# Pemrograman Basis Data
## *Menggunakan*

MySQL

RAHIMI FITRI


Diterbitkan Atas Kerjasama
Deepublish dengan Politeknik Banjarmasin

# PEMROGRAMAN BASIS DATA MENGGUNAKAN MYSQL

**UU No 28 tahun 2014 tentang Hak Cipta**

**Fungsi dan sifat hak cipta Pasal 4**

Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

**Pembatasan Pelindungan Pasal 26**

Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap:

i. Penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
iv. Penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

**Sanksi Pelanggaran Pasal 113**

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).

# PEMROGRAMAN BASIS DATA MENGGUNAKAN MYSQL

**Rahimi Fitri**

**PEMROGRAMAN BASIS DATA MENGGUNAKAN MYSQL**

**Penulis :**
**Rahimi Fitri**

**ISBN :**
**978-623-7694-16-8**

**ISBN Elektronis :**
**978-623-7694-32-8**

**Editor dan Penyunting :**
**Reza Fauzan**

**Desain Sampul dan Tata Letak :**
**Rahma Indera; Eko Sabar Prihatin**

**Penerbit :**
POLIBAN PRESS
Anggota APPTI (Asosiasi Penerbit Perguruan Tinggi Indonesia)
no.004.098.1.06.2019
Cetakan Pertama, 2020



**Redaksi :**
Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry,
Pangeran, Komp. Kampus ULM, Banjarmasin Utara
Telp: (0511)3305052
Email: press@poliban.ac.id

**Diterbitkan pertama kali oleh :**
Poliban Press, Banjarmasin, Oktober 2020

**Dicetak oleh :**
PERCETAKAN DEEPUBLISH
Jl.Rajawali, G. Elang 6, No 3, Drono, Sardonoharjo, Ngaglik, Sleman
Jl.Kaliurang Km.9,3 – Yogyakarta 55581
Telp/Faks: (0274) 4533427
Website: www.deepublish.co.id
www.penerbitdeepublish.com
E-mail: cs@deepublish.co.id

**Katalog Dalam Terbitan (KDT)**
**Rahimi Fitri** — Cet. **1**. — **Pemrograman Basis Data Menggunakan MYSQL,**
Banjarmasin: Poliban Press, Oktober 2020.

**xv**; **92** hlm.; **15.5x23 cm**

# UCAPAN TERIMA KASIH

Ucapan terima kasih kami sampaikan kepada Poliban Press karena telah mempercayakan proses percetakan buku *Pemrograman Basis Data Menggunakan MYSQL* kepada Penerbit Deepublish. Semoga buku ini dapat memberikan manfaat kepada seluruh pembaca dan kerja sama ini dapat terus terjalin.


# KATA PENGANTAR

Puji syukur ke hadirat Allah Swt. atas limpahan rahmat dan karunianya sehingga buku *Pemrograman Basis Data Menggunakan MYSQL* tahun 2020 telah dapat diselesaikan. Buku ini merupakan pengantar bagi mahasiswa Diploma III Jurusan Teknik Elektro Politeknik Negeri Banjarmasin.

Terima kasih disampaikan kepada Joni Riadi S.S.T., M.T. selaku Direktur Politeknik Negeri Banjarmasin dan Nurmahaludin, S.T., M.T. selaku Ketua Pusat Penelitian dan Pengabdian Masyarakat beserta sekretaris dan staf. Terima kasih juga disampaikan kepada Faris Ade Irawan, Reza Fauzan, Eko Sabar Prihatin dan Rahma Indera yang telah berkontribusi dalam editing serta seluruh tim Poliban Press dan semua pihak yang telah ikut membantu dalam penyelesaian buku ini.

Kami menyadari masih terdapat kekurangan dalam buku ini untuk itu kritik dan saran terhadap penyempurnaan buku ini sangat diharapkan. Semoga buku ini dapat memberi manfaat bagi semua pihak.

Banjarmasin, September 2020

Poliban Press

# PRAKATA

Dengan nama Allah Yang Maha Pengasih lagi Maha Penyayang, segala puji bagi Allah Swt., Tuhan semesta alam atas berkat rahmat dan hidayah-Nya jualah penulisan buku ajar ini dapat berjalan dengan sebaik-baiknya. Kemudian selawat serta salam tetap tercurah kepada Nabi Muhammad saw., beserta para keluarga, sahabat dan pengikut beliau sampai akhir zaman. Penulisan buku ***Pemrograman Basis Data Menggunakan MySQL*** ini adalah dalam rangka melengkapi perangkat pembelajaran mata kuliah Basis Data Lanjut pada program studi Teknik Informatika, Politeknik Negeri Banjarmasin.

Dalam penulisan buku ini, penulis menyadari sepenuhnya masih banyak sekali kekurangannya, untuk itu penulis mengharapkan masukan serta saran maupun kritik membangun dari berbagai pihak. Dengan rasa hormat, penulis mengucapkan terima kasih yang sebesar-besarnya dan penghargaan yang setinggi-tingginya kepada kedua orang tua, suami dan anak-anak yang selama ini telah memberikan dukungan semangat kepada penulis sehingga penulis dapat menyelesaikan baku ajar ini, semoga apa yang telah diberikan mendapat balasan yang lebih dari Allah Swt.

Dengan segala kerendahan dan ketulusan hati, perhatian, bantuan, dan motivasi yang diberikan, penulis mengucapkan terima kasih. Penulis juga memanjatkan doa semoga Allah Swt. memberikan balasan yang berlipat ganda. Akhirnya kepada Allah Swt. jualah sesuatu kita serahkan dan semoga buku ajar ini dapat memberikan manfaat bagi kita semua dan mendapat ridanya. Wassalamu'alaikum Warahmatullahi Wabarakatuh.

Banjarmasin, Juni 2020

Penulis

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

# BAB 1

## MENGENAL MYSQL

**Capaian Pembelajaran**

1. Mampu memahami, menguasai dan mampu mengimplementasi teori, konsep dan prinsip pemrograman *database* MySQL.
2. Mampu menginstal perangkat pendukung pemrograman basis data menggunakan MySQL.

### 1.1 Pengertian Basis Data

Pangkalan data (disebut juga basis data; bahasa Inggris: *database*) adalah kumpulan data yang terorganisir, yang umumnya disimpan dan diakses secara elektronik dari suatu sistem komputer. Pada saat pangkalan data menjadi semakin kompleks, maka pangkalan data dikembangkan menggunakan teknik perancangan dan pemodelan secara formal (https://id.wikipedia.org/wiki/Pangkalan_data).

Perangkat lunak yang dapat digunakan untuk mengelola basis data disebut sistem manajemen basis data (*database management* sistem) atau disingkat DBMS. DBMS merupakan perangkat lunak yang dirancang untuk dapat melakukan pengaturan dan mengelola koleksi data dalam jumlah yang besar dan dapat memanipulasi data secara lebih mudah. DBMS merupakan *interface* atau antar muka antara pengguna basis data (baik pengguna DBMS langsung maupun aplikasi) dengan data yang disimpan.

RDBMS atau *relationship database manajemen* sistem adalah salah satu jenis DBMS yang mendukung hubungan antar tabel. Contoh RDBMS di antaranya adalah Oracle, Ms SQL *Server*, MySQL, DB2, Ms Access.

### 1.2 *Structured Query Language*

*Structured Query Language* atau yang disingkat SQL adalah sebuah bahasa yang digunakan untuk mengakses data dalam basis data relasional.

Bahasa ini secara de facto merupakan bahasa standar yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua server basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya *(*https://id.wikipedia.org/wiki/SQL*)*.

SQL merupakan bahasa pemrograman khusus yang digunakan untuk memanajemen data dalam RDBMS. SQL biasanya berupa perintah sederhana yang berisi instruksi-instruksi untuk manipulasi dan pengambilan data pada relational *database* atau *database* yang terstruktur. Perintah SQL ini sering juga disingkat dengan sebutan '*query*'.

## 1.3 MySQL

Seiring berkembangnya zaman, teknologi semakin berkembang pesat termasuk perangkat lunak. Salah satu contoh perangkat lunak adalah MySQL yang selalu di *update* oleh prod*user*nya masing-masing. MySQL adalah pengembangan lanjutan dari proyek UNIREG yang dikerjakan oleh Michael Monty Widenius dan TcX (perusahaan perangkat lunak asal Swedia).

MySQL adalah DBMS yang *open source* dengan dua bentuk lisensi, yaitu *Free Software* (perangkat lunak bebas) dan *Shareware* (perangkat lunak berpemilik yang penggunaannya terbatas). Jadi MySQL adalah *database server* yang gratis dengan lisensi GNU *General Public License* (GPL) sehingga dapat Anda pakai untuk keperluan pribadi atau komersial tanpa harus membayar lisensi yang ada.

Seperti yang sudah disebutkan sebelumnya, MySQL masuk ke dalam jenis RDBMS (*Relational database Management* Sistem). Maka dari itu, istilah semacam baris, kolom, tabel, dipakai pada MySQL. Contohnya di dalam MySQL sebuah *database* terdapat satu atau beberapa tabel.

MySQL merupakan *database engine* atau *server database* yang mendukung bahasa *database* SQL sebagai bahasa interaktif dalam mengelola data. MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL atau DBMS yang *multithread*, *multi-user*.

## 1.4 Perbedaan SQL dan MySQL

SQL dan MySQL adalah dua hal yang berbeda. SQL adalah bahasa pemrograman yang digunakan untuk mengolah basis data, sedangkan MySQL adalah sebuah *brand software database management* sistem (DBMS) untuk mengolah basis data menggunakan bahasa SQL itu sendiri.

## 1.5 Alasan Menggunakan MySQL

Sebagai pengembang perangkat lunak terdapat beberapa alasan menggunakan MySQL untuk membuat basis data atau *database* yaitu sebagai berikut.

1. *Speed*
   MySQL menyediakan sistem basis data berkecepatan tinggi yang sempurna untuk proyek-proyek kecil hingga menengah. Ini berfungsi baik untuk perusahaan pemula, tetapi tidak memiliki banyak fitur seperti Oracle. Namun, sebagian besar perusahaan yang menggunakan MySQL tidak memerlukan fitur yang disediakan oleh Oracle karena mereka membangun fungsionalitas di tingkat menengah (https://itxdesign.com/MySQL-vs-oracle/).
2. *Opensource*
   MySQL dapat digunakan secara gratis. Meskipun demikian ada juga untuk versi komersial yang tentu sudah diberikan tambahan fitur berupa kemampuan spesifik dan layanan *technical support* dari MySQL.
3. *Scalability*
   Dapat menangani *database* dengan skala besar yaitu dengan jumlah *record* lebih dari 50 juta.
4. *Connectivity and Security*
   *Database* MySQL dapat diakses dari semua tempat di Internet dengan hak akses tertentu. MySQL adalah *database* menggunakan enkripsi *password*, jadi *database* ini cukup aman karena memiliki *password* untuk mengaksesnya.
5. *Flexibility/Portability*
   MySQL dapat digunakan untuk mengembangkan aplikasi berbasis dekstop maupun aplikasi berbasis web dengan menggunakan teknologi yang beragam. Hal Ini menunjukkan bahwa MySQL

memiliki fleksibilitas terhadap teknologi yang akan digunakan sebagai membangun aplikasi, yang menggunakan PHP, Java, C++, maupun yang lainnya. Membangun aplikasi dilakukan dengan cara menyediakan *plugin* dan *driver* yang spesifik pada masing-masing teknologi tersebut.

6. *Cross platform operating system/* Lintas Platform Sistem Operasi
   MySQL dapat berjalan stabil di berbagai sistem operasi seperti windows, Linux, Unix. Apabila diperlukan proses migrasi data antar sistem operasi dapat dilakukan dengan mudah.

*Database* MySQL memiliki dukungan terhadap *stored procedure*, fungsi, *trigger*, *view*, SQL standar ANSI, dan lain-lain yang tentu saja akan mempermudah dan mempercepat proses pengembangan aplikasi.

## 1.6 Instalasi MySQL

Pada buku ajar ini akan menggunakan XAMPP untuk dapat mengakses MySQL.

XAMPP adalah sebuah aplikasi *open source* terkait pengelolaan *server* yang dikembangkan oleh Apache Friends. Karena bersifat *open source*, aplikasi ini bisa Anda digunakan secara gratis. Selain itu, sesuai namanya, X pada XAMPP berarti *cross platform*. Artinya, mendukung berbagai platform seperti Windows, macOS dan Linux. XAMPP sendiri terdiri dari Apache, MariaDB (yang dikembangkan dari MySQL), PHP dan Perl. XAMPP juga memberikan solusi sederhana dan cukup ringan dijalankan, memungkinkan membuat *web server* lokal untuk melakukan pengetesan *website*. XAMPP dapat dijalankan pada Mac dan Linux. Dalam buku ajar ini penggunaan aplikasi XAMPP diimplementasikan pada sistem operasi Windows.

Untuk menginstal XAMPP maka lakukan langkah-langkah berikut ini.

1. **Unduh XAMPP**

   *Download* XAMPP melalui *website* Apache Friends di *link* ini *https://www.apachefriends.org/download.html.*


Gambar 1.1 Tampilan Awal *Website*

**2. Instal XAMPP**

a. Lakukan instalasi setelah selesai mengunduh. Selama proses instalasi mungkin akan melihat pesan yang menanyakan apakah yakin akan menginstalnya. Silakan tekan Yes untuk melanjutkan instalasi.

Gambar 1.2 Tampilan Instalasi XAMPP

b. Klik tombol *Next*.


Gambar 1.3 Tampilan Awal XAMPP (1)

c. Pada tampilan selanjutnya akan muncul pilihan mengenai komponen mana dari XAMPP yang ingin dan tidak ingin di instal. Beberapa pilihan seperti Apache dan PHP adalah bagian penting untuk menjalankan *website* dan akan otomatis diinstal. Silakan centang MySQL dan phpMyAdmin, untuk pilihan lainnya biarkan saja.

