tipe data yang sama menggunakan satu *statement* `DECLARE` seperti berikut ini.

```
DECLARE total_sale INT DEFAULT 0;
```

Pada contoh ini, kita mendeklrasikan 2 variabel bertipe *integer* yaitu x dan y, dimana kita tetapan nilai default nya menjadi 0.

Menetapkan Nilai Variabel

Ketika kita sudah mendeklrasikan suatu variabel, ktia dapat mulai menggunakannya. Untuk menetapkan nilai lain pada suatu variabel, kita dapat menggunakan statement `SET`, seperti dibawah ini:

```
DECLARE total_count INT DEFAULT 0;SET total_count = 10;
```

Pada contoh diatas, nilai dari variabel total_count adalah 10 setelah *assignment.*

Selain *statement* `SET`, bisa juga menggunakan *statement* `SELECT INTO` untuk menetapkan nilai pada suatu variabel berdasarkan hasil kueri yang mengembalikan nilai *scalar*. Contohnya seperti berikut ini:

```
DECLARE total_products INT DEFAULT 0; SELECT COUNT(*) INTO total_products FROM
   products;
```

Pada contoh di atas:

- Pertama, deklarasikan variabel bernama `total_products` dan inisialisasi nilainya dengan nilai default 0.

- Kemudian, menggunakan *statement* `SELECT INTO` isikan nilai variabel `total_products` menjadi jumlah produk yang dipilih dari tabel `products` dalam *database* sampel.

Cakupan Variabel

Suatu variabel memiliki cakupannya *(scope)* sendiri yang menentukan masa pemakaiannya. Apabila mendeklarasikan suatu variabel di dalam prosedur tersimpan, maka batas *scope* adalah ketika telah mencapai *statement* `END`.

Apabila mendeklarasikan suatu variabel di dalam blok `BEGIN END`, maka batas *scope* adalah ketika *statement* `END` telah tercapai. Deklarasi dua variabel atau lebih dengan nama yang sama dapat dilakukan dengan *scope* yang berbeda karena variabel hanya efektif dalam *scope*-nya sendiri. Namun hal ini sebaiknya tidak dilakukan.

Suatu variabel yang namanya diawali dengan tanda `@` adalah *session variable.* Variabel ini tersedia dan dapati diakses hingga sesi berakhir.

Pada tutorial ini, kita telah menunjukkan kepada Anda cara mendeklarasikan variabel di dalam prosedur tersimpan dan membahas cakupan variabel.

8.2.4. Parameter Prosedur Tersimpan MySQL

Hampir semua prosedur tersimpan yang dikembangkan membutuhkan parameter. Parameter membuat prosedur tersimpan lebih fleksibel dan berguna. Pada MySQL, suatu parameter memiliki satu diantara tiga model: `IN, OUT` atau `INOUT`.

- ***IN*** – adalah mode *default.* Saat mendefinisikan parameter `IN` dalam prosedur tersimpan, *calling program* harus memberikan argumen ke prosedur tersimpan. Sebagai tambahan, nilai pada parameter `IN` dilindungi (*protected*). Hal ini berarti bahwa apabila nilai parameter `IN` diubah di dalam prosedur tersimpan, nilai aslinya tetap dipertahankan setelah prosedur tersimpan berakhir. Dengan kata lain, prosedur tersimpan hanya berjalan menggunakan salinan parameter `IN`.
- ***OUT*** – nilai pada parameter `OUT` dapat diubah di dalam prosedur tersimpan dan nilai baru dapat dikembalikan ke *calling program.* Perlu diperhatikan bahwa prosedur tersimpan tidak dapat mengakses nilai awal dari parameter `OUT` ketika di awal.
- ***INOUT*** merupakan kombinasi dari parameter `IN` dan `OUT`. Hal ini berarti bahwa *calling program* dapat memberikan argumen dan prosedur tersimpan dapat mengubah parameter `INOUT` dan memberikan nilai baru kembali ke *calling program*

Sintaks mendefinisikan parameter dalam prosedur tersimpan adalah sebagai berikut:

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

- ***MODE*** dapat berupa *IN*, *OUT* atau *INOUT* tergantung pada tujuan parameter dalam prosedur tersimpan.
- ***param_name*** adalah nama parameter. Nama parameter harus mengikuti aturan penamaan nama kolom di MySQL.
- Nama parameter diikuti dengan tipe data dan ukurannya. Seperti variabel, tipe data suatu parameter dapat berupa semua tipe data MySQL yang valid.

Setiap parameter dipisahkan dengan koma (,) jika prosedur tersimpan memiliki lebih dari satu parameter.

Mari berlatih dengan beberapa contoh untuk mendapatkan pemahaman yang lebih baik. Kami akan menggunakan tabel sample database untuk demonstrasi.

Contoh Parameter Prosedur Tersimpan MySQL

Contoh Parameter IN

Berikut ini merupakan contoh penggunaan parameter `IN` pada prosedur tersimpan *`GetOfficeByCountry`* untuk menampilkan *office* yang lokasinya terletak di negara tertentu.

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
 BEGIN
 SELECT *
 FROM offices
 WHERE country = countryName;
 END //
DELIMITER ;
```

`countryName` merupakan parameter `IN` pada prosedur tersimpan. Di dalam prosedur tersimpan tersebut, kita akan menampilkan semua kantor yang berada di suatu negeri tertentu yang dispesifikasikan dengan menggunakan parameter `countryName`.

Misalkan, kita ingin mendapatkan data semua kantor di USA, kita hanya perlu memberikan nilai (USA) ke prosedur tersimpan seperti berikut:

```
CALL GetOfficeByCountry("USA");
```

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA

Untuk mendapatkan data semua kantor di Prancis, kami mengirimkan string literal *France* ke prosedur tersimpan *`GetOfficeByCountry`* sebagai berikut:

```
CALL GetOfficeByCountry("France")
```

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
▶	4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	NULL	NULL	France	75017	EMEA

Contoh Parameter OUT

Prosedur tersimpan berikut mengembalikan jumlah pesanan *(order)* berdasarkan status *order*. Prosedur tersimpan ini memiliki dua parameter:

- `orderStatus` : merupakan parameter `IN` yaitu status *order* yang akan kita hitung jumlah *order*-nya
- `total` : merupakan parameter `OUT` yang menyimpan jumlah *order* berdasarkan status *order* tertentu.

Berikut ini adalah *source code* dari yang prosedur tersimpan `CountOrderByStatus`

```
DELIMITER $$
CREATE PROCEDURE CountOrderByStatus(
 IN orderStatus VARCHAR(25),
 OUT total INT)
BEGIN
 SELECT count(orderNumber)
 INTO total
 FROM orders
 WHERE status = orderStatus;
END$$
DELIMITER ;
```

Untuk mendapatkan jumlah order yang dikirim *(shipped)* kita dapat menggunakan prosedur tersimpan `CountOrderByStatus` dan masukkan status order *'Shipped'*, dan juga argumen (@total) untuk mendapatkan nilai kembalian.

```
CALL CountOrderByStatus("Shipped",@total);
SELECT @total;
```

	@total
▶	303

Untuk mendapatkan jumlah order yang sedang dalam proses *(in process)* kita dapat memanggil prosedur tersimpan `CountOrderByStatus` seperti berikut ini.

```
CALL CountOrderByStatus('in process',@total);
SELECT @total AS  total_in_process;
```

	total_in_process
▶	6

Contoh Parameter INOUT

Contoh berikut ini menunjukkan bagaimana penggunaaan parameter `INOUT` pada prosedur tersimpan `set_counter` untuk melakukan perhitungan atau membuat *counter*.