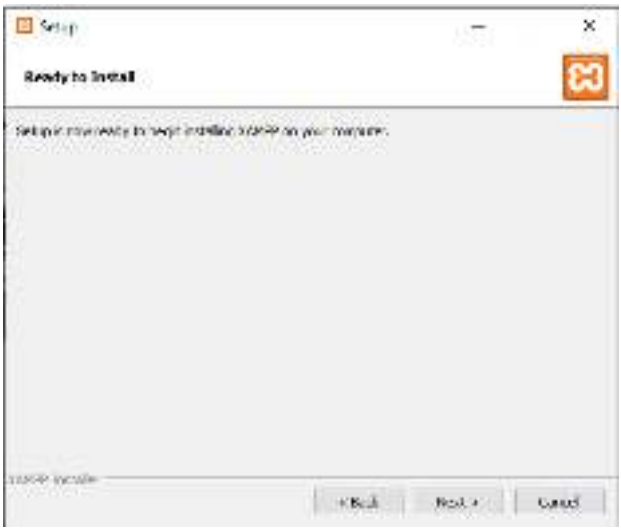
Gambar 1.4 Tampilan Awal XAMPP (2)

d. Berikutnya silakan pilih *folder* tujuan di mana XAMPP ingin di instal, pada tutorial ini pada direktori *C:\xampp*.

Gambar 1.5 Tampilan Awal XAMPP (3)

e. Pada halaman selanjutnya, akan ada pilihan apakah ingin menginstal Bitnami untuk XAMPP, di mana nantinya dapat digunakan untuk memasang aplikasi WordPress, Drupal, dan Joomla secara otomatis.

Gambar 1.6 Tampilan Awal XAMPP (4)

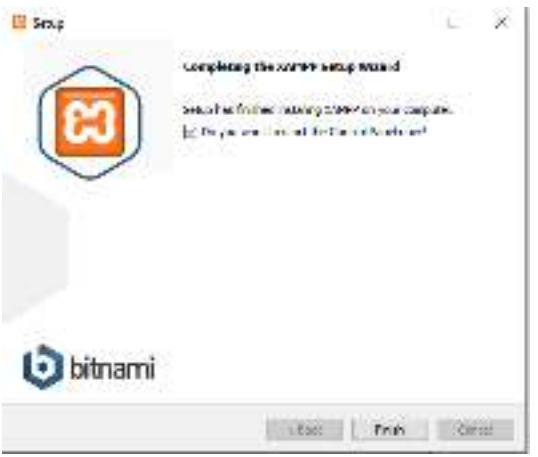


Gambar 1.7 Tampilan Awal XAMPP (5)

f. Pada langkah ini proses instalasi XAMPP akan dimulai. Silakan klik tombol ***Next***.


Gambar 1.8 Tampilan Awal XAMPP (6)

g. Setelah berhasil diinstal, akan muncul notifikasi untuk langsung menjalankan *control panel*. Silakan klik Finish.




Gambar 1.9 Tampilan Awal XAMPP (7)

h. Pilih Bahasa yang digunakan klik *Save*.

Gambar 1.10 Tampilan Pilihan Bahasa

### 3. Langkah 3: Jalankan XAMPP

Silakan buka aplikasi XAMPP kemudian klik tombol *Start* pada Apache dan MySQL. Jika berhasil dijalankan, Apache dan MySQL akan berwarna hijau seperti gambar di bawah ini.


Gambar 1.11 Tampilan Control Panel XAMPP

Untuk menjalankan MYSQL maka modul MYSQL harus diaktifkan dengan menekan tombol start pada bagian modul MYSQL.

Gambar 1.12 Tampilan Control Panel XAMPP (2)

## 1.7 Studi Kasus dalam Praktik dan Latihan

### 1. Skenario

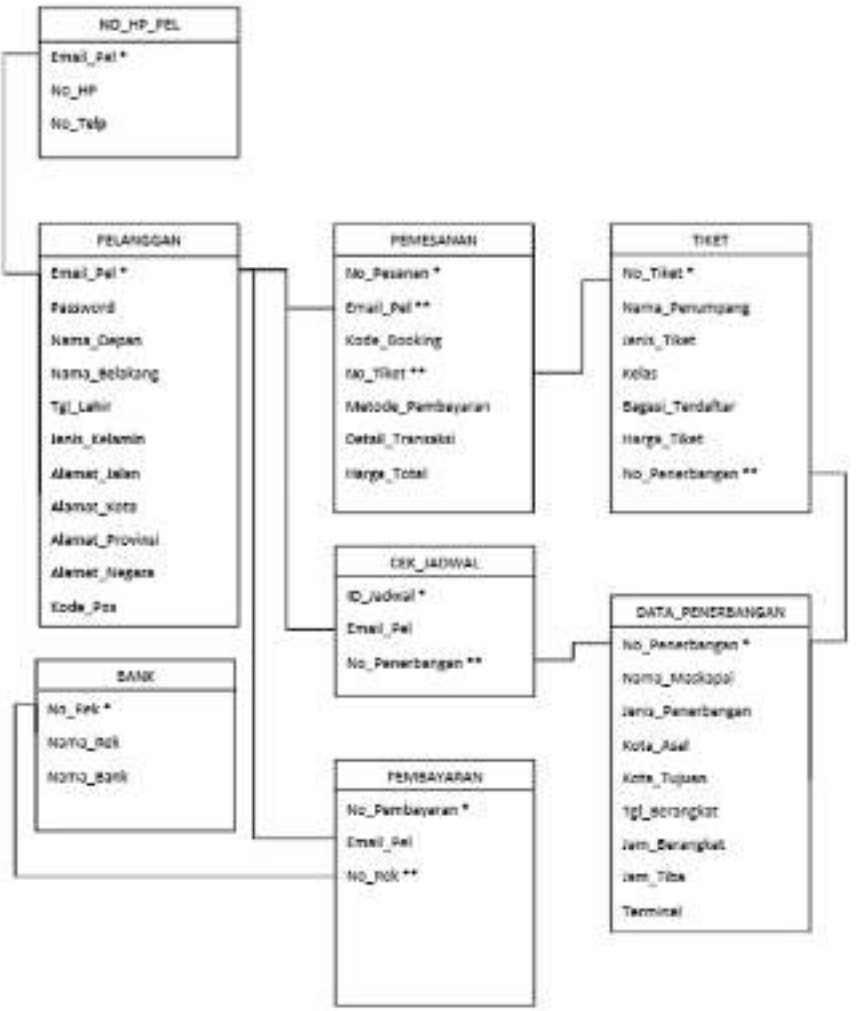
Judul *database* yang untuk kegiatan praktikum adalah *database* Penerbangan. Rancangan fisik *database* ini terdiri dari lima *table master* dan tiga *table* transaksi. Pada *table master* ada *table* pelanggan, tiket, data_penerbangan, bank, no_hp_pel. Sedangkan pada *table* transaksi ada *table* pemesanan, dan cek_jadwal.

Skenario kasus untuk pembuatan *database* penerbangan ini adalah sebagai berikut.

> Pelanggan yang ingin memesan tiket pesawat meminta data penerbangan terlebih dahulu melalui website yang sudah disediakan dengan menginputkan tujuan penerbangan dan waktu penerbangan untuk mengecek jadwal penerbangan yang tersedia. Kemudian pelanggan memilih jadwal penerbangan yang diinginkan. Setelah itu, pelanggan melakukan registrasi *online* untuk mendapatkan *username* dan *password*. Kemudian, menginputkan data-data pelanggan lainnya berupa nama, alamat, nomor hp serta nama-nama penumpang. Setelah berhasil melakukan registrasi, pelanggan diminta membayar baik melalui transfer bank, kartu kredit, *internet banking* dengan nominal yang sudah ditentukan. Lalu melakukan konfirmasi pembayaran melalui *form* yang telah disediakan di website. Setelah pelanggan membayar, pelanggan akan menerima e-Mail berupa tiket dan bukti pembayaran yang nantinya dapat dicetak.

**2. Desain Relasi Basis Data Penerbangan**

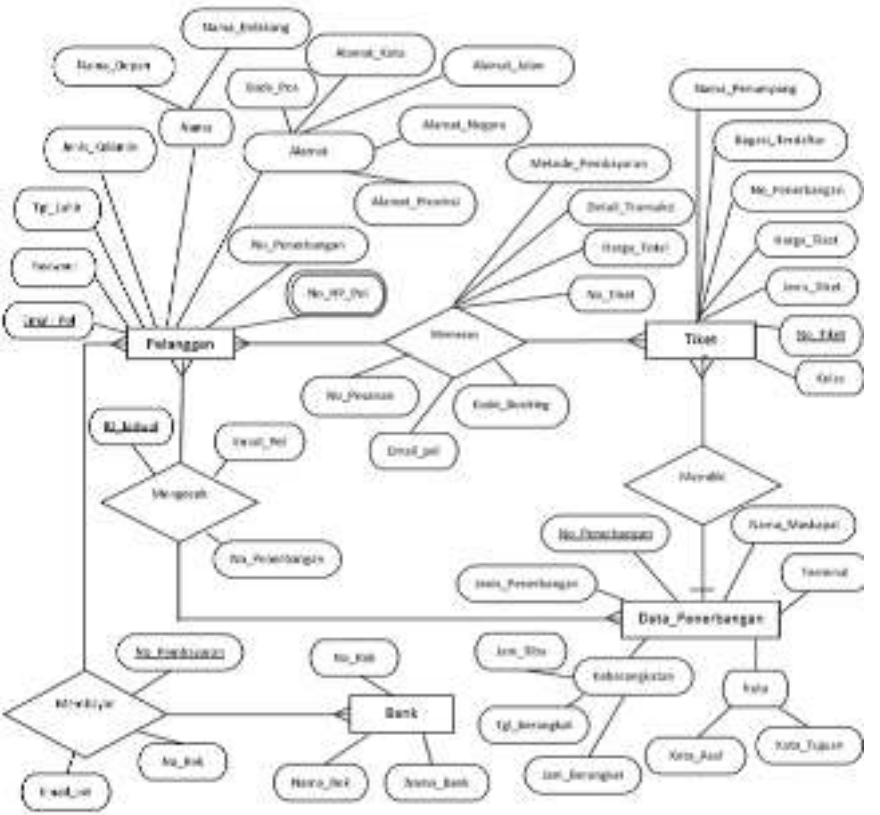
Berdasarkan ilustrasi di atas, dapat menghasilkan desain relasi basis data penerbangan seperti di bawah ini.




Gambar 1.13 Desain Relasi *database* Penerbangan

### 3. *Entity Relationship Diagram* (ERD) Tabel Master

ERD (*Entity Relationship Diagram*) merupakan suatu model jaringan yang menggunakan susunan data yang disimpan pada sistem secara abstrak. ERD juga menggambarkan hubungan antara satu entitas yang memiliki sejumlah atribut dengan entitas yang lain dalam suatu sistem yang terintegrasi.




Gambar 1.14 Desain ERD *Database* Penerbangan

# BAB 2

## DATA DEFINITION LANGUAGE

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL (DDL) pada DBMS MySQL.
2. Mampu menerjemahkan dan mengimplementasikan konsep *Entity Relationship Diagram* (ERD) ke dalam bentuk DDL pada DBMS MySQL.

### 2.1 Data Definition Language (DDL)

*Data Definition Language* adalah bagian dari SQL. DDL berfungsi lebih kepada memanipulasi struktur *database*. DDL digunakan untuk membuat *database*, membuat tabel beserta struktur tabel, mengubah struktur tabel, membuat relasi antar tabel, menghapus *database*, dan menghapus tabel. Berikut adalah perintah DDL.

#### 2.1.1 Perintah Create

Perintah ini digunakan untuk membuat *database* dan membuat tabel.

**1. Membuat *database***

Untuk membuat *database* dengan perintah "**CREATE *database***" atau dengan mengikuti pola sebagai berikut.

```
MariaDB [(none)]> create database nama_database;
```

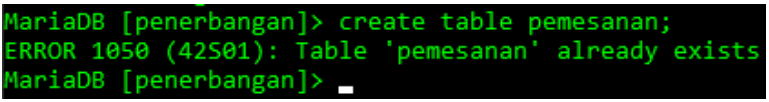
Gambar 2.1 Sintaks Membuat *database*

Di mana *create database* adalah perintah untuk membuat *database*, sedangkan *nama_database adalah nama dari *database* yang dibuat. Untuk penamaan *database* tidak boleh menggunakan spasi, spasi diganti dengan (_). Contoh pembuatan *database* sebagai berikut.


Gambar 2.2 Membuat *database* Penerbangan

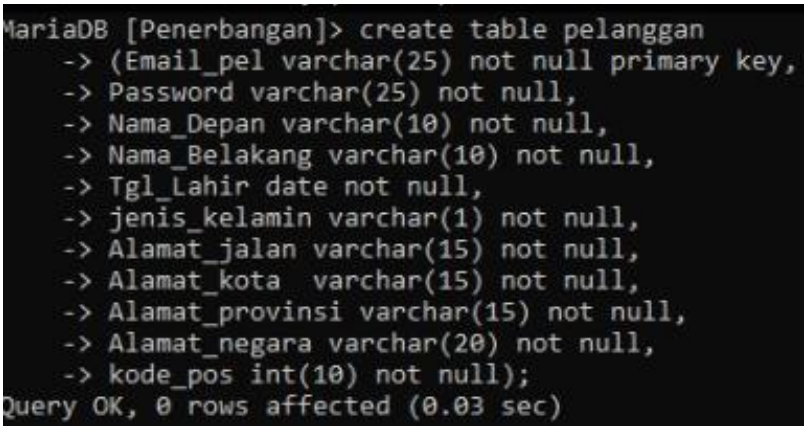
**2. Membuat tabel**

Untuk membuat tabel dengan perintah "**CREATE *TABLE***".




Gambar 2.3 Sintaks Membuat Tabel (*Error*)

Perintah tersebut *error*, karena untuk membuat tabel pada MySQL kita harus memasukkan minimal 1 buah *field*/kolom di dalamnya. Contoh perintah yang benar adalah sebagai berikut.