```
DELIMITER $$
CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
BEGIN
 SET count = count + inc;
END$$
DELIMITER ;
```

Bagaimana prosedur tersimpan ini bekerja.

- Prosedur tersimpan `set_counter` ini memerlukan 2 parameter, yaitu Parameter `INOUT` (`count`) dan Parameter `IN` (`inc`), yang keduaanya bertipe *INT* dan memiliki *size* 4.
- Di dalam prosedur tersimpan ini, kita akan menambahkan nilai *counter* (`count`) dengan nilai dari parameter `inc`.

Perhatikan bagaimana kita memanggil prosedur tersimpan `set_counter`:

```
SET @counter = 1;
CALL set_counter(@counter,1); -- 2
CALL set_counter(@counter,1); -- 3
CALL set_counter(@counter,5); -- 8
SELECT @counter; -- 8
```

Dalam tutorial ini, telah dijelaskan bagaimana mendefinisiian parameter untuk suatu prosedur tersimpan, dan telah diperkenalkan beberapa mode parameter yang berbeda: `IN`, `OUT`, dan `INOUT`.

8.2.5. *MySQL IF Statement*

MySQL `IF` Statement digunakan untuk mengeksekusi sekumpulan perintah SQL berdasarkan kondisi atau nilai ekspresi tertentu. Untuk membentuk suatu ekspresi pada MySQL dapat digabungkan dengan literal, variabel, operator, dan bahkan fungsi. Suatu ekspresi dapat

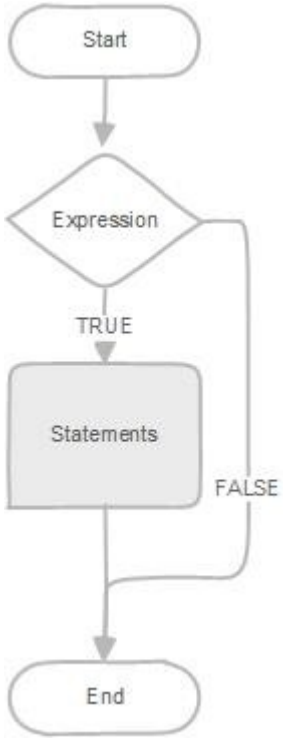
mengembalikan nilai TRUE, FALSE, atau NULL.

Berikut ini merupakan ilustrasi sintaks dari `IF` *statement*:

```
IF expression THEN
   statements;
END IF;
```

Jika evaluasi ekspresi bernilai TRUE, maka *statement* akan dieksekusi, jika tidak, kontrol diteruskan ke *statement* berikutnya setelah `END IF`.

Flowchart berikut menunjukkan `IF` *statement*:

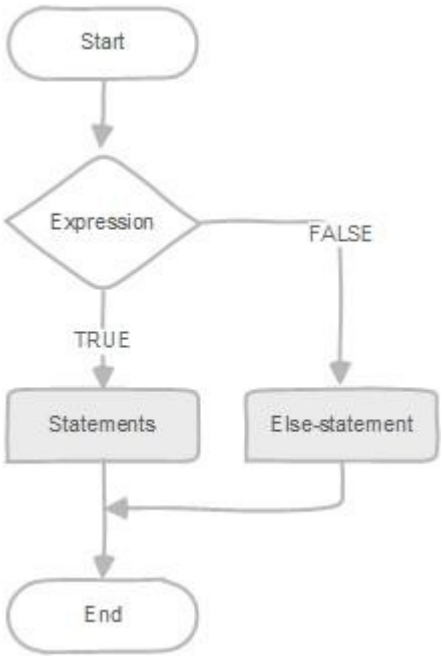


MySQL IF-ELSE Statament

Jika Anda ingin mengeksekusi *statement* ketika ekspresi yang dievaluasi tidak bernilai TRUE, Anda dapat menggunakan `IF-ELSE` *statement* sebagai berikut:

```
IF expression THEN
   statements;
ELSE
   else-statements;
END IF;
```

Flowchart berikut menunjukkan `IF-ELSE` *statement*:



Statament MySQL IF-ELSE IF-ELSE

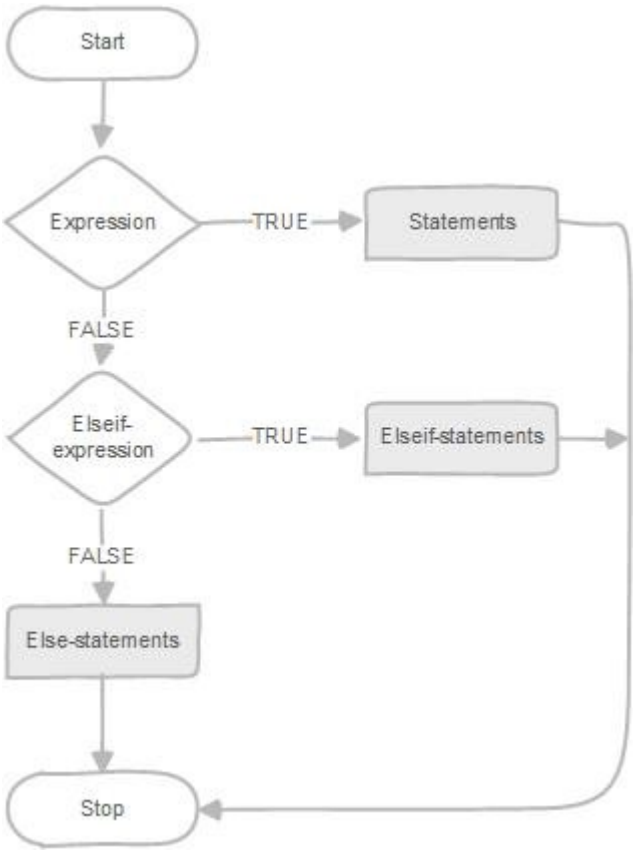
Jika Anda ingin mengeksekusi *statement* secara kondisional berdasarkan beberapa ekspresi, Anda menggunakan `IF-ELSE IF-ELSE` *statement* sebagai berikut:

```
IF expression THEN
   statements;
ELSEIF elseif-expression THEN
   elseif-statements;
...
ELSE
   else-statements;
END IF;
```

Jika evaluasi ekspresi menghasilkan nilai `TRUE`, *statement* pada cabang `IF` akan dieksekusi. Jika tidak, MySQL akan mengevaluasi ekspresi `ELSE IF` dan mengeksekusi *statement* di cabang `ELSE IF` apabila hasil evaluasi ekepresi `ELSE IF` bernilai `TRUE`.

Suatu `IF` *statement* dapat terdiri dari beberapa cabang `ELSE IF` untuk memeriksa beberapa ekspresi. Jika ekspresi yang dievaluasi tidak ada yang bernilai `TRUE`, maka *statement* di cabang `ELSE` akan dijalankan.

Flowchart berikut menunjukkan `IF-ELSE IF-ELSE` *statement*:



Contoh *Statement MySQL IF*

Contoh berikut ini mengilustrasikan bagaimana menggunakan `IF-ELSE IF-ELSE` *statement*. Prosedur tersimpan `GetCustomerLevel()` menggunakan dua parameter *customer number* dan *customer level.*

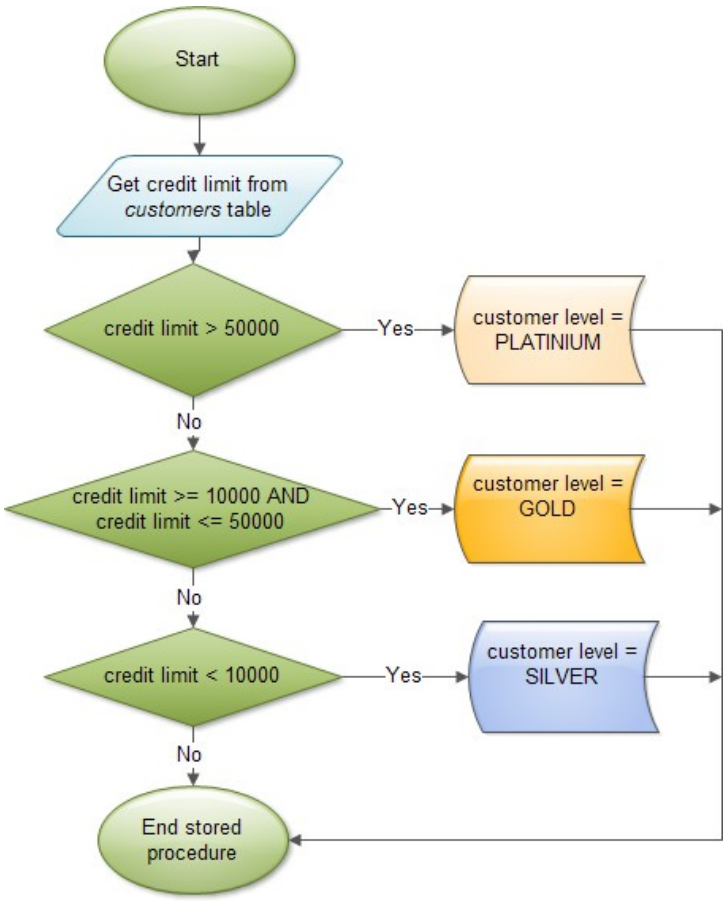
- Pertama, *customer number* digunakan untuk mendapatkan nilai *credit limit* dari tabel *customer*.
- Kemudian, berdasarkan *credit limit* tersebut digunakan untuk menentukan *customer level*: PLATINUM, GOLD, SILVER

Parameter `p_customerLevel` menyimpam level *customer* dan digunakan dengan memanggil program.

```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
    in  p_customerNumber int(11),
    out p_customerLevel  varchar(10))
BEGIN
    DECLARE creditlim double;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    IF creditlim > 50000 THEN
 SET p_customerLevel = 'PLATINUM';
    ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
        SET p_customerLevel = 'GOLD';
    ELSEIF creditlim < 10000 THEN
        SET p_customerLevel = 'SILVER';
    END IF;
```

```
END$$
```

Flowchart berikut menunjukkan logika penentuan *customer level*.



8.2.6. *MySQL CASE Statement*

Selain `IF` *statement*, MySQL juga menyediakan *statement* kondisional lain yang disebut *statement* `CASE`. `CASE` *statement* membuat kode lebih mudah dibaca dan efisien. Terdapat dua bentuk *statement* `CASE`: *simple case statement* dan *searched case statement.*

Simple CASE Statement

Perhatikan sintaks *statement simple* `CASE` berikut:

```
CASE  case_expression
   WHEN when_expression_1 THEN commands
   WHEN when_expression_2 THEN commands
   ---
   ELSE commands
END CASE;
```

Anda dapat menggunakan *simple* `CASE` *statement* untuk mengecek nilai dari suatu ekpresi dibandingkan dengan sekumpulan nilai yang unik.

`case_expression` dapat berupa ekpresi yang valid apapun. `case_expression` ini akan dibandingkan dengan `when_expression` pada setiap klausa `WHEN`, seperti `when_expression_1`, `when_expression_2`, dst. Apabila nilai `case_expression` dan `when_expression_n` sama maka *commands* pada cabang `WHEN` tersebut akan dijalankan.

Apabila tidak ada `when_expression` pada klasa `WHEN` yang cocok dengan nilai `case_expression`, maka *commands* pada klausa `ELSE` akan dijalankan. Klausa `ELSE` opsional untuk ada dalam *statement case*. Jika klausa `ELSE` dihilangkan dan tidak ada kecocokan yang ditemukan, MySQL akan memunculkan *error*.

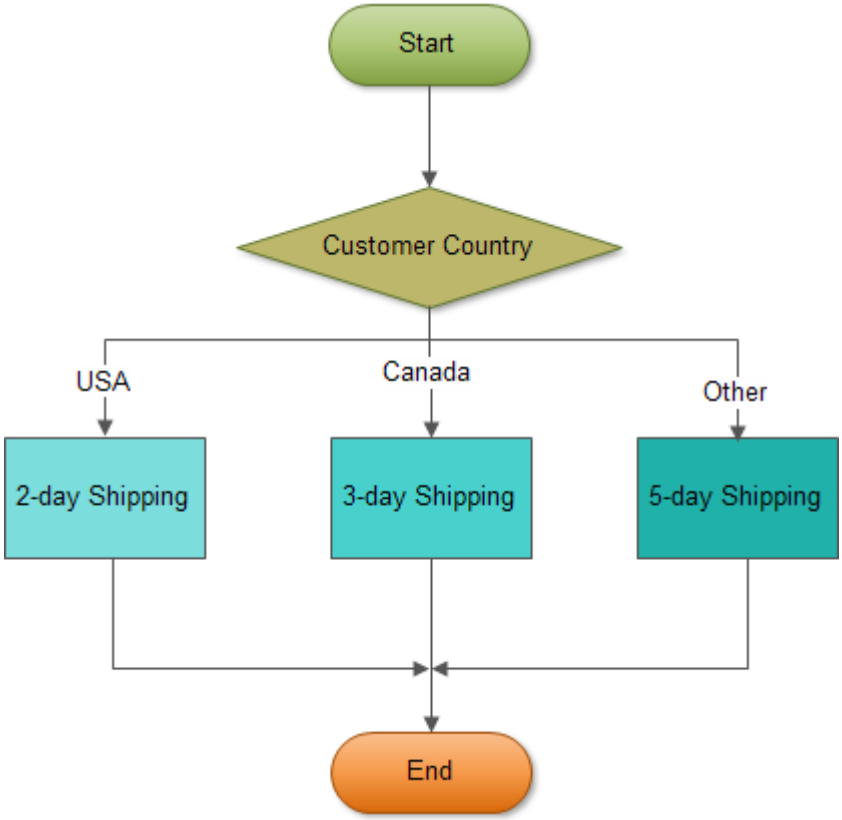
Contoh berikut mengilustrasikan bagaimana menggunakan itu *simple* `CASE` *statement:*

```
DELIMITER $$

CREATE PROCEDURE GetCustomerShipping(
 in  p_customerNumber int(11),
 out p_shiping        varchar(50))
BEGIN
    DECLARE customerCountry varchar(50);

    SELECT country INTO customerCountry
 FROM customers
 WHERE customerNumber = p_customerNumber;

    CASE customerCountry
 WHEN  "USA" THEN
    SET p_shiping = "2-day Shipping";
 WHEN "Canada" THEN
    SET p_shiping = "3-day Shipping";
 ELSE
    SET p_shiping = "5-day Shipping";
 END CASE;
END$$
```

Bagaimana prosedur tersimpan bekerja.

- Prosedur tersimpan `GetCustomerShipping()` menerima *customer number* sebagai parameter `IN` dan mengembalikan *shipping period* berdasarkan *country* dari *customer*
- Pada prosedur tersimpan, awalnya kita akan mendapatkan *country* dari *customer* berdasarkan masukan *customer number*. Kemudian, kita menggunakan *statement simple* `CASE` untuk membandingkan *country* dari *customer* untuk menentukan waktu *shipping*. Apabila lokasi *customer* di USA, maka *shipping period*-nya adalah *2-day-shipping*. Sedangkan apabila lokasi *customer* di Canada, maka *shipping period*-nya adalah *3-day-shipping*. *Customer* dari negara lain *shipping period*-nya *5-day-shipping*

Flowchart berikut menunjukkan logika penentuan *shipping time.*



Berikut ini adalah skrip pengujian untuk prosedur tersimpan diatas:

```
SET @customerNo = 112;

SELECT country into @country
FROM customers
WHERE customernumber = @customerNo;

CALL GetCustomerShipping(@customerNo,@shipping);

SELECT @customerNo AS Customer,
       @country    AS Country,
       @shipping   AS Shipping;
```

Inilah hasilnya:

	Customer	Country	Shipping
▶	112	USA	2-day Shipping

Statement Searched CASE

Simple `CASE` *statement* hanya mencocokkan nilai ekspresi dengan sekumpulan nilai yang berbeda. Untuk melakukan pencocokan yang lebih kompleks seperti *range,* kita dapat menggunakan *searched* `CASE` *statement*. *Searched* `CASE` *statement* sama dengan `IF` *statement*, namun sintaksnya jauh lebih mudah dibaca.

Sintaks *searched* `CASE` *statement* diilustrasikan sebagai berikut.