Gambar 2.4 Membuat Tabel Pelanggan

**3. Melihat Struktur Tabel**

Untuk melihat struktur pada tabel kita bisa menggunakan perintah "**DESC**" atau "**DESCRIBE**". Contohnya adalah sebagai berikut.

```
MariaDB [Penerbangan]> desc pelanggan;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| Email_pel      | varchar(25) | NO   | PRI | NULL    |       |
| Password       | varchar(25) | NO   |     | NULL    |       |
| Nama_Depan     | varchar(10) | NO   |     | NULL    |       |
| Nama_Belakang  | varchar(10) | NO   |     | NULL    |       |
| Tgl_Lahir      | date        | NO   |     | NULL    |       |
| jenis_kelamin  | varchar(1)  | NO   |     | NULL    |       |
| Alamat_jalan   | varchar(15) | NO   |     | NULL    |       |
| Alamat_kota    | varchar(15) | NO   |     | NULL    |       |
| Alamat_provinsi| varchar(15) | NO   |     | NULL    |       |
| Alamat_negara  | varchar(20) | NO   |     | NULL    |       |
| kode_pos       | int(10)     | NO   |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
11 rows in set (0.02 sec)
```

Gambar 2.5 Melihat Struktur Tabel Pelanggan

Selanjutnya buatlah tabel yang menyimpan nomor hp Pelanggan dengan struktur tabel sebagai berikut.

```
MariaDB [penerbangan1]> desc no_hp_pel;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| email_pel | varchar(25) | NO   | PRI | NULL    |       |
| no_hp     | char(15)    | NO   |     | NULL    |       |
| no_telp   | char(15)    | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

Selanjutnya buatlah tabel yang menyimpan data penerbangan dengan struktur tabel sebagai berikut.

```
MariaDB [penerbangan1]> desc data_penerbangan;
+----------------+------------+------+-----+---------+-------+
| Field          | Type       | Null | Key | Default | Extra |
+----------------+------------+------+-----+---------+-------+
| No_Penerbangan | varchar(20) | NO   | PRI | NULL    |       |
| Nama_Maskapai  | varchar(20) | NO   |     | NULL    |       |
| Kota_Asal      | varchar(15) | NO   |     | NULL    |       |
| Kota_Tujuan    | varchar(15) | NO   |     | NULL    |       |
| Tgl_Berangkat  | date       | NO   |     | NULL    |       |
| Jam_Berangkat  | time       | NO   |     | NULL    |       |
| Jam_Tiba       | time       | NO   |     | NULL    |       |
| Terminal       | varchar(20) | NO   |     | NULL    |       |
| kursi_eko      | varchar(5) | NO   |     | NULL    |       |
| kursi_bis      | varchar(5) | NO   |     | NULL    |       |
+----------------+------------+------+-----+---------+-------+
```

Selanjutnya buatlah tabel yang menyimpan data pemesanan tiket dengan struktur tabel sebagai berikut.

```
MariaDB [penerbangan1]> desc pemesanan;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| no_pesanan       | char(8)     | NO   | PRI | NULL    |       |
| email_pel        | varchar(25) | NO   |     | NULL    |       |
| kode_booking     | char(30)    | NO   |     | NULL    |       |
| no_tiket         | int(10)     | NO   |     | NULL    |       |
| metode_pembayaran | varchar(15) | NO   |     | NULL    |       |
| detail_transaksi | varchar(20) | NO   |     | NULL    |       |
| harga_total      | int(20)     | NO   |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
```

Selanjutnya buatlah tabel yang menyimpan data tiket dengan struktur tabel sebagai berikut.

```
MariaDB [penerbangan1]> desc tiket;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| no_tiket         | int(10)     | NO   | PRI | NULL    |       |
| nama_penumpang   | varchar(25) | NO   |     | NULL    |       |
| jenis_tiket      | varchar(15) | NO   |     | NULL    |       |
| kelas            | varchar(15) | NO   |     | NULL    |       |
| bagasi_terdaftar | varchar(10) | YES  |     | NULL    |       |
| harga_tiket      | int(20)     | NO   |     | NULL    |       |
| no_penerbangan   | varchar(20) | NO   |     | NULL    |       |
| ekonomi          | char(5)     | YES  |     | NULL    |       |
| bisnis           | char(5)     | YES  |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
```

Selanjutnya buatlah tabel yang menyimpan data bank dengan struktur tabel sebagai berikut.

```
MariaDB [penerbangan1]> desc bank;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| no_rek    | varchar(15) | NO   | PRI | NULL    |       |
| nama_rek  | varchar(30) | NO   |     | NULL    |       |
| nama_bank | varchar(10) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

### 2.1.2 Mengubah Struktur Tabel dengan Perintah Alter

Pada saat membangun suatu basis data adakalanya diperlukan perubahan struktur tabel yang pernah kita buat sebelumnya dikarenakan adanya perubahan kebutuhan sistem. Perubahan struktur yang terjadi dapat dalam bentuk penambahan kolom baru (ADD), perubahan lebar dan jenis kolom yang telah dibuat sebelumnya (MODIFY), atau dapat juga melakukan penghapusan kolom dan indeks (DROP), perubahan nama kolom sebelumnya (CHANGE), perubahan nama tabel (RENAME), dan lain sebagainya.

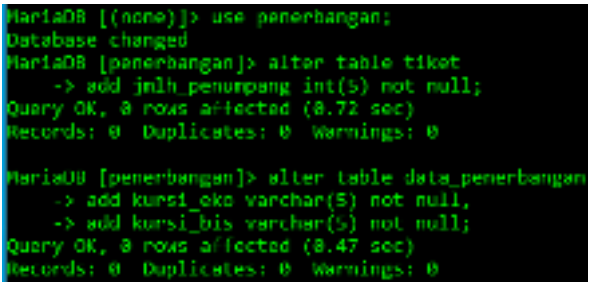
Catatan penting dalam kegiatan mengubah struktur tabel adalah apa pun perubahan yang dilakukan pada kolom ataupun tabel tentu memiliki

mempunyai dampak langsung pada data yang telah dimasukkan sebelumnya.

Perintah DDL yang dapat digunakan untuk mengubah struktur tabel adalah menggunakan perintah "ALTER *TABLE*".

**1. Menambahkan *field* (ADD)**

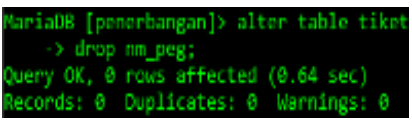
Perintah yang digunakan untuk menambahkan *field* adalah "`ALTER TABLE *NAMA_TABEL ADD *NAMA_FIELD`". Contohnya adalah sebagai berikut.




Gambar 2.6 Menambahkan *Field*

**2. Menghapus *field* (DROP)**

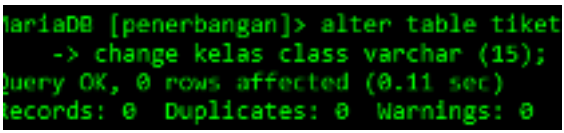
Perintah yang digunakan untuk menghapus *field* adalah "`ALTER TABLE *NAMA_TABEL DROP *NAMA_FIELD`". Contohnya adalah sebagai berikut.




Gambar 2.7 Menghapus *Field*

**3. Mengubah Nama *Field* (CHANGE)**

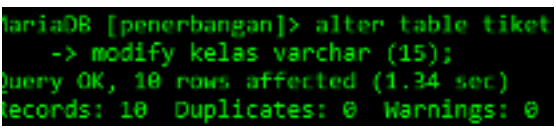
Selain untuk menambahkan *field* pada tabel, perintah ALTER juga memungkinkan untuk mengubah *field* pada tabel. Perintah yang digunakan untuk mengubah *field* adalah "`ALTER TABLE *NAMA_TABEL CHANGE *NAMA_FIELD`". Contohnya adalah sebagai berikut.




Gambar 2.8 Mengubah Nama *Field*

**4. Mengubah Lebar dan Jenis *Field* (MODIFY)**

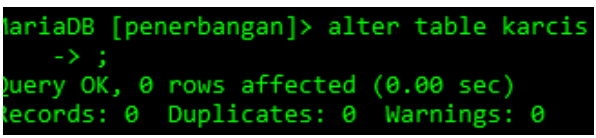
Perintah yang digunakan untuk mengubah lebar dan jenis *field* adalah "`ALTER TABLE *NAMA_TABEL MODIFY *NAMA_FIELD`". Contohnya adalah sebagai berikut.




Gambar 2.9 Mengubah Lebar dan Jenis *Field*

**5. Mengubah Nama *Table* (RENAME)**

Perintah yang digunakan untuk mengubah nama tabel adalah "`ALTER TABLE *NAMA_TABEL RENAME *NAMA_TABEL_BARU`". Contohnya adalah sebagai berikut.




Gambar 2.10 Mengubah Nama Tabel

### 2.1.3 Perintah *Drop*

Perintah *drop* digunakan untuk menghapus *database* dan menghapus tabel.

**1. Menghapus *database***

Perintah yang digunakan untuk menghapus *database* adalah "DROP *database* `*NAMA_DATABASE`". Contohnya adalah sebagai berikut.

```
MariaDB [penerbangan]> drop database test;
Query OK, 0 rows affected (0.12 sec)
```

Gambar 2.11 Menghapus *database*

**2. Menghapus Tabel**

Perintah yang digunakan untuk menghapus tabel adalah "`DROP TABLE *NAMA_TABEL`". Contohnya adalah sebagai berikut.

```
MariaDB [penerbangan]> drop table pembeli;
Query OK, 0 rows affected (0.22 sec)
```

Gambar 2.12 Menghapus Tabel

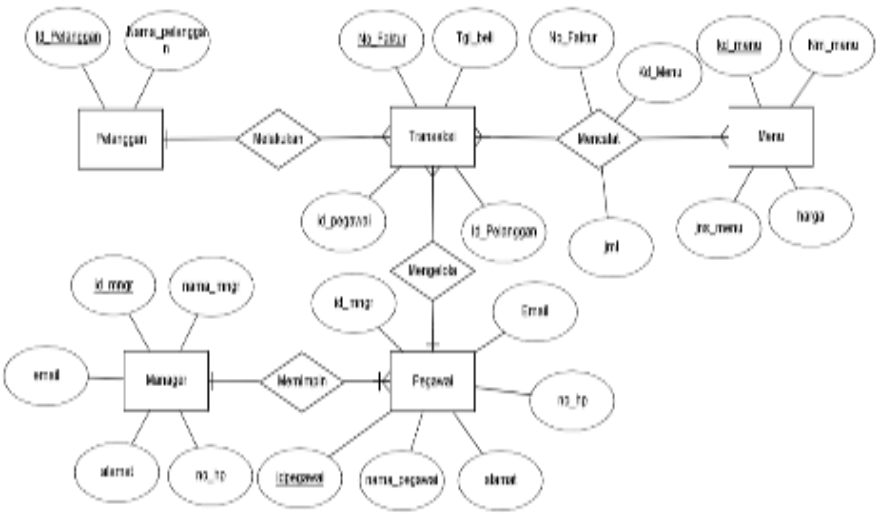
## 2.2 Soal Latihan

Sebagai tugas Latihan maka kerjakan lah soal berikut ini:

1. Buat *database* dengan nama dbCaffe dengan scenario kasus sebagai berikut.

Dimalam yang cerah ini, Amel dan kelima sahabatnya pergi mengunjungi kafe Askioo. Sesampainya di sana mereka langsung disambut oleh salah satu pelayan kafe di sana yaitu Zainal. Mereka diarahkan oleh Dio, pelayan yang cukup tampan, ke meja yang masih kosong dan menyerahkan buku menu. Masing-masing dari mereka mulai memesan makanan yang ada di sama. Mulai dari Amel memesan 1 cheese cake dan 1 coffe americano, Clara memesan 1 pasta extra mozarella dan 1 gelas air putih, Dara 1 jus stroberi, Tyas 2 lava cake, Wendy 1 cheese burger dan 1 kaleng soda, dan Mira 1 porsi salad buah. Pesanan mereka disiapkan oleh Filo sebagai koki sekaligus barista di kafe tersebut. Setelah sekian lama menunggu tiba akhirnya pesanan mereka. Setelah selesai dengan segala basa-basi membahas gosip terbaru Amel dan kawan-kawan berniat pergi tapi sebelum itu mereka membayar semua pesanan mereka ke kasir. Ternyata yang si kasir adalah Salsa salah satu teman mereka dan sekaligus karyawan di kafe tersebut. Tak lama kemudian kafe tutup dan Salsa melaporkan hasil dari pendapatan hari ini kepada manajer yaitu Alexander.

2. Berdasarkan skenario tersebut berikut ini adalah ERD yang sesuai dengan studi kasus di atas.




3. Dengan menggunakan MySQL buatlah *database* dbcaffe_nama, nama adalah nama masing-masing mahasiswa.
4. Pilih dan buka *database* tersebut. Buat tabel seperti yang di bawah ini

a. Tabel Pelanggan, dengan struktur berikut ini.

| | |
|---|---|
| id_Pelanggan | Char (2) not null primary key |
| Nama_pelanggan | Varchar (20) not null |

b. Tabel Pegawai, dengan struktur berikut ini.

| | |
|---|---|
| idpegawai | Char (4) not null primary key |
| id_mngr | Char (4) not null primary key |
| Nama_pegawai | Varchar (20) not null |
| Alamat | Varchar (30) not null |
| No_hp | Int (12) not null |

c. Tabel Manager, dengan struktur berikut ini.

| | |
|---|---|
| id_mngr | Char (4) not null primary key |
| Nm_mngr | Varchar (20) not null |
| Alamat | Varchar (30) not null |
| No_hp | Int (12) not null |

d. Tabel Menu, dengan struktur berikut ini.

| | |
|---|---|
| Kd_menu | Char (3) not null primary key |
| Nm_menu | Varchar (20) not null |
| Jns_menu | Varchar (10) not null |
| Harga | Int (10) not null |

e. Transaksi, dengan struktur berikut ini.

| | |
|---|---|
| Tgl_trx | Date not null |
| no_faktur | Char (6) not null |
| Tgl_beli | Date |
| Id_pelanggan | Char (2) not null |
| Idpegawai | char (4) not null |

f. Tabel detail transaksi dengan struktur berikut ini.

| | |
|---|---|
| Kd_menu | Char (3) |
| no_faktur | Char (6) |
| Jml | Int (4) |

g. Tabel log_menu, dengan struktur berikut ini.

| | |
|---|---|
| Kd_menu | Char (3) |
| Nm_menu | Varchar (20) |
| Harga | Int (10) |

5. Perbaiki struktur tabel dengan rincian seperti soal berikut ini.
   a. Tambahkan *field* email pada Tabel pegawai dan manajer dengan struktur data varchar(30) null.
   b. Ganti lebar data *field* alamat menjadi 50.

c. Ganti struktur data *field* harga pada Tabel menu menjadi int (12) not null *default* 0.
d. Ganti nama Tabel pegawai menjadi karyawan.
e. Ganti idPegawai menjadi kdKaryawan dengan tipe & lebar data yang sama.
f. Ganti nama_pegawai menjadi nmKaryawan dengan tipe & lebar data yang sama.