```
CASE
    WHEN condition_1 THEN commands
    WHEN condition_2 THEN commands
    ---
    ELSE commands
END CASE;
```

MySQL akan mengevaluasi setiap kondisi dalam klausa `WHEN` hingga menemukan suatu kondisi yang bernilai `TRUE`, kemudian *commands* yg berhubungan dalam klausa `THEN` akan dieksekusi.

Jika tidak ada kondisi yang bernilai `TRUE`, *commands* dalam klausa `ELSE` akan dieksekusi. Apabila klausa `ELSE` tidak didefinisikan dan tidak ada kondisi yang bernilai `TRUE`, MySQL akan memunculkan *error*.

MySQL tidak memperbolehkan terdapat perintah kosong pada klausa `THEN` atau `ELSE`. Apabila tidak ingin menangani logika dalam klausa `ELSE` dan mencegah MySQL memunculkan *error*, *commands* dapat diganti dengan blok `BEGIN END` kosong pada klausa `ELSE`.

Contoh berikut menunjukkan pengugnakan *searched* `CASE` *statement* untuk mendapatkan *customer level SILVER, GOLD,* atau *PLATINUM* berdasarkan *credit linit customer.*

```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
 in  p_customerNumber int(11),
 out p_customerLevel  varchar(10))
BEGIN
    DECLARE creditlim double;

    SELECT creditlimit INTO creditlim
 FROM customers
 WHERE customerNumber = p_customerNumber;

    CASE
 WHEN creditlim > 50000 THEN
    SET p_customerLevel = 'PLATINUM';
 WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN
    SET p_customerLevel = 'GOLD';
 WHEN creditlim < 10000 THEN
    SET p_customerLevel = 'SILVER';
 END CASE;

END$$
```

Jika nilai *credit limit:*

- lebih besar dari 50K, maka *customer* merupakan *customer* PLATINUM
- kurang dari 50K dan lebih besar dibandingkan 10K, maka *customer* merupakan *customer* GOLD
- kurang dari 10K, maka *customer* merupakan *customer* SILVER

Kami dapat menguji prosedur tersimpan dengan menjalankan skrip pengujian berikut:

```
CALL GetCustomerLevel(112,@level);
SELECT @level AS "Customer Level";
```

	Customer Level
▶	PLATINUM

8.3. Aktivitas Mahasiswa

1. Tuliskan suatu prosedur tersimpan untuk menaikkan harga kategori produk tertentu dengan persentase tertentu. Anda perlu membuat tabel *product* dengan data yang sesuai untuk menguji prosedur Anda. Sebagai alternatif, *load* database *ClassicModels* pada mesin pribadi Anda sehingga Anda memiliki akses lengkap untuk *database* tersebut. Anda perlu mengubah *DELIMITER* sebelumnya untuk membuat prosedur tersimpan.
2. Tuliskan suatu prosedur tersimpan untuk melaporkan jumlah yang dipesan secara spesifik berdasarkan bulan dan tahun untuk pelanggan yang memiliki string karakter tertentu pada namanya.
3. Tuliskan suatu prosedur tersimpan untuk mengubah *credit limit* semua pelanggan di negara tertentu dengan persentase tertentu.

BAB IX
FUNCTION DAN TRIGGER

9.1. Capaian Pembelajaran

1. Mahasiswa mampu membuat MySQL *Function* untuk memecahkan masalah sederhana pada MySQL
2. Mahasiswa mampu membuat *MySQL Trigger* untuk mengotomatisasi proses pada MySQL

9.2. Dasar Teori

9.2.1. *MySQL Stored Function*

Fungsi tersimpan (*stored function*) merupakan program tersimpan jenis khusus yang mengembalikan hasil berupa nilai tunggal. Fungsi tersimpan digunakan untuk merangkum formula umum atau aturan bisnis yang dapat digunakan kembali pada *SQL* atau program yang disimpan.

Berbeda dari prosedur tersimpan, fungsi tersimpan dapat digunakan dalam statemen *SQL* dimana pun ekspresi digunakan. Hal ini membantu meningkatkan keterbacaan dan pemeliharaan kode prosedural.

Sintaks *MySQL Stored Function*

Berikut ini mengilustrasikan sintaks paling sederhana untuk membuat fungsi tersimpan baru:

```
CREATE FUNCTION function_name(param1,param2,…)
    RETURNS datatype
   [NOT] DETERMINISTIC
 statements
```

Pertama, tentukan nama fungsi tersimpan setelah klausa `CREATE FUNCTION`.

Kedua, list semua parameter pada fungsi tersimpan di dalam tanda kurung *(parentheses)*. Secara default, semua parameter adalah parameter `IN`. Kita tidak dapat menspesifikasikan *modifier* `IN`, `OUT` atau `INOUT` pada parameter tersebut.

Ketiga, tentukan tipe data dari nilai yang dikembalikan *(return value)* pada *statement* `RETURNS`. *Return value* berupa tipe data MySQL yang valid.

Keempat, untuk parameter input yang sama, jika *stored function* mengembalikan hasil yang sama, hal ini dianggap *deterministic*; apabila tidak, *stored function* tidak deterministik. Harus diputuskan apakah *stored function* bersifat deterministik atau tidak. Jika dinyatakan secara tidak benar, *stored function* dapat menghasilkan hasil yang tidak terduga, atau optimasi yang tersedia tidak digunakan, artinya akan menurunkan kinerja.

Kelima, kode ditulis di *body stored function* . Hal ini bisa berupa *statement* tunggal atau banyak

statement. Di dalam bagian *body*, kita harus ditentukan setidaknya satu *statement* `RETURN`. *Statement* `RETURN` akan mengembalikan nilai ke *caller*. Kapan pun *statement* `RETURN` sudah tercapai, *stored function* yang dijlaankan segera dihentikan.

Contoh *MySQL Stored Function*

Perhatikan contoh penggunaan fungsi tersimpan. Kita akan menggunakan tabel *customers* pada sample database sebagai demonstrasi.

Contoh berikut ini merupakan suatu fungsi yang akan mengembalikan nilai berupa level *customer* berdasarkan *credit limit*-nya. Kita dapat menggunakan `IF` statement untuk menentukan level customer tersebut berdasarkan *credit limit*-nya.

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

    IF p_creditLimit > 50000 THEN
 SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;

 RETURN (lvl);
END
```

customerName	CustomerLevel(creditLimit)
Alpha Cognac	PLATINUM
American Souvenirs Inc	SILVER
Amica Models & Co.	PLATINUM
ANG Resellers	SILVER
Anna's Decorations, Ltd	PLATINUM
Anton Designs, Ltd.	SILVER
Asian Shopping Network, Co	SILVER
Asian Treasures, Inc.	SILVER
Atelier graphique	GOLD
Australian Collectables, Ltd	PLATINUM
Australian Collectors, Co.	PLATINUM

Selanjutnya, untuk memanggil fungsi tersimpan `CustomerLevel()`, kita dapat menggunakannya dalam *statement* `SELECT` seperti berikut:

```
SELECT
    customerName,
    CustomerLevel(creditLimit)
FROM
    customers
ORDER BY
    customerName;
```

Kita juga dapat menulis ulang prosedur tersimpan `GetCustomerLevel()` yang sudah kita buat pada tutorial *MySQL IF statement* sebelumnya seperti berikut:

```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
    IN  p_customerNumber INT(11),
    OUT p_customerLevel  varchar(10)
)
BEGIN
    DECLARE creditlim DOUBLE;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    SELECT CUSTOMERLEVEL(creditlim)
    INTO p_customerLevel;

END
```

Seperti yang dapat dilihat, prosedur tersimpan `GetCustomerLevel()` lebih mudah dibaca ketika digunakan pada fungsi tersimpan `CustomerLevel()`.

Perhatikan bahwa dfungsi tersimpan hanya mengembalikan satu nilai saja. Jika kita menyertakan *statement* `SELECT` tanpa klausa `INTO`, maka kita akan mendapatkan error.

Selain itu, jika fungsi tersimpan berisi statemen SQL, kita tidak boleh menggunakannya di dalam statemen *SQL* lainnya; jika tidak, fungsi tersimpan akan memperlambat kecepatan kueri.

9.2.2. *MySQL Trigger*

SQL trigger merupakan seperangkat statemen SQL yang disimpan dalam katalog database. *SQL trigger* dijalankan ketika terdapat suatu event yang terkait dengan suatu tabel misalnya, *insert*, *update*, atau *delete*.

SQL trigger merupakan tipe khusus dari prosedur tersimpan. *Trigger* lebih special karena tidak dipanggil secara langsung seperti prosedur tersimpan. Perbedaan utama antara *trigger* dan prosedur tersimpan adalah bahwa *trigger* dipanggil secara otomatis ketika terjadi modifikasi data yang dibuat pada tabel, sedangkan prosedur tersimpan harus dipanggil secara eksplisit.

Penting untuk memahami kelebihan dan kekurangan *SQL trigger* sehingga Anda dapat menggunakannya dengan tepat. Pada bagian berikut, kita akan membahas keuntungan dan kerugian menggunakan *SQL trigger*.

Kelebihan Penggunaan *SQL Trigger*

- *SQL trigger* menyediakan cara alternatif untuk memeriksa integritas data.
- *SQL trigger* dapat menangkap *error* dalam logika bisnis di *layer database*.
- *SQL trigger* menyediakan cara alternatif untuk menjalankan berbagai *task* yang dijadwalkan. Dengan menggunakan *SQL trigger*, tidak perlu menunggu untuk menjalankan *task* yang dijadwalkan karena *trigger* dapat dipanggil secara otomatis sebelum atau sesudah perubahan dilakukan pada data dalam tabel.
- *SQL trigger* sangat berguna untuk mengaudit perubahan data dalam tabel.

Kekurangan Penggunaan *SQL Trigger*

- *SQL trigger* hanya dapat memberikan *extended validation* dan tidak dapat mengganti semua validasi. Beberapa validasi sederhana harus dilakukan di layer aplikasi. Misalnya, untuk memvalidasi input user di sisi klien dengan menggunakan *JavaScript* atau di sisi *server* menggunakan *server-side scripting languages* seperti *JSP, PHP, ASP.NET, Perl.*
- *SQL trigger* dipanggil dan dijalankan secara tidak terlihat dari aplikasi klien, oleh karena itu, sulit untuk mengetahui apa yang terjadi di layer *database*.
- *SQL trigger* dapat meningkatkan *overhead server database*.

Implementasi *MySQL Trigger*

Pada *MySQL*, *trigger* merupakan sekumpulan statemen *SQL* yang dipanggil secara otomatis ketika perubahan dibuat ke data pada tabel terkait. Suatu *trigger* dapat didefinisikan untuk dipanggil sebelum atau setelah data diubah dengan statemen *INSERT*, *UPDATE* atau *DELETE*. Sebelum MySQL versi 5.7.2, kita hanya dapat menentukan maksimum enam trigger untuk setiap tabel.

- `BEFORE INSERT` - diaktifkan sebelum data dimasukkan ke dalam tabel.
- `AFTER INSERT` - diaktifkan setelah data dimasukkan ke dalam tabel.
- `BEFORE UPDATE` - diaktifkan sebelum data dalam tabel diperbarui.
- `AFTER UPDATE` - diaktifkan setelah data dalam tabel diperbarui.
- `BEFORE DELETE` - diaktifkan sebelum data dihapus dari tabel.

- *AFTER DELETE* - diaktifkan setelah data dihapus dari tabel.

Namun, dari *MySQL* versi 5.7.2+, dapat ditentukan beberapa *trigger* untuk *event* dan *action time* yang sama.

Ketika menggunakan suatu *statement* yang tidak menggunakan *statement INSERT, DELETE* atau *UPDATE* untuk mengubah data dalam tabel, *trigger* yang terkait dengan tabel tidak dipanggil. Misalnya, *statement* *TRUNCATE*. menghapus semua data dari tabel tetapi tidak memanggil *trigger* yang terkait dengan tabel itu.

Terdapat beberapa *statement* yang menggunakan *statement INSERT* secara implisit seperti *statement* *REPLACE* atau *statement* *LOAD DATA*. Apabila menggunakan *statement* ini, *trigger* akan terkait dengan tabel yang dipanggil.

Kita harus menggunakan nama unik untuk setiap *trigger* yang terkait dengan suatu tabel. Namun, dapat juga menggunakan nama *trigger* yang sama untuk tabel yang berbeda, hal ini merupakan praktik yang baik.

Pemberian nama *trigger* menggunakan konvensi penamaan berikut:

```
(BEFORE | AFTER)_tableName_(INSERT| UPDATE | DELETE)
```

Misalnya, `before_order_update` adalah *trigger* yang dipanggil sebelum baris dalam tabel *order* di-*update*.

Konvensi penamaan berikut sama baiknya dengan yang di atas.

```
tablename_(BEFORE | AFTER)_(INSERT| UPDATE | DELETE)
```

Misalnya, `order_before_update` sama dengan *trigger before_order_update* di atas.

Penyimpanan *MySQL Trigger*

MySQL menyimpan *trigger* di suatu direktori data misalnya, `/data/classicmodels/` dengan file bernama `tablename.TRG` dan `triggername.TRN`:

- File `tablename.TRG` memetakan *(mapping) trigger* ke tabel terkait.
- File `triggername.TRN` berisi definisi *trigger*.

Anda dapat melakukan *backup trigger* MySQL dengan menyalin file *trigger* ke folder *backup*. *Backup* dapat juga dilakukan dengan menggunakan *mysqldump tool*.

Batasan *MySQL Trigger*

MySQL trigger mencakup semua fitur yang didefinisikan dalam SQL standar. Namun, ada beberapa batasan yang harus diketahui sebelum menggunakannya pada aplikasi.

MySQL trigger TIDAK DAPAT:

- Menggunakan *statement* `SHOW`, `LOAD DATA`, `LOAD TABLE`, `BACKUP DATABASE`, `RESTORE` , `FLUSH` dan `RETURN`.
- Menggunakan *statement* yang *commit* atau *rollback* secara implisit atau eksplisit seperti `COMMIT` , `ROLLBACK` , `START TRANSACTION` , `LOCK/UNLOCK TABLES` , `ALTER` , `CREATE` , `DROP` , `RENAME`.
- Menggunakan *prepared statement* seperti `PREPARE` and `EXECUTE` .
- Menggunakan *statement* SQL dinamis.

Sintaks *MySQL Trigger*

Untuk membuat suatu *trigger* baru, kita dapat menggunakan *statemen* `CREATE TRIGGER`. Berikut ini mengilustrasikan sintaks dari *statement* `CREATE TRIGGER`.

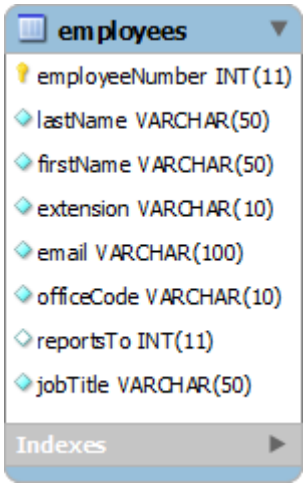
```
CREATE TRIGGER trigger_name trigger_time trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
 ---
 END;
```

Matilah kita kaji sintaks di atas secara lebih detail.

- Nama *trigger (trigger_name)* diletakkan setelah *statement* `CREATE TRIGGER`. Nama *trigger* harus mengikuti konvensi penamaan yang telah dijelaskan sebelumnya, yaitu `[trigger time]_[table name]_[trigger event]`, misalnya `before_employees_update`.
- Waktu aktivasi *trigger (trigger_time)* dapat diisi dengan `BEFORE` atau `AFTER`. Kita harus mendefinisikan waktu aktivasi ketika mendefinisikan *trigger*. *Keyword* `BEFORE` digunakan apabila ingin memproses *action* sebelum perubahan dilakukan pada tabel dan *keyword* `AFTER` apabila perlu memproses *action* setelah perubahan dilakukan.
- *Trigger event* dapat berupa `INSERT`, `UPDATE`, atau `DELETE`. *Event* ini menyebabkan *trigger* dipanggil. Satu *trigger* hanya dapat dipanggil oleh satu *event*. Untuk menentukan suatu *trigger* yang dipanggil dengan beberapa *event*, kita harus mendefinisikan multiple *trigger*, satu *trigger* untuk setiap *event*.
- Sebuah *trigger* harus dikaitkan dengan tabel tertentu. Tanpa tabel, *trigger* tidak akan ada, oleh karena itu harus ditentukan nama tabel setelah *keword* `ON`.
- *Statement* SQL ditempatkan antara blok `BEGIN` dan `END`. Disinilah didefinisikan logika untuk *trigger*.