6. Pada tabel transaksi lakukan perubahan sebagai berikut:
   a. Tambahkan *field* total_harga int (12) not null *default* 0.
7. Rubahlah nama tabel dengan ketentuan sebagai berikut.
   a. Dari karyawan menjadi pegawai.
   b. Tabel pegawai kembali menjadi karyawan.

# BAB 3

## DATA MANIPULATION LANGUAGE

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL (DML) pada DBMS MySQL.
2. Mampu menerjemahkan dan mengimplementasikan konsep *Entity Relationship Diagram* (ERD) ke dalam bentuk DML pada DBMS MySQL.

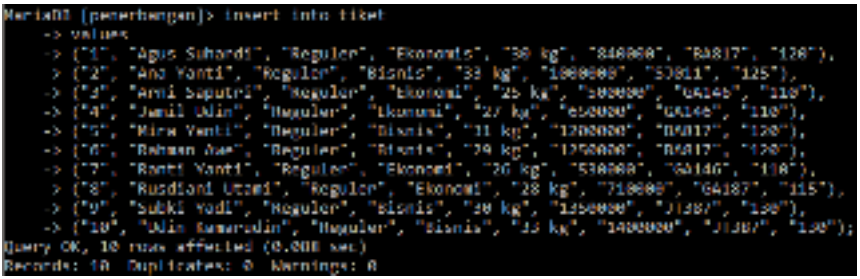
### 3.1 *Data Manipulation Language* (DML)

*Data Manipulation Language* adalah sekumpulan elemen sintaks yang mirip dengan bahasa pemrograman komputer yang digunakan untuk memanipulasi data, misalnya memasukkan data, menghapus data, dan memperbarui data yang ada di dalam *database*.

Berikut adalah perintah yang termasuk dalam kelompok DML (*Data Manipulation Languange*).

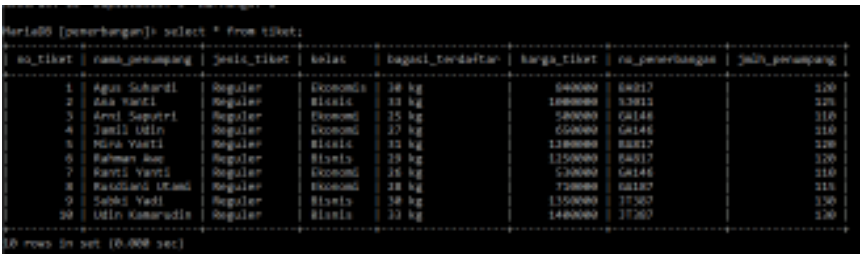
#### 3.1.1 Perintah *Insert*

*Insert* digunakan untuk menambahkan data baru kedalam tabel. Perintah ini dapat dibuat setelah *database* dan tabel berhasil dibuat. Berikut adalah contoh melakukan 10 *insert* data ke tabel tiket menggunakan "INSERT INTO" dan melihat hasilnya menggunakan perintah "SELECT".



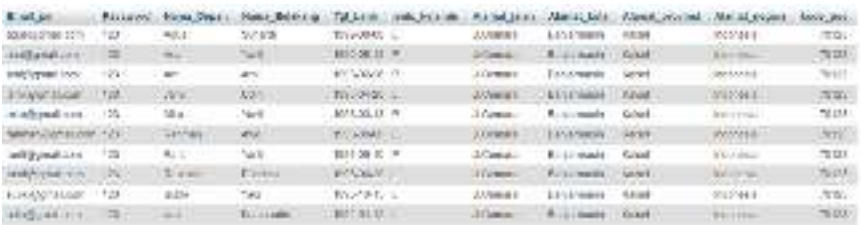

Gambar 3.1 *Insert* Data Tabel Tiket

Maka hasil dari tabel tiket seperti tampilan berikut ini.




Gambar 3.2 Hasil *Insert* Data Tabel Tiket

Selanjutnya tambahkan data pada tabel-tabel yang telah dibuat sebelumnya dengan isian sebagai berikut.



Gambar 3.3 Isian Tabel Pelanggan

| no_rek | nama_rek | nama_bank |
|---|---|---|
| 192487021245523 | Rusdiani | BNI |
| 192487022211023 | Bubki | Mega |
| 192487092115523 | Ranti | Mandiri |
| 192487092212113 | Udin | BJB |
| 192487221142123 | Rahman | BRI |
| 192812920187827 | agus | BRI |
| 192812920913123 | Jamil | BNI |
| 192812920815227 | Ani | BCA |
| 192887820843423 | Mira | BRI |
| 193332920627827 | Ana | BCA |

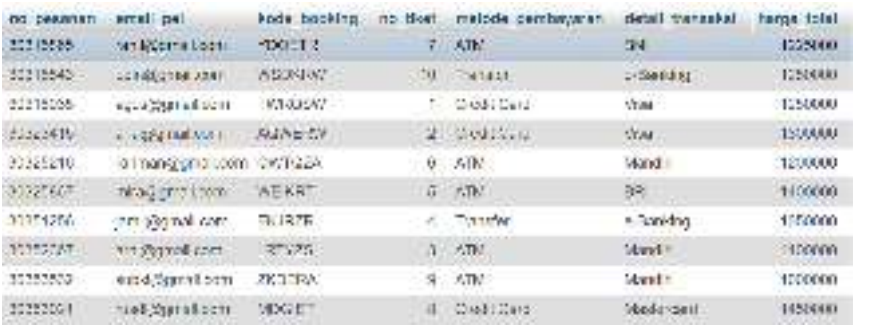
Gambar 3.4 Isian Tabel Bank

| No_Penerbangan | Nama_Maskapai | Kota_Asal | Kota_Tujuan | Tgl_Berangkat | Jam_Berangkat | Jam_Tiba | Tarif/Kursi | Kursi_ekst | Kursi_bis |
|---|---|---|---|---|---|---|---|---|---|
| GA347 | Garuda | Banjarmasin | Bali | 2013-04-15 | 14:15:00 | 16:15:00 | 0 | 10 | 10 |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| GA107 | Garuda Indonesia | Banjarmasin | Surabaya | 2013-04-15 | 14:15:00 | 15:15:00 | A | 100 | 25 |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| SR215 | Lion | Banjarmasin | Surabaya | 2013-04-15 | 07:15:00 | 08:15:00 | A | 100 | 10 |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |

Gambar 3.5 Isian Tabel Data_penerbangan

| email_pel | no_hp | no_telp |
|---|---|---|
| agus@gmail.com | 082144566678 | 021266533 |
| ana@gmail.com | 087756784321 | 021334412 |
| ani@gmail.com | 085789980001 | 021123454 |
| jamil@gmail.com | 088759760987 | 021113315 |
| mira@gmail.com | 081351[illegible]9080 | 021222317 |
| rahman@gmail.com | 081221344444 | 021555777 |
| ranti@gmail.com | 082133447854 | 021678794 |
| rusdi@gmail.com | 082122334455 | 021555088 |
| subki@gmail.com | 085765436780 | 021088008 |
| udin@gmail.com | 08779599[illegible]57 | 021543253 |

Gambar 3.6 Isian Data Tabel Nomor Hp Pelanggan



Gambar 3.7 Isian Data Tabel Pemesanan

Gambar 3.8 Isian Data Tabel Tiket

### 3.1.2 Perintah *Select*

*Select* adalah perintah yang paling sering digunakan pada SQL, sehingga kadang-kadang istilah *query* dirujukkan pada perintah *select*. *Select* digunakan untuk menampilkan data dari satu atau lebih tabel, biasanya dalam sebuah basis data yang sama.

Umumnya perintah *select* sebagai berikut.

```
SELECT * FROM nama_tabel WHERE predikat;
```

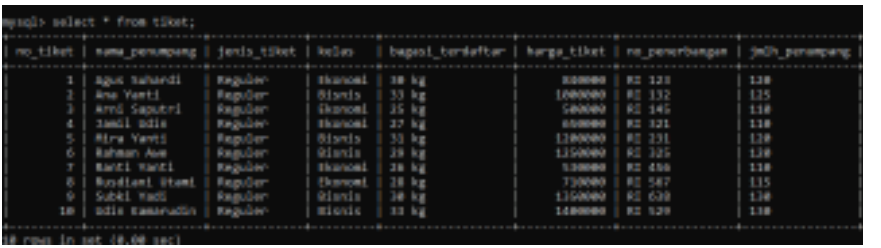
Jika kata “WHERE tidak dicantumkan, maka secara otomatis akan menampilkan seluruh isi tabel terkait. Beberapa bagian yang terdapat pada *select* adalah sebagai berikut.

**1. *Where***

*Where* berfungsi untuk menampilkan data sesuai syarat yang diberikan. Beberapa operator yang biasa digunakan adalah sama dengan (=), tidak sama dengan (< >), lebih kecil (<), lebih kecil atau sama (<=), lebih besar (>), dan lebih besar atau sama (>=).

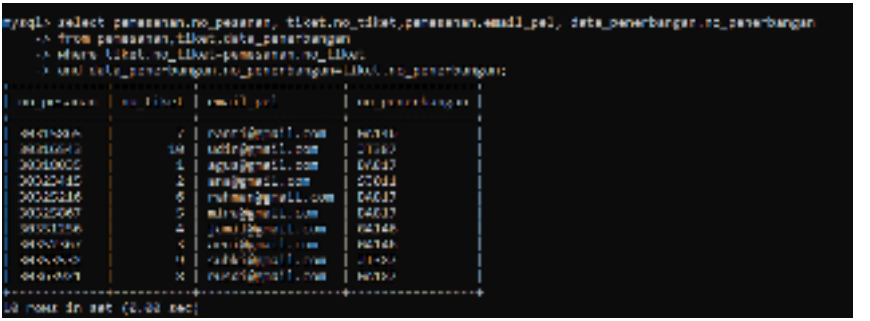
**2. *AND*, *OR*, dan *NOT***

*AND* dan *OR* berfungsi untuk memadukan antara satu kondisi dengan kondisi lainnya. Sedangkan *NOT* digunakan untuk kondisi ingkaran. Contoh penggunaan perintah *SELECT* untuk menampilkan seluruh data yang terdapat pada *table* tiket.



Gambar 3.9 Contoh Penggunaan *Select*

Menampilkan data (menggunakan perintah *SELECT*) yang terdapat pada tiga buah tabel yakni tabel pemesanan, tabel tiket, dan tabel data_penerbangan menggunakan klausa “WHERE”.



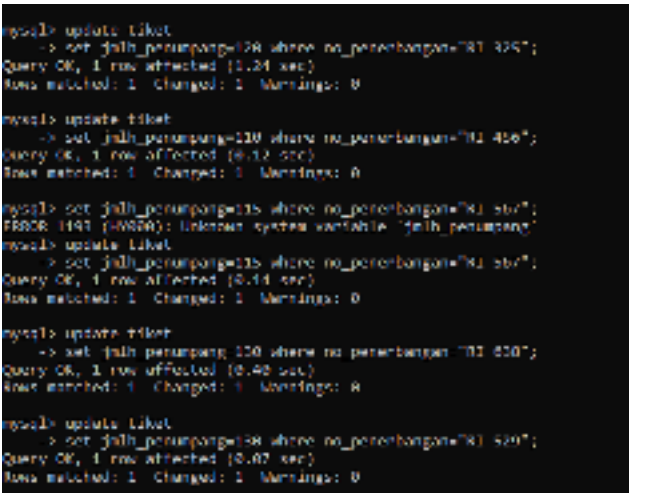
Gambar 3.10 Contoh Penggunaan Select dengan Kondisi

### 3.1.3 Perintah *Update*

*Update* digunakan untuk mengubah/memperbaharui data yang sudah ada. Umumnya perintah *update* sebagai berikut.

```
UPDATE nama_tabel SET kolom1 = nilai1, kolom2 = nilai2,
... WHERE kondisi;
```

Contoh melakukan UPDATE untuk melakukan *set* data jumlah penumpang (jmlh_penumpang) sesuai dengan nomor penerbangan (no_penerbangan) pada *table* tiket.



Gambar 3.11 *Update* Tabel Tiket

**Peringatan!!**

*Pada saat update data jika tidak mengetikkan kondisi atau tidak menggunakan perintah where, maka seluruh data pada field akan ikut terubah sesuai data baru yang kalian masukkan*

### 3.1.4 Perintah *Delete*

Perintah *delete* berguna untuk menghapus data yang ada di tabel. Tentunya data yang dihapus yakni data yang dianggap sudah tidak digunakan lagi. Umumnya perintah *delete* sebagai berikut.

```
DELETE FROM nama_tabel WHERE kondisi;
```

Contoh menghapus data pada tabel pemesanan sesuai dengan nomor pesanan (no_pesanan) sebagai berikut.

```
mysql> delete from pemesanan where no_pesanan="0000001";
Query OK, 1 row affected (0.06 sec)
```

Gambar 3.12 Sintaks *Delete*

## 3.2 Macam-macam bentuk Penggabungan (*Join*)

*Join* berfungsi untuk menggabungkan dua tabel yang didapat melalui proses seleksi melalui kolom/kata kunci tertentu untuk menemukan suatu informasi yang lengkap.