Contoh *MySQL Trigger*

Marilah kita mulai membuat *trigger* di MySQL untuk mencatat perubahan tabel *employee*.



Pertama, buatlah sebuah tabel baru yang dinamai `employee_audit` untuk menunjukkan perubahan pada tabel `employee`. *Statement* berikut ini digunakan untuk membuat tabel `employee_audit`.

```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

Kemudian, buatlah *trigger* `BEFORE UPDATE` yang dipanggil sebelum perubahan dibuat pada tabel `employee`.

```
DELIMITER $$
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
BEGIN
    INSERT INTO employees_audit
    SET action = 'update',
     employeeNumber = OLD.employeeNumber,
        lastname = OLD.lastname,
        changedat = NOW();
END$$
DELIMITER ;
```

Di dalam *body trigger*, penggunaan *keyword* OLD untuk mengakses kolom `employeeNumber` dan `lastname` dari baris yang dipengaruhi oleh *trigger*.

Perhatikan bahwa pada *trigger* yang digunakan untuk INSERT, hanya dapat digunakan keyword `NEW` saja, tidak dapat menggunakan keyword `OLD`. Namun, pada *trigger* yang digunakan untuk DELETE, tidak ada baris baru sehingga dapat menggunakan *keyword* `OLD` saja. Pada *trigger* UPDATE, `OLD` mengacu ke baris sebelum di-update dan `NEW` mengacu ke baris setelah di-update.

Kemudian, untuk menampilkan semua *trigger* pada *database*, gunakan *statement* *`SHOW TRIGGERS`*.

```
SHOW TRIGGERS;
```

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer
before_employee_update	UPDATE	employees	BEGIN INSERT INTO employ...	BEFORE	2015-11-14 21:39:09.08	STRICT_TRANS_TABLES,NO_AUTO_CREATE_U...	root@localhost

Setelah itu, *update* tabel *`employees`* untuk memeriksa apakah *trigger* dipanggil.

```
UPDATE employees
SET
    lastName = "Phan"
WHERE
    employeeNumber = 1056;
```

Terakhir, untuk memeriksa apakah *trigger* telah dipanggil oleh *statement* `UPDATE`, kita dapat membuat *query* untuk menampilkan tabel *`employee_audit`* seperti berikut:

```
SELECT
    *
FROM
    employees_audit;
```

Berikut ini adalah output dari kueri:

id	employeeNumber	lastname	changedat	action
1	1056	Phan	2015-11-14 21:39:12	update

Seperti yang Anda lihat, *trigger* benar-benar dipanggil dan baris baru dimasukkan ke dalam tabel *`employee_audit`*.

9.3. Aktivitas Mahasiswa

1. Buatlah suatu fungsi untuk mengkonversi harga *(price)* dalam *database classicmodels* ke mata uang baru, yang akan dihitung dengan: *price**14560
2. Buatlah trigger untuk mengurangi stok suatu produk secara otomatis ketika ada penambahan pada tabel *order_details*

BAB X
NORMALISASI

10.1. Capaian Pembelajaran

1. Mahasiswa mampu memahami urgensi normalisasi dalam tabel-tabel *database*
2. Mahasiswa memahami bentuk tidak normal.
3. Mahasiswa memahami 1NF, 2NF dan 3NF.

10.2.Dasar Teori

10.2.1. Proses Normalisasi

Normalisasi merupakan suatu teknik analisis data untuk merancang sistem *database.* Normalisasi membantu *database designer* untuk memahami struktur data dalam suatu organisasi saat ini, Selain itu, normalisasi membantu perubahan dan peningkatan sistem di masa depan.

Normalisasi merupakan suatu teknik untuk menghasilkan skema relasional dengan kriteria sebagai berikut:

- *No Information Redundancy,* artinya tidak ada informasi yang redundan atau duplikat
- *No Update Anomalies,* artinya tidak ada permasalahan yang timbul ketika dilakukan *update* pada data

Hasil akhir dari normalisasi adalah sekumpulan entitas, yang menghilangkan redundansi yang tidak perlu (misalnya data yang duplikat) dan menghindari anomali.

10.2.2. Anomali

Anomali merupakan situasi yang menimbulkan permasalahan atau rawan kesalahan ketika memproses suatu tabel. Terdapat tiga jenis anomali, yaitu:

- *Update Anomalies*
- *Delete Anomalies*
- *Insert Anomalies*

Update Anomalies

Update Anomaly muncul ketika satu atau beberapa *instance* data duplikat di-*update,* tetapi tidak semua ter-*update*. Misalnya, apabila *Jones* akan berpindah alamat, maka data alamat *Jones* pada baris1 dan baris2 harus *di-update* semua.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

Delete Anomalies

Delete Anomaly terjadi ketika atribut tertentu hilang karena menghapus atribut lainnya. Contohnya, apabila siswa dengan *StudentNum* S30 meninggalkan *course,* maka seluruh informasi terkait *course Maths* dan *Computing* akan hilang.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

Insert Anomalies

Insert Anomaly terjadi ketika atribut tertentu dimasukkan ke dalam *database* tanpa adanya atribut lainnya. Hal ini merupakan kebalikan dari *delete anomaly.* Misalnya, kita tidak dapat menambahakan suatu *course* baru kecuali ada setidaknya satu siswa terdaftar di *course* tersebut.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

10.2.3. Tahapan Normalisasi

Proses normalisasi menerapkan serangkaian tes pada suatu relasi untuk menentukan apakah memenuhi atau melanggar persyaratan bentuk normal yang diberikan.

- Ketika suatu tes gagal, relasi didekomposisi menjadi relasi yang lebih sederhana dimana secara individual memenuhi uji normaliasi.
- Semakin tinggi bentuk normal, semakin dapat menangani *update anomalies* pada suatu relasi.
- Tiga bentuk nomal: 1NF, 2NF dan 3NF (diusulkan oleh *Codd*)
- Semua bentuk normal ini didasarkan pada dependensi fungsional di antara atribut-atribut suatu relasi.

Normalisasi mengikuti proses bertahap dengan serangkaian aturan. Langkah-langkah normalisasi sebagai berikut:

- Step 1: Memilih sumber data dan mengkonversikannya ke *unnormalised table (UNF)* atau disebut juga bentuk tidak normal
- Step 2: Mengubah *UNF* menjadi *first normal form/*Bentuk Normal Pertama *(1NF)* dengan menghilangkan Perulangan Grup dalam bentuk *multivalued attribute* atau kombinasi atribut dalam data yang sama). Hasil dari bentuk normal pertama adalah setiap atribut dalam tabel bernilai atomic (tidak dapat dibagi lagi)
- Step 3: Mengubah *1NF* menjadi *second normal form (2NF)* dengan menghilangkan Ketergantungan Parsial. Jadi, semua atribut non-key memiliki *functional dependency* pada *primary key* secara utuh
- Step 4: Mengubah 2*NF* menjadi *third normal form (3NF) dengan* menghilangkan Ketergantungan Transitif. Jadi, tidak ada atribut non-key yang memiliki ketergantungan dengan atribut non-key lainnya

Terkadang data masih dapat mengalami anomali pada *third normal form (3NF).* Apabila hal ini terjadi, transformasi mungkin harus dilakukan lebih lanju

- Mengubah *third normal form (3NF)* menjadi Boyce-Codd normal form (BCNF) dengan menghilangkan anomali-anomali hasil dari ketergantungan fungsional,
- Mengubah *Boyce-Codd normal form* menjadi *fourth normal form (4NF)* dengan menghilangkan ketergantungan multivalue
- Mengubah *fourth normal form (4NF)* menjadi *fifth normal form (5NF)* dengan menghilangkan anomali-anomali yang tersisa

10.2.4. Contoh Normalisasi

Kita akan mendemonstrasikan proses normalisasi (hingga 3NF) dengan menggunakan sebuah contoh. Normalisasi adalah teknik *bottom-up* untuk mendesain *database*, biasanya berdasarkan sistem yang ada (yang mungkin *paper-based*). Kita mulai dengan menganalisis dokumentasi, misalnya laporan, *screen layouts* dari sistem. Kita akan mulai dengan Laporan Manajemen Proyek, yang menggambarkan proyek yang sedang dikerjakan oleh pegawai. Laporan ini akan dinormalisasi. Masing-masing dari empat tahapan normalisasi pertama dijelaskan sebagai berikut.

Project Management Report

Project Code: PC010
Project Title: Pensions System

Project Manager: M Phillips **Project Budget:** £24,500

Employee No.	Employee Name	Department No.	Department Name	Hourly Rate
S10001	A Smith	L004	IT	£22.00
S10030	L Jones	L023	Pensions	£18.50
S21010	P Lewis	L004	IT	£21.00
S00232	R Smith	L003	Programming	£26.00

Total Staff on Project: 4 **Average Hourly Rate:** £21.88

Calculated Fields

Langkah 1

Langkah pertama yang dilakukan dalah memilih sumber data (misalnya suatu laporan) dan mengubahnya ke dalam sebuah tabel yang tidak dinormalisasi (*Unnormalised table/UNF*). Proses ini dilakukan dengan:

- Membuat kolom tabel untuk setiap item data dari sumber data (dengan mengabaikan semua bidang kalkulasi/*calculated field*). *Calculated field* adalah sesuatu yang dapat diturunkan dari informasi lain. Pada kasus ini misalnya *total staff* dan *average hourly rate*.
- Memasukkan data sampel ke dalam tabel. Data ini tidak hanya data pada *report* tetapi suatu sampel yang representatif. Pada contoh ini, beberapa pegawai dapat bekerja pada beberapa proyek. Pada perusahaan ini, pegawai yang sama juga dapat bekerja ke proyek yang berbeda dan dengan gaji per jam *(hourly rate)* yang berbeda pula.
- Mengidentifikasi suatu *key* untuk tabel (dan digaris bawahi)
- Menghilangkan duplikat data. Pada contoh ini, untuk *key* dari *project code* yang dipilih,

nilai untuk *project code, project tile, project manager,* dan *project budget* akan dipulikat apabila ada dua atau lebih *pegawai* bekerja pada proyek yang sama. *Project code* yang terpilih untuk *key* dan merupakan data duplikat, yang terkait dengan setiap *project code*, dihapus. Jangan bingung, data duplikat dengan atribut berulang akan dijelaskan pada langkah berikutnya.

Project Code	Project Title	Project Manager	Project Budget	Employee No.	Employee Name	Department No.	Department Name	Hourly Rate
PC010	Pensions System	M Phillips	24500	S10001	A Smith	L004	IT	22.00
PC010	Pensions System	M Phillips	24500	S10030	L Jones	L023	Pensions	18.50
PC010	Pensions System	M Phillips	24500	S21010	P Lewis	L004	IT	21.00
PC045	Salaries System	H Martin	17400	S10010	B Jones	L004	IT	21.75
PC045	Salaries System	H Martin	17400	S10001	A Smith	L004	IT	18.00
PC045	Salaries System	H Martin	17400	S31002	T Gilbert	L028	Database	25.50
PC045	Salaries System	H Martin	17400	S13210	W Richards	L008	Salary	17.00
PC064	HR System	K Lewis	12250	S31002	T Gilbert	L028	Database	23.25
PC064	HR System	K Lewis	12250	S21010	P Lewis	L004	IT	17.50
PC064	HR System	K Lewis	12250	S10034	B James	L009	HR	16.50

UNF: Unnormalized table

Langkah 2

Langkah kedua adalah mengubah sebuah *unnormalised table* menjadi bentuk normal pertama *(first normal form/1NF)* dengan menghilangkan atribut yang berulang menjadi suatu tabel baru yang unik (tidak berulang). Atribut berulang *(repeating attribute)* adalah *field* data dalam relasi *UNF* yang mungkin muncul dengan banyak nilai-nilai untuk sebuah nilai tunggal dari *key*-nya. Prosesnya sebagai berikut:

- Mengidentifikasi atribut berulang.
- Menghilangkan atribut berulang menjadi suatu tabel baru dengan suatu salinan *key* dari *unnormalised table (UNF)*
- Mendefinisikan suatu *key* untuk tabel baru (dan digaris bawahi). *Key* dari *unnormalised table* original selalu menjadi bagian dari *key* tabel baru *(*atau disebut menjadi *compound key)*. Nilai *key* ini harus unik untuk setiap entitas yang muncul

Catatan:

- Setelah menghapus data duplikat atribut berulang dengan mudah teridentifikasi.
- Pada tabel ssebelumnya atribut *Employee No, Employee Name, Department No, Department Name,* and *Hourly Rate* berulang. Artinya, ada potensi lebih dari satu kemunculan dari atribut ini untuk setiap *project code*. Ini merupakan atribut yang berulang dan dapat dibuat menjadi tabel baru bersama dengan menyalin *key* original (yaitu : *Project Code*).