1. ***Cross Join***

   *Cross join* adalah bentuk penggabungan yang tidak memerlukan adanya kondisi. Bentuk umum:

   ```
   SELECT kolom1, kolom2 FROM tabel1 CROSS JOIN tabel2;
   ```

2. ***Inner Join***

   *Inner join* merupakan bentuk penggabungan yang mirip dengan *cross join*, bedanya pada *Inner Join* ditambahkan kondisi. Bentuk umum:

   ```
   SELECT kolom FROM tabel1 INNER JOIN tabel2 ON kondisi;
   ```

3. ***Straight Join***

   *Straight join* mirip dengan *inner join* bedanya tidak menggunakan "*where*". Bentuk umum:

   ```
   SELECT kolom FROM Tabel1 STRAIGHT JOIN tabel2;
   ```

4. ***Left (outer) Join***

   *Join* yang menampilkan semua data disebelah kiri dan menampilkan data disebelah kanan yang cocok dengan data disebelah kiri dari tabel yang dijoinkan. Jika tidak ada data yang cocok, maka secara otomatis akan di set *null*. Bentuk umum:

   ```
   SELECT kolom FROM tabel1 LEFT JOIN tabel2 ON kondisi;
   ```

**5. *Right (outer) Join***

Merupakan lawan dari *left join*. Bentuk umum:

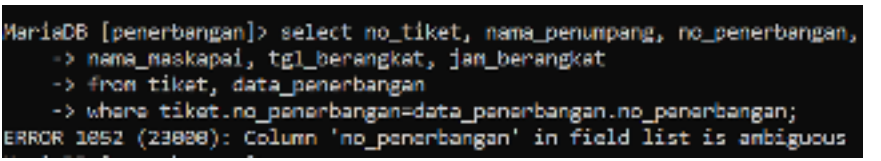
```
SELECT kolom FROM tabel1 RIGHT JOIN tabel2 ON
kondisi;
```

Contoh penggunaan salah satu jenis join (Inner Join) dengan menampilkan "no_tiket" dan "no_penerbangan" yang ada di *table* tiket dan data_penerbangan. Kemudian digabungkan menggunakan perintah INNER JOIN dengan syarat *field* "no_penerbangan" yang ada di kedua *table* itu sama.

```
mysql> select tiket.no_tiket, data_penerbangan.no_penerbangan
    -> from tiket INNER JOIN data_penerbangan
    -> ON tiket.no_penerbangan=data_penerbangan.no_penerbangan;
+-----------+-----------------+
| no_tiket  | no_penerbangan  |
+-----------+-----------------+
|         1 | BA817           |
|         2 | SJ011           |
|         3 | GA146           |
|         4 | GA146           |
|         5 | BA817           |
|         6 | BA817           |
|         7 | GA146           |
|         8 | GA187           |
|         9 | JT387           |
|        10 | JT387           |
+-----------+-----------------+
10 rows in set (0.00 sec)
```

Gambar 3.13 Contoh Penggunaan *Select* dengan *Inner Join*

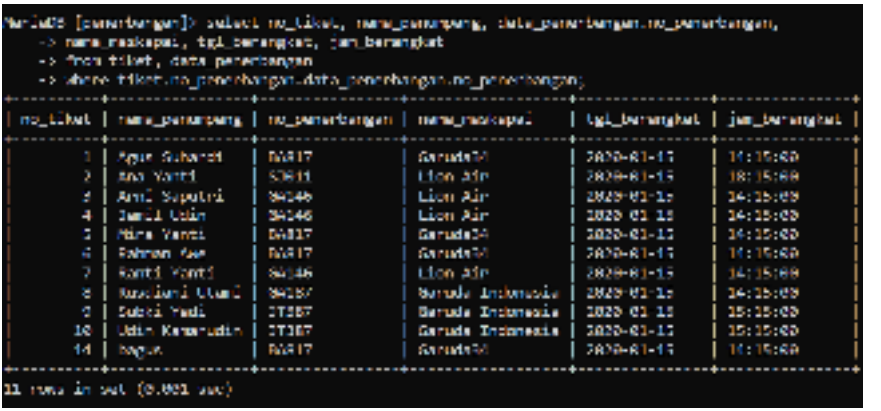
Selain menggunakan perintah *join* untuk menampilkan data yang diperoleh dari relasi beberapa tabel, kita juga dapat bermain dengan menggunakan perintah *where.* Sebagai contoh Menampilkan "no_tiket" dan "no_penerbangan" yang ada di *table* tiket dan data_penerbangan.




Gambar 3.14 Contoh menampilkan data dari beberapa tabel (1)

Ketika diketikkan perintah seperti Gambar 3.14 maka akan muncul pesan *column 'no_penerbangan' in field list is ambiguous* mengapa bisa demikian? Perhatikan pesan yang muncul yang menyatakan bahwa kolom no_penerbangan ambigu yang artinya terdapat nama kolom yang sama pada kedua tabel. No_penerbangan terdapat di dalam tabel tiket dan terdapat di dalam tabel data_penerbangan.

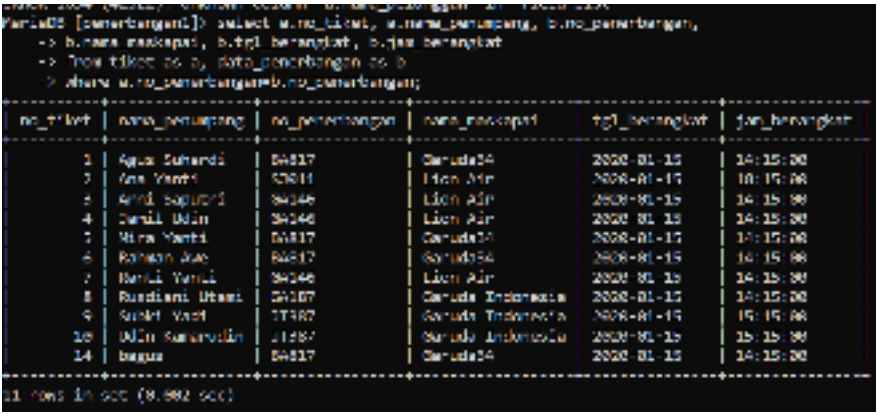
Jika terdapat dua kolom dengan nama yang sama maka kolom tersebut harus menyertakan nama tabelnya, contoh *tiket.no_penerbangan*. Perhatikan perintah seperti pada Gambar 3.15 berikut ini.




Gambar 3.15 Contoh menampilkan data dari beberapa tabel (2)

Ketika menampilkan data dari beberapa tabel, pada bagian *from* harus disebutkan nama-nama tabel yang saling berhubungan yang dapat digunakan untuk menampilkan data. Untuk mempermudah dan

mempersingkat penulisan nama tabel kita dapat menggunakan alias atau perintah (*as).* Perhatikan perintah pada Gambar 3.16 berikut ini.



Gambar 3.16 Menampilkan data dari beberapa tabel dengan tabel alias

### 3.3 Operator Pembanding Dan Operator Logika

Operator pembanding yakni operator untuk membandingkan dua nilai. Hasil perbandingan akan bernilai benar jika kondisi perbandingan tersebut benar, dan bernilai salah jika kondisi perbandingan tersebut bernilai salah. Operator logika adalah operator yang digunakan untuk membandingkan logika benar dan logika salah. Kedua operator ini biasanya digunakan untuk menampilkan data dengan suatu syarat atau kondisi. Berikut jenis-jenis operator pembanding dan operator logika.

**1. Operator Pembanding**

Berikut ini adalah operator pembanding yang biasanya digunakan pada Bahasa SQL, operator tersebut di antaranya adalah lebih besar(>), lebih kecil (<), lebih kecil atau sama dengan (<=), sama dengan (=), dan sama dengan (!=).

Contoh operator pembanding dalam menampilkan data penerbangan yang menggunakan maskapai "Garuda Indonesia" menggunakan perintah SELECT pada *table* data_penerbangan sebagai berikut.

Gambar 3.17 Contoh Penggunaan Operator Pembanding

**2. Operator Logika**

Operator logika yang dapat digunakan adalah “dan” (AND atau &&), “atau” (OR atau ||), dan “lebih besar atau sama dengan” (NOT atau !). Contoh penggunaan operator logika, misalnya akan menampilkan data penerbangan yang kota_tujuan bukan “Jakarta” (<> Jakarta) sebagai berikut.

Gambar 3.18 Contoh Penggunaan Operator Logika

Contoh selanjutnya dalam penerapan operator pembanding dan operator logika akan dibahas berikut ini. Misalnya kita akan menampilkan data pelanggan yang tanggal lahirnya sebelum tanggal 1993-09-10, maka perintahnya adalah sebagai berikut.

```
MariaDB [penerbangan]> select email_pel, nama_depan, nama_belakang, tgl_lahir
    -> from pelanggan where tgl_lahir < "1993-09-10"
    -> order by  nama_depan;
+-----------------+------------+---------------+------------+
| email_pel       | nama_depan | nama_belakang | tgl_lahir  |
+-----------------+------------+---------------+------------+
| arni@gmail.com  | Arni       | Arni          | 1993-02-26 |
| mira@gmail.com  | Mira       | Yanti         | 1993-03-13 |
| rusdi@gmail.com | Rusdiani   | Rusdiani      | 1993-04-20 |
+-----------------+------------+---------------+------------+
3 rows in set (0.003 sec)
```

Gambar 3.19 Penggunaan operator pembanding mencari tanggal lahir

Penggunaan kata "*order by*" adalah untuk mengurutkan data berdasarkan nama depan. Pada MySQL terdapat kelonggaran dalam hal penulisan tanggal selama format penulisannya tetap mengikuti aturan "tahun-bulan-tanggal". Misal "1993-09-10" dapat ditulis "1993-09-10", atau "1993#09#10" atau "1993.09.10", atau 19930910, atau 930910. Contoh selanjutnya adalah menampilkan data pelanggan yang lahir sebelum tanggal 1995-04-20 dan berjenis kelamin "laki-laki".

```
MariaDB [penerbangan]> select email_pel, nama_depan, nama_belakang, tgl_lahir
    -> tgl_lahir, jenis_kelamin
    -> from pelanggan where tgl_lahir <= "1995#04#20"
    -> and jenis_kelamin="L"
    -> order by  nama_depan;
+-----------------+------------+---------------+------------+---------------+
| email_pel       | nama_depan | nama_belakang | tgl_lahir  | jenis_kelamin |
+-----------------+------------+---------------+------------+---------------+
| jamil@gmail.com | Jamil      | Udin          | 1995-04-20 | L             |
| rusdi@gmail.com | Rusdiani   | Rusdiani      | 1993-04-20 | L             |
| subki@gmail.com | Subki      | Yadi          | 1993-10-15 | L             |
| udin@gmail.com  | Udin       | Kamarudin     | 1994-04-12 | L             |
+-----------------+------------+---------------+------------+---------------+
4 rows in set (0.001 sec)
```

Gambar 3.20 Penggunaan Operator logika dan Operator Pembanding

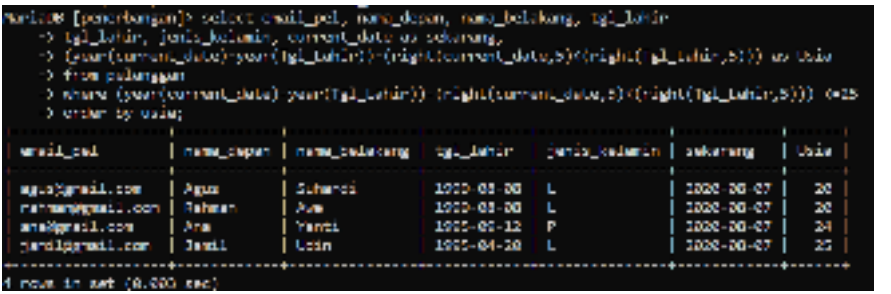
Selanjutnya untuk mencoba penggunaan format tanggal dapat anda coba pada contoh di atas kemudian tangkapan layar dapat dimasukkan kedalam laporan.

Selanjutnya adalah penggunaan operator pembanding dan operator logika tingkat lanjut. Latihan selanjutnya adalah menampilkan semua data pelanggan berikut usianya saat ini, beberapa cara dapat dilakukan seperti mengurangkan tahun sekarang dengan tahun lahir, akan tetapi cara ini masih kurang valid khususnya bagi pelanggan yang lahir di akhir tahun, dan proses perhitungan di awal tahun berikutnya, tentu saja usia pelanggan sudah dihitung 1 tahun. Berikut ini ada perintah SQL yang akan kita gunakan untuk menghitung tanggal lahir.

```
VariaDB [penerbangan]> select Nama_Depan, Nama_Belakang,
    -> current_date as Sekarang,
    -> (year(current_date)-year(Tgl_Lahir))-
    -> (right(current_date,5)<(right(Tgl_Lahir,5))) as Usia
    -> from pelanggan;
+------------+--------------+------------+-------+
| Nama_Depan | Nama_Belakang | Sekarang   | Usia  |
+------------+--------------+------------+-------+
| Agus       | Suhardi      | 2020-08-06 |    26 |
| Ana        | Yanti        | 2020-08-06 |    24 |
| Arni       | Arni         | 2020-08-06 |    27 |
| Jamil      | Udin         | 2020-08-06 |    25 |
| Mira       | Yanti        | 2020-08-06 |    27 |
| Rahman     | Amo          | 2020-08-06 |    26 |
| Ranti      | Yanti        | 2020-08-06 |    26 |
| Rusdiani   | Rusdiani     | 2020-08-06 |    27 |
| Subki      | Yadi         | 2020-08-06 |    26 |
| Udin       | Kamarudin    | 2020-08-06 |    26 |
+------------+--------------+------------+-------+
10 rows in set (0.004 sec)
```

Gambar 3.21 Penerapan Operator Pembanding untuk menghitung umur

Selanjutnya adalah menampilkan data pelanggan yang umur nya di bawah 25 tahun dan data diurutkan berdasarkan usia




Gambar 3.22 Penerapan Operator Pembanding dengan berbagai kondisi

## 3.4 Soal Latihan

Jawablah pertanyaan berikut ini kemudian sertakan hasil ambilan layar pada laporan tugas yang anda buat.

1. Memasukkan data pada tabel berikut ini.
   a. Tabel manajer, tampilkan hasil.

b. Tabel karyawan, Pada kolom id_manager masukkan salah satu id_manager yang telah dimasukkan pada *table* manager. Kemudian tampilkan.
c. Tabel pelanggan, tampilkan hasil.
d. Tabel menu, Menampilkan hasil.
e. Tabel transaksi, Pada tabel transaksi masukkan id_pegawai dan id_pelanggan berdasarkan data yang telah dimasukkan pada *table* pelanggan dan pada *table* pegawai sebelumnya. Kemudian tampilkan hasil.
f. Tabel detail transaksi (sesuaikan data yang dimasukkan berdasarkan data pada tabel transaksi dan tabel menu).

2. Menampilkan kdKaryawan, nmKaryawan dari tabel karyawan.
3. Menampilkan kdKaryawan, nmKaryawan dari tabel karyawan, urut berdasarkan nama.
4. Menampilkan id_pelanggan, nama_pelanggan dari tabel pelanggan, urut berdasarkan nama (*descending*).
5. Mengupdate data: alamat karyawan dengan kdKaryawan = KD05 menjadi 'Banjarbaru'.
6. Menampilkan id_pelanggan, total_harga, tgl_trx dari tabel transaksi. Di mana tgl_trx sebelum 15 November 2019.
7. Menampilkan id_pelanggan, total_harga, tgl_trx dari tabel transaksi. Di mana tgl_trx antara 11 oktober 2019 sampai 16 Desember 2019, dan id_pelanggan = 14.
8. Menampilkan tabel transaksi, yang kd_menu nya selain M15.
9. Hanya menampilkan tabel transaksi dengan kd_menu M15.
10. Menampilkan jumlah menu dengan harga lebih dari 25000.
11. Menampilkan rata-rata harga dari tabel transaksi.
12. Menampilkan nilai maksimal harga dari tabel transaksi.
13. Merelasikan tabel pelanggan dengan tabel transaksi menggunakan *natural join* dan perintah *where*.
14. Merelasikan tabel menu dengan tabel transaksi menggunakan *natural join* dan perintah *where*.

# BAB 4

## DATA MANIPULATION LANGUAGE 2

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL (DML) pada DBMS MySQL.
2. Mampu menggunakan operator pada saat memanipulasi data.

### 4.1 Operator *Precedence*, *Like*, *Not Like*, REGEXP

Pada bab ini akan dibahas terkait operator yang dapat digunakan pada saat melakukan manipulasi data dalam MySQL.

#### 4.1.1 Operator Precedence

Operator *precedence* adalah tingkatan hierarki dalam memproses serangkaian operator.

Tabel 4.1 Tabel Operator *Precedence*

| Tingkatan Hierarki | Jenis Operator |
|---|---|
| Paling Tinggi | BINARY |
| | NOT ! |
| | - (unary minus) |
| | * / % |
| | + - |
| | << >> |
| | & |
| | \| |
| | < <= = <=> != <> >= > IN IS LIKE REGEXP RLIKE |
| | BETWEEN |
| | AND && |
| Paling Rendah | OR \|\| |

Semakin ke atas letak operator, maka semakin tinggi tingkat hierarki operator tersebut. Begitu pula sebaliknya, semakin rendah letaknya maka akan semakin lemah hierarkinya. Untuk operator yang sama kuat, misal + dan - digabung dengan operator * / %, maka akan ditentukan hierarkinya tergantung dari letak mana yang paling kiri/paling awal ditemukan. Dan untungnya letak hierarki ini dapat diubah dengan bantuan tanda kurung "(" dan ")".

### 4.1.2 Operator LIKE, NOT LIKE

Operator LIKE, NOT LIKE banyak digunakan dalam operasi karakter.

1. **Operator LIKE**

Operator LIKE digunakan untuk mencari data yang "menyerupai" atau "hampir sama" dengan kriteria tertentu. Untuk mencari data *string*/teks. Simbol "%" digunakan untuk membantu pelaksanaan operator LIKE. Letak "%" sangat berpengaruh dalam menentukan kriteria. Contoh menampilkan data pelanggan yang namanya berawalan huruf "a": (perhatikan letak simbol persennya "%").

```
MariaDB [penerbangan]> select Nama_Depan, Tgl_Lahir
    -> from pelanggan
    -> where nama_depan LIKE "a%";
+------------+------------+
| Nama_Depan | Tgl_Lahir  |
+------------+------------+
| Agus       | 1999-08-08 |
| Ana        | 1995-09-12 |
| Arni       | 1993-02-26 |
+------------+------------+
3 rows in set (0.00 sec)
```

Gambar 4.1 Operator Like Menampilkan Nama Berawalan Huruf "A"

Baris pertama ‘select...’ digunakan untuk menunjukkan *field* mana saja yang ingin ditampilkan. Baris kedua ‘from...’ digunakan untuk tampilkan data pelanggan yang namanya berawalan huruf "r".

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan LIKE "r%";
+------------+------------+
| nama_depan | Tgl_lahir  |
+------------+------------+
| Rahman     | 1999-08-08 |
| Ranti      | 1993-09-10 |
| Rusdiani   | 1993-04-20 |
+------------+------------+
3 rows in set (0.00 sec)
```

Gambar 4.2 Operator Like Menampilkan Nama Berawalan Huruf "R"

Tampilkan data pelanggan yang namanya berakhiran huruf "i". Perhatikan letak penulisan tanda "%".

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan LIKE "%i";
+------------+------------+
| nama_depan | Tgl_lahir  |
+------------+------------+
| Arni       | 1993-02-26 |
| Ranti      | 1993-09-10 |
| Rusdiani   | 1993-04-20 |
| Subki      | 1993-10-15 |
+------------+------------+
4 rows in set (0.00 sec)
```

Gambar 4.3 Operator Like Menampilkan Nama Berakhiran Huruf "I"

Tampilkan data pelanggan yang namanya berakhiran "ni".

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan LIKE "%ni";
+------------+------------+
| nama_depan | Tgl_lahir  |
+------------+------------+
| Arni       | 1993-02-26 |
| Rusdiani   | 1993-04-20 |
+------------+------------+
2 rows in set (0.00 sec)
```

Gambar 4.4 Operator Like Menampilkan Nama Berakhiran Kata "Ni"

Agar operator LIKE dapat membedakan huruf besar dan kecil tambahkan kata BINARY setelah perintah LIKE (sehingga perintahnya menjadi LIKE BINARY). Pertama coba untuk melihat pelanggan yang menggunakan huruf ‘y’ kecil pada awal huruf nama belakangnya.

```
MariaDB [penerbangan]> select email_pel, nama_depan, nama_belakang
    -> from pelanggan
    -> where nama_belakang LIKE BINARY "y%";
Empty set (0.00 sec)
```

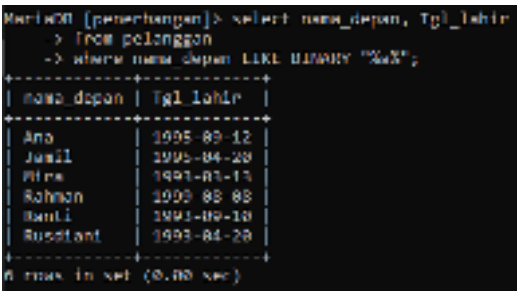
Gambar 4.5 Operator *Like* dengan *Binary*

Tidak ditemukan (‘empty set’) karena tidak ada pelanggan yang menggunakan huruf ‘y’ kecil pada awal huruf nama belakangnya. Lalu coba untuk melihat pelanggan dengan nama_depan yang memiliki awalan huruf kapital ‘A’.

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan LIKE BINARY "A%";
+------------+------------+
| nama_depan | Tgl_lahir  |
+------------+------------+
| Agus       | 1999-08-08 |
| Ana        | 1995-09-12 |
| Arni       | 1993-02-26 |
+------------+------------+
3 rows in set (0.00 sec)
```

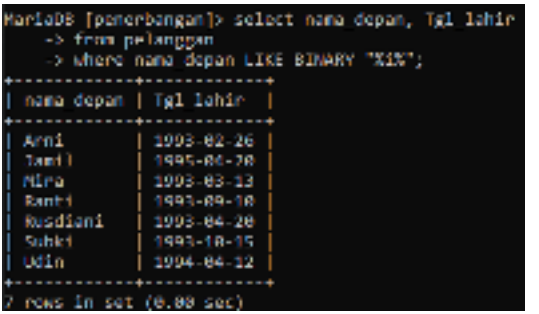
Gambar 4.6 Operator Like dengan Binary

Menampilkan nama pelanggan yang memiliki huruf "a" di mana pun letak huruf tersebut (bisa juga berupa kata). Perhatikan penulisan tanda "%".




Gambar 4.7 Operator Like dengan Binary

Atau memiliki huruf "i" pada namanya? (Bisa juga berupa kata)




Gambar 4.8 Operator Like dengan Binary

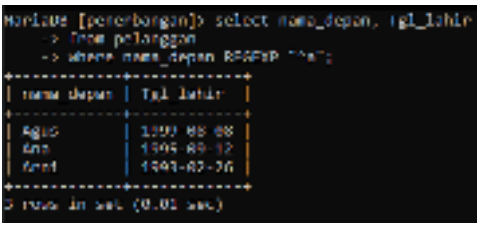
### 4.1.3 Operator REGEXP

Operator REGEXP (singkatan dari *REGular EXPressions*) merupakan bentuk lain dari operator LIKE, dengan fungsi yang lebih disempurnakan. Operator REGEXP biasanya ditemani juga dengan simbol-simbol tertentu dalam melaksanakan tugasnya, seperti Tabel 4.2.

Tabel 4.2 Tabel Operator REGEXP

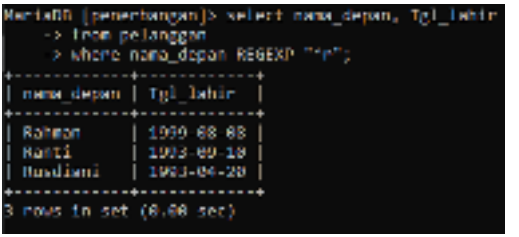
| Simbol | Keterangan |
|---|---|
| . | Satu tanda titik (.) adalah untuk mewakili satu karakter |
| [?] | Mewakili beberapa karakter atau *range* yang ditentukan. |
| ^ | Menampilkan letak awal dari sebuah kriteria yang ditentukan |
| $ | Menampilkan letak akhir dari sebuah kriteria yang telah ditentukan |

Contohnya menampilkan nama pelanggan yang memiliki awal huruf 'a'.



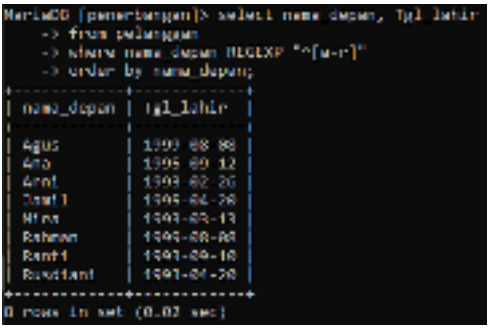

Gambar 4.9 Operator REGEXP

Tampilkan data pelanggan yang namanya berawalan huruf "r".




Gambar 4.10 Operator REGEXP

Tampilkan nama pelanggan yang memiliki awal huruf 'a' sampai dengan huruf 'r'.




Gambar 4.11 Operator REGEXP

Tampilkan data pelanggan yang namanya berakhiran huruf "i".

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan REGEXP "i$"
    -> order by nama_depan;
+-------------+------------+
| nama_depan | Tgl_lahir  |
+-------------+------------+
| Arni        | 1993-02-26 |
| Ranti       | 1993-09-10 |
| Rusdiani    | 1993-04-20 |
| Subki       | 1993-10-15 |
+-------------+------------+
4 rows in set (0.00 sec)
```

Gambar 4.12 Operator REGEXP

Tampilkan data pelanggan yang namanya berakhiran "ni".

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan REGEXP "ni$"
    -> order by nama_depan;
+-------------+------------+
| nama_depan | Tgl_lahir  |
+-------------+------------+
| Arni        | 1993-02-26 |
| Rusdiani    | 1993-04-20 |
+-------------+------------+
2 rows in set (0.00 sec)
```

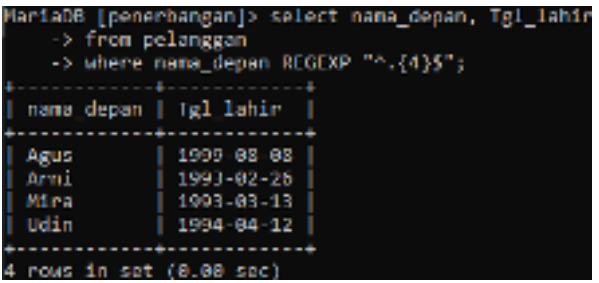
Gambar 4.13 Operator REGEXP

Tampilkan nama pelanggan yang panjangnya 4 karakter.

```
MariaDB [penerbangan]> select nama_depan, Tgl_lahir
    -> from pelanggan
    -> where nama_depan REGEXP "^....$";
+-------------+------------+
| nama_depan | Tgl_lahir  |
+-------------+------------+
| Agus        | 1999-08-08 |
| Arni        | 1993-02-26 |
| Mira        | 1993-03-13 |
| Udin        | 1994-04-12 |
+-------------+------------+
4 rows in set (0.00 sec)
```

Gambar 4.14 Operator REGEXP (1)

Perintah di atas bisa juga ditulis seperti Gambar 4.15 berikut ini.




Gambar 4.15 Operator REGEXP (2)

## 4.2 Fungsi Statistik Dasar

Fungsi statistik dasar pada MySQL digunakan untuk berbagai kebutuhan yang dikenakan pada kolom yang mempunyai jenis numerik atau angka. Berikut ini adalah fungsi statistik dasar yang dapat digunakan untuk menampilkan data pada *database* MySQL.