- Suatu *key* pada *Project Code* dan *Employee No* telah ditentukan untuk tabel baru ini. Kombinasi ini unik untuk setiap baris dalam tabel.

Tabel 1NF: Atribut Berulang *(Repeating Attributes)* Dihapus

Project Code	Project Title	Project Manager	Project Budget
PC010	Pensions System	M Phillips	24500
PC045	Salaries System	H Martin	17400
PC064	HR System	K Lewis	12250

Project Code	Employee No.	Employee Name	Department No.	Department Name	Hourly Rate
PC010	S10001	A Smith	L004	IT	22.00
PC010	S10030	L Jones	L023	Pensions	18.50
PC010	S21010	P Lewis	L004	IT	21.00
PC045	S10010	B Jones	L004	IT	21.75
PC045	S10001	A Smith	L004	IT	18.00
PC045	S31002	T Gilbert	L028	Database	25.50
PC045	S13210	W Richards	L008	Salary	17.00
PC064	S31002	T Gilbert	L028	Database	23.25
PC064	S21010	P Lewis	L004	IT	17.50
PC064	S10034	B James	L009	HR	16.50

Langkah 3

Langkah ketiga adalah mengubah tabel *1NF* menjadi bentuk normal kedua (*second normal form /2NF)* dengan menghilangkan atribut *non-key* yang hanya bergantung sebagian dari *(compound) key (atau disebut juga bergantung secara partial/ partial dependencies)* menjadi suatu tabel baru.

Hal yang perlu ditentukan ”apakah *field* A bergantung pada *field* B atau sebaliknya?”. Artinya ”Diberikan satu nilai untuk A, apakah kita hanya memiliki satu nilai yang mungkin untuk B, dan sebaliknya?” Apabila jawabannya ya, A dan B harus dimasukkan ke dalam suatu relasi baru dengan A menjadi *primary key.* A harus dibiarkan dalam relasi/tabel original dan ditandai sebagai *foreign ke*.

Abaikan tabel dengan (a) kunci sederhana *(simple key)* atau (b) tanpa atribut non-kunci (ini langsung ke 2NF tanpa perubahan).

Prosesnya seperti berikut:

- Mengambil setiap atribut *non-key* secara bergantian dan jawab pertanyaan: apakah atribut in bergantung pada satu bagian *(compound) key* saja?
- Apabila iya, pindahkan atribut ke suatu tabel baru dengan salinan bagian *key* yang bergantung padanya. *Key* ini akan menjadi *primary key* di tabel baru. Garis bawahi *key* di tabel baru ini.
- Apabila tidak, cek *key* yang lain dan ulangi langkah diatas.
- Apabila tetap tidak bergantung dengan salah satu *key* saja (bergantung dengan keduanya), atribut tetap berada di dalam tabel

Catatan:

- Tabel pertama langsung menjadi 2NF karena memiliki kunci sederhana (*Project Code*).
- *Employee name, Department No,* dan *Deparment Names* bergantung pada *Employee No* saja. Oleh karena itu, atribut tersebut dipindah ke tabel baru dengan *Employye No* menjadi *key*-nya.
- Namun, *Hourly Rate* bergantung pada keduanya *Project Code* Dan *Employee No*. Sebagai seorang pegawai mungkin memiliki *hourly rate hourly rate* yang berbeda tergantung pada proyek yang sedang mereka kerjakan. Oleh karena itu, *hourly rate* tetap berada dalam tabel original.

Tabel 2NF: Dependensi *Partial Key* Dihapus

Project Code	Project Title	Project Manager	Project Budget
PC010	Pensions System	M Phillips	24500
PC045	Salaries System	H Martin	17400
PC064	HR System	K Lewis	12250

Project Code	Employee No.	Hourly Rate		Employee No.	Employee Name	Department No.	Department Name
PC010	S10001	22.00		S10001	A Smith	L004	IT
PC010	S10030	18.50		S10030	L Jones	L023	Pensions
PC010	S21010	21.00		S21010	P Lewis	L004	IT
PC045	S10010	21.75		S10010	B Jones	L004	IT
PC045	S10001	18.00		S31002	T Gilbert	L028	Database
PC045	S31002	25.50		S13210	W Richards	L008	Salary
PC045	S13210	17.00		S10034	B James	L009	HR

PC064	S31002	23.25					
PC064	S21010	17.50					
PC064	S10034	16.50					

Langkah 4

Langkah keempat adalah mengubah tabel *second normal form (2NF)* menjadi *third normal form (3NF)* dengan memindahkan ke suatu tabel baru atribut *non-key* yang lebih bergantung pada atribut *non-key* yang lain dibanding dengan *(primary) key* tabel tersebut.

Hal yang perlu ditentukan pada awalnya adalah ”apakah *field* A bergantung pada *field* B atau sebaliknya?” Artinya: ”Diberikan nilai untuk A, apakah kita hanya memilikisatu nilai yang mungkin untuk B, dan sebeliknya?” Apabila jawaannya ya, maka A dan B harus dimasukkan ke dalam sautu relasi ari, dengan A menjadi *primary key.* A harus dibiarkan dalam relasi/tabel original dan ditandai sebagai *foreign key.*

Abaikan tabel dengan nol atau hanya satu atribut *non-key* (tabel in langsung menjadi *3NF* tanpa perubaha.

Prosesnya sebagai berikut:

- Mengecek setiap atribut *non-key* apakah lebih bergantung pada atribut *non-key* lain dibandung dengan *(primary) key* tabel tersebut.
- Pindahkan atribut yang lebih bergantung ke atribut *non-key*, bersama dengan salinan atribut *non-key* ke tabel baru.
- Buat atribut *non-key* dimana atribut lain bergantung padanya menjadi *key* di tabel baru. Garis bawahi *key* dalam tabel baru ini,
- Biarkan atribut non-*key* tersebut di tabel original dan tandai sebagai *foreign key*(*)

Catatan:

- Tabel *project team* langsung dari *2NF* ke *3NF* karena hanya memiliki satu atribut *non-key.*
- *Department name* adalah lebih bergantung pada *Department No* daripada *Employee No* dan oleh karena itu pindahkan ke tabel baru. *Departemen No* adalah *key* dalam tabel baru ini dan *foreign key* di tabel *Employee*.

Tabel 3NF: Dependensi *Non-Key* Dihapus

Project Code	Project Title	Project Manager	Project Budget
PC010	Pensions System	M Phillips	24500
PC045	Salaries System	H Martin	17400
PC064	HR System	K Lewis	12250

Project Code	Employee No.	Hourly Rate
PC010	S10001	22.00
PC010	S10030	18.50
PC010	S21010	21.00
PC045	S10010	21.75
PC045	S10001	18.00
PC045	S31002	25.50
PC045	S13210	17.00
064	S31002	23.25
PC064	S21010	17.50
PC064	S10034	16.50

Employee No.	Employee Name	Department No. *
S10001	A Smith	L004
S10030	L Jones	L023
S21010	P Lewis	L004
S10010	B Jones	L004
S31002	T Gilbert	L023
S13210	W Richards	L008
S10034	B James	L0009

Department No.	Department Name
L004	IT
L023	Pensions
L028	Database
L008	Salary
L009	HR

10.2.5. Ringkasan Aturan Normalisasi

Normalisasi dimulai dengan membuat suatu *unnormalized table* dan biasanya proses normalisasi akan selesai dalam 4 tahap dan mendapatkan bentuk normal ketiga *3NF*. Anda akan melihat itu duplikasi telah dihilangkan (terpisah dari *keys* yang diperlukan untuk membuat hubungan/relasi antar tabel).

Proses normalisasi mungkin terlihat rumit. Namun, apabila mengikuti aturan sepenuhnya, dan tidak melewatkan langkah apapun, maka solusi yang tepat akan didapatkan. Apabila suatu aturan dihilangkan, kemungkinan besar akan mendapatkan tabel yang terlalu sedikit atau *key* yang salah.

Berikut ini merupakan tipe bentuk normal yang telah dibahas dalam bagian ini

1. ***First normal form***: Suatu tabel berada dalam *first normal form (1NF)* apabila tidak mengandung kolom berulang.
2. ***Second normal form***: Suatu tabel berada pada *second normal form (2NF)* jika berada dalam *first normal form (1NF)* dan hanya memuat kolom yang bergantung penuh pada suatu *key (primary key)*.
3. ***Third normal form***: Suatu tabel berada dalam *third normal form (3NF)* apabila berada dalam *second normal form (2NF)* dan semua kolon non-*key* bergantung pada *primary key*. Apabila nilai kolom non-*key* bergantung pada nilai kolom non-*key* lain (biasanya disebut dengan *transitive dependency)*. Hal ini dapat diselesaikan dengan menghilangkan kolom yang bergantung pada item non-*key* dipindahkan ke tabel lain.

10.3 Aktivitas Mahasiswa

1. Mahasiswa belajar dan mencoab contoh normalisasi di atas
2. Mahasiswa mengerjakan latihan

Latihan

Latihan Normalisasi 1

PET ID	PET NAME	PET TYPE	PET AGE	OWNER	VISIT DATE	PROCEDURE
246	ROVER	DOG	12	SAM COOK	JAN 13/2002	01 - RABIES VACCINATION
					MAR 27/2002	10 - EXAMINE and TREAT WOUND
					APR 02/2002	05 - HEART WORM TEST
298	SPOT	DOG	2	TERRY KIM	JAN 21/2002	08 - TETANUS VACCINATION
					MAR 10/2002	05 - HEART WORM TEST
341	MORRIS	CAT	4	SAM COOK	JAN 23/2001	01 - RABIES VACCINATION
					JAN 13/2002	01 - RABIES VACCINATION
519	TWEEDY	BIRD	2	TERRY KIM	APR 30/2002	20 - ANNUAL CHECK UP
					APR 30/2002	12 - EYE WASH

UNF:

1NF:

2NF:

3NF:

Latihan Normalisasi 2

INVOICE

HILLTOP ANIMAL HOSPITAL DATE: JAN 13/2002

INVOICE # 987

MR. RICHARD COOK

123 THIS STREET

MY CITY, ONTARIO

Z5Z 6G6

PET	PROCEDURE	AMOUNT
ROVER	RABIES VACCINATION	30.00
MORRIS	RABIES VACCINATION	24.00
	TOTAL	54.00
	TAX (8%)	4.32
	AMOUNT OWING	58.32

UNF:

1NF:

2NF:

3NF:

Latihan Normalisasi 3

Gallery Customer History Form

Customer Name

Jackson, Elizabeth Phone (206) 284-6783

123 – 4th Avenue

Fonthill, ON

L3J 4S4

Purchases Made

Artist	Title	Purchase Date	Sales Price
03 - Carol Channing	Laugh with Teeth	09/17/2000	7000.00
15 - Dennis Frings	South toward Emerald Sea	05/11/2000	1800.00
03 - Carol Channing	At the Movies	02/14/2002	5550.00
15 - Dennis Frings	South toward Emerald Sea	07/15/2003	2200.00

Galeri Seni Gill ingin menyimpan data tentang pelanggan, seniman, dan lukisan mereka. Mereka mungkin memiliki beberapa lukisan karya masing-masing seniman di galeri sekaligus. Lukisan dapat dibeli dan dijual beberapa kali. Dengan kata lain, galeri dapat menjual sebuah lukisan, kemudian membelinya kembali di kemudian hari dan menjualnya ke pelanggan lain.

UNF:

1NF:

2NF:

3NF:

REFERENSI

Modul ini banyak dikutip dari http://mysqltutorial.org , mereka memiliki *database* mengagumkan yang disebut *classicmodels* yang dijelaskan di seluruh bab.