**1. AVG (ekspresi)**

Fungsi AVG () atau *average* dipergunakan untuk mencari nilai rata-rata dalam suatu kolom pada tabel. Pernyataan dalam fungsi AVG() umumnya adalah nama kolom dengan tipe data numerik atau angka. Contohnya sebagai berikut.

```
MariaDB [penerbangan1]> select avg(harga_total) as rata_rata
    -> from pemesanan;
+---------------+
| rata_rata     |
+---------------+
| 1252500.0000  |
+---------------+
1 row in set (0.005 sec)
```

**2. COUNT(x)**

Fungsi COUNT() dipergunakan untuk menghitung jumlah baris dari sebuah kolom dalam suatu tabel. Huruf (x) adalah nama kolom dari tabel yang ingin dicari jumlah baris datanya. Contohnya sebagai berikut.

```
MariaDB [penerbangan1]> select count(harga total) as jumlah pemesanan
    -> from pemesanan;
+------------------+
| jumlah pemesanan |
+------------------+
|               18 |
+------------------+
1 row in set (0.001 sec)
```

**3. MAX(ekspresi)**

Fungsi MAX dipergunakan untuk menghitung nilai terbesar dari suatu kolom pada suatu tabel, dengan persyaratan yaitu kolom yang dicari nilai terbesarnya harus memiliki tipe data numerik atau angka. Contohnya sebagai berikut.

```
MariaDB [penerbangan1]> select max(harga total) as max harga
    -> from pemesanan;
+-----------+
| max_harga |
+-----------+
|   1410000 |
+-----------+
1 row in set (0.001 sec)
```

**4. MIN(ekspresi)**

Fungsi MIN () merupakan kebalikan dari fungsi MAX (). Fungsi ini dipergunakan untuk mencari nilai terkecil dari suatu kolom dalam suatu tabel. Contohnya sebagai berikut.

```
MariaDB [penerbangan1]> select min(harga total) as min harga
    -> from pemesanan;
+-----------+
| min harga |
+-----------+
|   1000000 |
+-----------+
1 row in set (0.001 sec)
```

**5. STD(ekspresi) dan STDDEV(ekspresi)**

Fungsi ini dapat kita gunakan untuk mendapatkan standar deviasi dari suatu kolom dalam suatu tabel. Ekspresi biasanya menyatakan kolom dalam suatu tabel dengan tipe data numerik. Contohnya sebagai berikut.

```
MariaDB [penerbangan1]> select std(harga total) as standar deviasi
    -> from pemesanan;
+-----------------+
| standar deviasi |
+-----------------+
|     127695.9279 |
+-----------------+
1 row in set (0.002 sec)
```

**6. SUM(ekspresi)**

Fungsi SUM () dipergunakan untuk mendapatkan total nilai dari suatu kolom dalam suatu tabel. Contohnya sebagai berikut.

```
MariaDB [penerbangan1]> select sum(harga total) as total harga
    -> from pemesanan;
+-------------+
| total_harga |
+-------------+
|    12125000 |
+-------------+
1 row in set (0.001 sec)
```

## 4.3 Soal Latihan

Berdasarkan *database* caffe tampilkan data berikut ini.

1. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan, dengan nama berawal huruf a.
2. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan, dengan nama berakhiran huruf i.
3. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan, dengan nama berakhiran ‘ani’.
4. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan, yang memiliki ‘li’ di antara namanya.
5. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan, yang berawalan huruf M.
6. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan yang memiliki nama berakhiran huruf a.
7. Menampilkan id_pelanggan dan nama_pelanggan dari tabel pelanggan, yang nama pelanggan tersebut memiliki panjang karakter 15.

# BAB 5

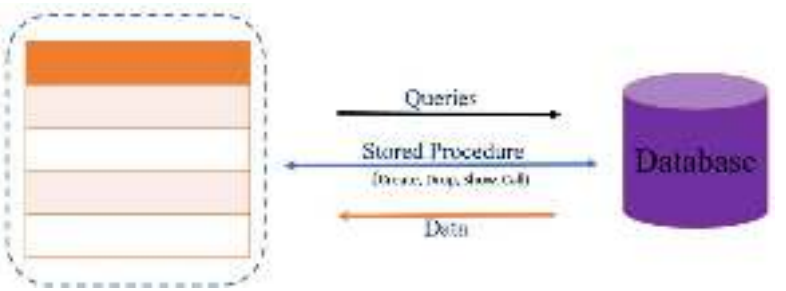
## STORED PROCEDURE

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL *Stored Procedure* pada DBMS MySQL.
2. Mampu mengimplementasikan pembuatan *stored procedure* yang diperlukan pada DBMS MySQL.

### 5.1 Stored procedure

*Stored procedure* adalah sebuah kelompok kode SQL yang di simpan di katalog *database* dan dapat di panggil kemudian oleh program, *trigger* atau bahkan *stored procedure*. Sebuah *Stored procedure* yang memanggil diri nya sendiri di sebut rekursif *stored procedure*. Fungsi dari *stored procedure* ini adalah untuk memanggil suatu data dalam tabel. Dengan menggunakan *stored procedure* SQL tidak akan melakukan *loading* semua tabel yang terrelasi, tetapi langsung melakukan *filtering* berdasarkan *query* yang kita maksud. *Stored procedure* menyimpan *statement-statement* SQL dalam sebuah berkas yang disimpan di *database server*, sehingga dari sisi performa eksekusi, kegunaan jaringan, dan keamanan, *stored procedure* banyak dipakai sebagai solusi akses data.




Gambar 5.1 Ilustrasi *Stored Procedure*

Proses *Query* meliputi; Pengecekan *Syntax*, Pemilihan *execution plan* yang paling optimal, dan Eksekusi *query*. *Query* yang ada di SP sudah di-*compile* terlebih dahulu, jadi ada 1 step yang di-*skip* pada SP. *Compile* maksudnya adalah pemilihan mana *execution plan* yang paling optimal.

### 5.1.1 Format Penulisan Stored procedure

Ada 4 proses yang dapat terjadi dalam *stored procedure*, yaitu sebagai berikut.

1. Membuat Prosedur (CREATE)
   Contoh: `CREATE PROCEDURE sp_name ([proc_parameter [,…]]) [characteristic..] routine_body`
2. Memanggil Prosedur (*CALL*)
   Contoh: `CALL sp_name`
3. Menampilkan Prosedur (SHOW)
   Contoh: `SHOW {PROCEDURE|FUNCTION} status;`
4. Menghapus Prosedur (DROP)
   Contoh: `DROP {PROCEDURE|FUNCTION} [IF EXIST] sp_name`
   Format penulisan perintah *stored procedure* adalah sebagai berikut.

```
delimeter //
create procedure NamaTabel(in NamaField_TipeData_LebarData,
in…)
begin
select * from NamaDatabase
end //
delimeter;
```

Penggunaan perintah *delimiter* digunakan untuk memberi tahu *shell* myql soal delimiter (akhir *statement*) yang digunakan, secara *default* delimiter yang menggunakan tanda titik koma (;) jadi bila ada tanda ; MySQL akan mengartikan akhir dari *statement*, pada contoh di atas delimeter yang digunakan // jadi akhir *statement*nya adalah //, hal ini dilakukan karena di dalam badan *procedure* (antara *begin .... end)* menggunakan titik koma (;) sebagai akhir suatu *statement*.

Dalam membuat *stored procedure* kita dapat membuat variabel yang digunakan untuk menyimpan *procedure* ke penyimpanan hasil dengan segera, yaitu dengan mendeklarasikan variabel dengan perintah berikut:

```
DECLARE nama variabel tipe data (ukuran) DEFAULT nilai default;
```

Contoh :

```
DECLARE total_sales INT DEFAULT 0
```

Selanjutnya untuk memberikan nilai ke variabel dapat dilakukan dengan beberapa cara yaitu sebagai berikut.

1. Menggunakan perintah **SET**
   Contoh :
   ```
   DECLARE total_bayar INT DEFAULT 0
   SET total_bayar =0
   ```
2. Menggunakan perintah **SELECT … INTO**
   Contoh:
   ```
   DECLARE total_bayar INT DEFAULT 0
   SELECT COUNT(*) INTO total_bayar FROM pemesanan
   ```

Dalam sebuah *stored procedure* suatu variabel hanya berlaku di dalam ruang lingkup nya masing-masing, yaitu di antara BEGIN dan END, dan sebuah variabel yang di awali dengan tanda @, disebut dengan variabel *session*, yang tetap ada hingga *session* berakhir.

### 5.1.2 Parameter dalam *Stored procedure*

Parameter dalam *Stored procedure* terdiri dari 3 bentuk, yaitu sebagai berikut.

1. IN
   *Mode Default*, dapat digunakan di dalam sebuah *stored procedure*, namun *stored procedure* tidak dapat mengubah nilainya.
2. OUT
   Parameter ini dapat di rubah oleh sebuah *stored procedure* yang dilewatinya.
3. INOUT
   Dapat melewati *stored procedure* dan mendapatkan kembali nilainya yang berbeda dari program yang memanggil.
   *Syntax* untuk mendefinisikan sebuah parameter sebagai berikut.
   ```
   MODE nama_parameter tipe_parameter (ukuran parameter);
   ```

### 5.1.3 Keuntungan kegunaan *Stored procedure*

Keuntungan dari penggunaan *stored procedure* adalah sebagai berikut.

1. Dapat meningkatkan *performance* aplikasi, karena *Stored procedure* dapat di simpan dan di *compile* di katalog *database* sehingga dapat di proses lebih cepat di bandingkan SQL yang tidak di-*compile* dari kode aplikasi.
2. Mengurangi traffic antara aplikasi dan *database* server. Aplikasi hanya mengirim nama *stored procedure* untuk mengeksekusi SQL.
3. Dapat digunakan kembali, penggunaan *Stored procedure* dapat di akses hak nya oleh aplikasi melalui *database* administrator.

### 5.1.4 Kekurangan kegunaan *Stored procedure*

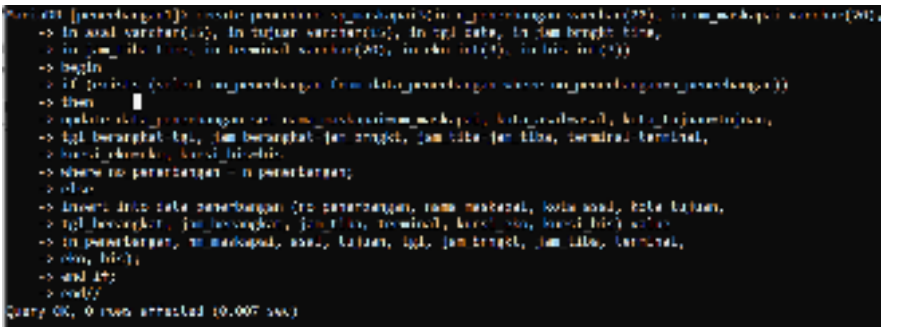
Kekurangan dari penggunaan *stored procedure* adalah sebagai berikut.

1. Dapat mengakibatkan *database* server membutuhkan *memory* dan prosessor lebih tinggi.
2. *Stored procedure* hanya berisi SQL deklaratif, sehingga sangat sulit untuk menulis sebuah *procedure* dengan kompleksitas logika.
3. *Stored procedure* tidak dapat dijalankan di berbagai RDBMS, termasuk MySQL.
4. Membutuhkan keahlian khusus untuk menulis dan memelihara *stored procedure*.

## 5.2 Uji Coba

Membuat *stored procedure* tabel data_penerbangan, jika ingin menambah data di tabel data_penerbangan tetapi data tersebut sudah ada di *table* data_penerbangan maka yang terjadi adalah proses *Update* pada kota_tujuan dan jam_tiba, tetapi jika data yang ingin ditambah tidak ada di *table* Barang maka akan dilakukan proses *Insert* seluruh data pada tabel data_penerbangan.

1. Buat *stored procedure* dengan studi kasus seperti di atas.



Gambar 5.2 Membuat *Stored procedure* sp_maskapai

a. DELIMITER adalah untuk memberi tahu kepada MySQL soal delimiter yang digunakan, secara *default* menggunakan ; jadi bila ada tanda ; MySQL akan mengartikan akhir dari *statement*, pada contoh di atas delimeter yang digunakan // jadi akhir *statement*nya adalah //.
b. CREATE PROCEDURE adalah *header* untuk membuat *procedure.*
c. BEGIN untuk memulai instruksi dari *procedure.*
d. If, Then, dan Else adalah untuk memberikan perintah-perintah yang akan berjalan dalam *procedure* tersebut.
e. END If adalah untuk menghentikan proses tindakan *procedure* yang dibuat.
f. END adalah untuk mengakhiri proses.

2. Tampilkan *table* data_penerbangan sebelum proses *call*, agar dapat mengetahui bagaimana perubahan *table* setelah proses *call* dilakukan dengan perintah SELECT * FROM data_penerbangan//.

Gambar 5.3 Menampilkan tabel data_penerbangan

3. Lakukan proses *call* dengan data yang belum ada di *table* data_penerbangan seperti berikut.

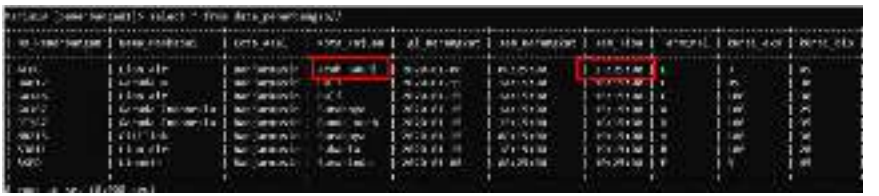
```
CALL sp_maskapai3 ('SSPD', 'LionAir', 'Banjarmasin',
'Samarinda', '2020-01-06', '08:25:00.000000',
'10:25:00.000000', 'E', '3', '45');
```

Gambar 5.4 Proses *Call* sp_maskapai

Terjadi pertambahan data dalam *table* data_penerbangan karena pemanggilan data yang dilakukan oleh "*call*" sebelumnya belum terdapat pada tabel data_penerbangan.

4. Lakukan lagi proses *call* dengan data yang sudah ada di *table* data_penerbangan untuk memperbarui data yang ada seperti berikut.

```
CALL sp_maskapai3 ('ACDG', 'Lion Air', 'Banjarmasin',
'Arab Saudi', '2020-01-06', '08:25:00.000000',
'23:25:00.000000', 'E', '3', '45');
```

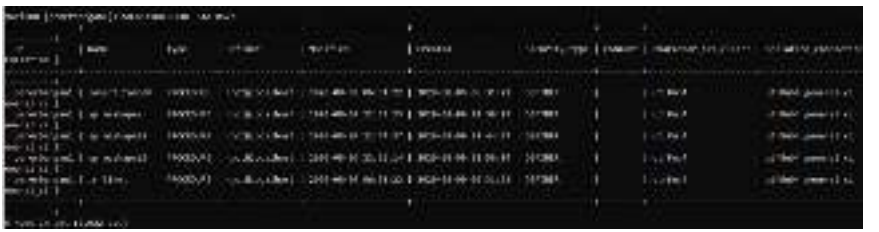


Gambar 5.5 Proses *Call* sp_maskapai

Terjadi pembaharuan data pada *field* kota_tujuan dan jam_tiba karena data yang dimasukkan sudah tersimpan dalam tabel data_penerbangan sehingga hanya terjadi *update* atau pembaharuan pada *field* tertentu / yang diperbaharui saja.

5. Jika ingin melihat status dari *stored procedure* dapat dilakukan dengan perintah berikut.

```
SHOW PROCEDURE STATUS;
```



Gambar 5.6 Menampilkan Status *Stored procedure*

Maka terlihat hasil dari *stored procedure* yang telah dibuat.

6. Jika Ingin menghapus *stored procedure* dapat dilakukan dengan perintah berikut.

```
DROP + PROCEDURE + sp_name;
```


Gambar 5.7 Drop Procedure

Maka prosedur sp_maskapai telah terhapus, untuk memastikannya maka dapat menggunakan perintah yang dapat menampilkan status prosedur.

## 5.3 Soal Latihan

Buatlah *procedure* untuk mengubah daftar menu pada dbcaffe, ketika proses *call* dijalankan maka menu ditambahkan (proses *insert*) kedalam daftar menu jika tidak ada kd_menu yang sama tapi jika kd_menu yang diinputkan ada yang sama dengan kd_menu di daftar menu maka menu akan diedit (proses *update*) sesuai dengan yang diinputkan.

1. Tampilkan data awal sebelum proses pemanggilan *call* pada Tabel menu.
2. Selanjutnya lakukan perintah *call* dan buat perintah menjalankan instruksi *insert* dan *update*.

# BAB 6

# FUNCTION

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL *function* pada DBMS MySQL.
2. Mampu mengimplementasikan pembuatan *function* yang diperlukan pada DBMS MySQL.

## 6.1 Function

*Function* adalah sebuah kumpulan *statement* yang akan mengembalikan sebuah nilai balik pada pemanggilnya. Nilai yang dihasilkan *function* harus ditampung kedalam sebuah variabel. Fungsi merupakan suatu bagian dari program yang digunakan untuk mengerjakan suatu tugas tertentu yang menghasilkan suatu nilai untuk dikembalikan ke program pemanggil dan letaknya dipisahkan dari bagian program yang menggunakannya.

### 6.1.1 Perbedaan *Function* dan Procedure

*Function* akan mengembalikan suatu nilai pada pemanggilnya, sedangkan *procedure* tidak akan mengembalikan nilai apapun pada fungsi pemanggilnya. Fungsi (*Function*) adalah suatu bagian dari program yang dipergunakan untuk mengerjakan suatu tugas tertentu yang menghasilkan suatu nilai untuk dikembalikan ke program pemanggil dan letaknya dipisahkan dari bagian program yang menggunakannya.

### 6.1.2 Kegunaan *Function*

Kegunaan dari *function* adalah sebagai berikut.

1. Menghindari pengulangan, tujuannya untuk menghindari penulisan bagian kode program berulang-ulang.

2. Penataan program, program yang besar dan kompleks dibagi-bagi menjadi aktivitas yang berbeda dan ditempatkan dalam subrutine yang terpisah, sehingga setiap aktivitas bisa ditulis dan diperiksa secara mandiri.
3. Kemandirian, mempunyai variabel "*private*" yaitu variabel yang tidak bisa diakses program pemanggil atau subrutin lain.

### 6.1.3 Alasan membuat *function* di MySQL

Alasan membuat *function* di MySQL adalah sebagai berikut.

1. Penggunaan Menjadi Lebih Mudah

Dengan kita membuat *function* berarti kita telah meringkas beberapa perintah SQL menjadi satu perintah saja, sehingga dalam penggunaanya menjadi mudah. Mudah di sini berarti *user* atau pengguna tidak perlu mengetahui isi dari fungsi tersebut. Contohnya seperti kita menggunakan fungsi *string* yang telah disediakan oleh MySQL, kita cukup mengetahui cara penggunaannya saja tanpa harus mengetahui perintah SQL yang berada di dalam fungsi *string* tersebut.

2. Keamanan Lebih Terjaga

Dengan membuat *function* kita dapat memberi hak akses kepada masing-masing *user*. Kita cukup memberikan hak akses untuk menjalankan *function* saja kepada *user*, tanpa harus memberikan hak akses untuk memanipulasi Tabel aslinya secara langsung. Selain itu, *user* juga tidak akan mengetahui perintah yang berada di dalam *function* tersebut, karena telah disembunyikan.

### 6.1.4 Bentuk umum membuat *function*

Bentuk umum dalam membuat *function* adalah sebagai berikut.

```
CREATE FUNCTION ([parameter,[…]])
RETURNS tipe_data_hasil_kembalian
RETURN isi_fungsi
```

Keterangan dari bentuk umum di atas adalah sebagai berikut.

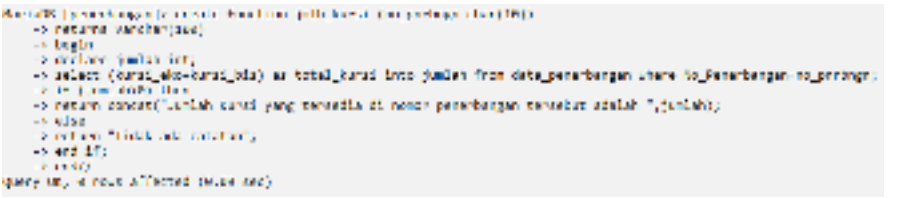
1. DELIMITER untuk memberi tahu kepada myql soal delimiter yang digunakan, secara *default* menggunakan ;.
2. CREATE *FUNCTION header* untuk membuat *function*.
3. RETURNS adalah untuk menentukan tipe data yang di *return* oleh *function.*
4. DETERMINISTIC/ NOT DETERMINISTIC untuk menentukan yang bisa menggunakan *function* ini adalah *user* pembuatnya saja (*determinisric*) atau siapa saja (*not determinisric*).
5. BEGIN END adalah *body* dari *function* jadi semua SQL nya di tulis disini.

## 6.2 Uji Coba

Ketika ingin menggunakan jasa penerbangan para calon penumpang maskapai biasanya mencek terlebih dahulu apakah masih ada slot kursi di salah satu maskapai yang digunakan, oleh karena itu pada BAB ini kita akan menggunakan yang namanya *function* (fungsi) untuk memudahkan para calon penumpang yang ingin mengecek slot kursi pesawat di salah satu maskapai yang diinginkan dengan menggunakan *FUNCTION*.

Contoh kasusnya adalah menampilkan jumlah kursi penerbangan yang tersedia pada setiap nomor penerbangan di tabel data_penerbangan.

1. Buat *function.*

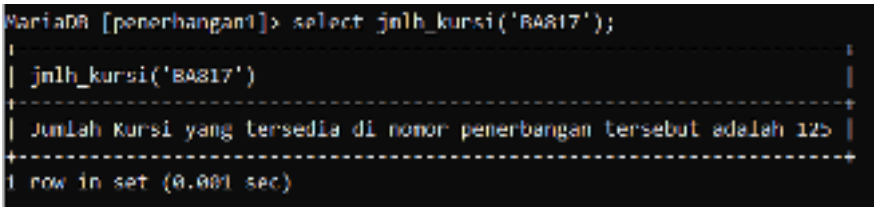


Gambar 6.1 Membuat *Function* jmlh_kursi

2. Tampilkan *table* data_penerbangan untuk perbandingan.

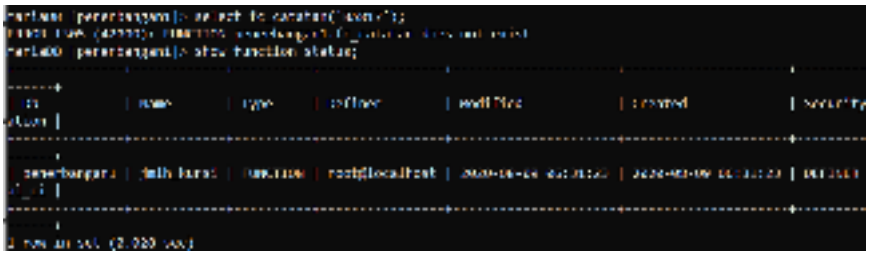
Gambar 6.2 Menampilkan Data Tabel data_penerbangan

3. Untuk menampilkan hasil *function*, lakukan proses *select*, tampilkan jumlah kursi yang ada pada no penerbangan “BA817”.




Gambar 6.3 Menampilkan Hasil *Function* jmlh_kursi

4. Untuk melihat status dari *function.*



Gambar 6.4 Menampilkan Status *Function*

## 6.3 Soal Latihan

1. Buatlah *function* fc_catatan di *database* dbcaffe dan tampilkan tabel transaksi.
2. Tampilkan catatan untuk harga 20000.
3. Tampilkan catatan untuk harga 35000.
4. Tampilkan status dari *function.*

# BAB 7

# TRIGGER

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL *Trigger* pada DBMS MySQL.
2. Mampu mengimplementasikan pembuatan *trigger* pada DBMS MySQL.

## 7.1 Trigger

*Trigger* merupakan *stored procedure* yang dijalankan secara automatis saat *user* melakukan modifikasi data pada tabel. Modifikasi data yang dilakukan pada tabel yaitu berupa perintah INSERT, UPDATE, dan DELETE. INSERT , UPDATE dan DELETE bisa digabung jadi satu *trigger* yang dinamakan Multiple *Trigger*. Event tersebut meliputi operasi yang biasa dilakukan dalam mengolah *database*, seperti berikut.

1. DML (*Data Manipulation Language*) yang meliputi DELETE, INSERT atau UPDATE.
2. DDL (*Data Definition Language*) yang meliputi CREATE, ALTER atau DROP.
3. Operasi *database* lainnya, seperti SERVERERROR, LOGON, LOGOFF, STARTUP atau SHUTDOWN).

Keterangan dari bentuk perintah umum dalam membuat *trigger*, yaitu sebagai berikut.

1. nama_trigger adalah nama *trigger* yang dibuat sesuai dengan karakteristik penamaan dalam MySQL.
2. [BEFORE|AFTER] menunjukkan waktu untuk mengeksekusi *trigger* secara otomatis, apakah sebelum atau sesudah perubahan pada *row* data *table*. Jadi pilihannya adalah AFTER atau BEFORE.

3. [INSERT | UPDATE | DELETE] digunakan untuk menentukan *event* yang menyebabkan terjadinya *trigger*, pilihan *event* tersebut terdiri dari INSER, UPDATE dan DELETE.
4. nama_*table* menunjukkan *table* yang akan dilakukan *trigger* di dalamnya.
5. trigger_body menunjukkan *statement* perintah dalam MySQL yang akan otomatis dijalankan jika *event* sedang aktif.

### 7.1.1 Jenis Trigger

*Trigger* terdiri dari beberapa jenis, yaitu sebagai berikut.

1. Application *trigger*, diaktifkan pada saat terjadi *event* yang berhubungan dengan sebuah aplikasi.
2. *Database trigger*, diaktifkan pada saat terjadi *event* yang berhubungan dengan data (seperti operasi DML) atau *event* yang berhubungan dengan sistem (semisal *logon* atau *shutdown*) yang terjadi pada sebuah skema atau *database*.

### 7.1.2 Sintaks Penulisan

Sintak penulisan dari *database trigger*, berisi komponen berikut.

1. *Trigger timing*:
   a. BEFORE : *trigger* dijalankan sebelum DML *event* pada tabel.
   b. AFTER : *trigger* dijalankan setelah DML *event* pada tabel.
   c. INSTEAD OF : *trigger* dijalankan pada sebuah *view*.
2. *Trigger event***:** INSERT, UPDATE atau DELETE
3. Nama tabel**:** nama tabel atau *view* yang berhubungan dengan *trigger*.
4. Tipe *trigger:* baris atau pernyataan (*statement*)**.**
5. Klausa WHEN: untuk kondisi pembatasan.
6. *Trigger body*: bagian prosedur yang dituliskan pada *trigger.*

### 7.1.3 Tipe Trigger

Tipe *trigger* ada 2 macam, yaitu sebagai berikut.

1. *Statement*, *trigger* dijalankan sekali saja pada saat terjadi sebuah *event* dan tidak mempengaruhi satupun baris dari *event* yang terjadi.

2. *Row*, *trigger* dijalankan pada setiap baris yang dipengaruhi oleh terjadinya sebuah *event*. Row *trigger* tidak akan jalan jika *event* dari *trigger* tidak terpengaruh.

Berikut sintak atau cara penulisan pembuatan DML *Statement trigger.*

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON table_name
trigger_body
```

Berikut contoh pembuatan DML *Statement trigger*.

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees
BEGIN
IF   (TO_CHAR(SYSDATE,'DY')   IN   ('SAT','SUN'))   OR
(TO_CHAR(SYSDATE,'HH24:MI')  NOT  BETWEEN  '08:00'  AND
'18:00')
THEN  RAISE_APPLICATION_ERROR  (-20500,'Penyisipan  data
pada table
EMPLOYEES hanya diperbolehkan selama jam kerja');
END IF;
END//
```

## 7.2 Uji Coba

Contoh kasusnya ialah *trigger* tiket, mengurangi jumlah kursi_eko di tabel data_penerbangan ketika *insert* (menambahkan / memasukkan) data baru jika ber*field* “Ekonomi” di *table* tiket, dan mengurangi jumlah kursi_bis di *table* data_penerbangan ketika *insert* (menambahkan/ memasukkan) data baru jika ber*field* “ Bisnis” pada *table* tiket.

1. Buat *syntax trigger.*

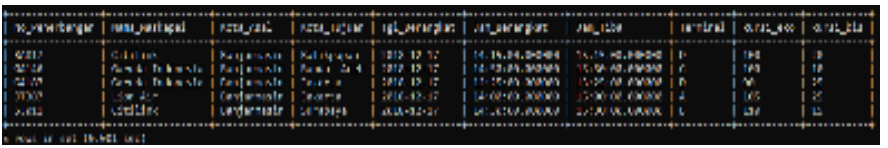
```
MariaDB [Penerbangan4]> delimiter //
MariaDB [Penerbangan4]> create trigger insertTiket
    -> after insert on tiket
    -> for each row
    -> begin
    -> if new.kelas='Ekonomi' then
    -> update data_penerbangan set kursi_eko=kursi_eko-new.ekonomi
    -> where No_Penerbangan=new.no_penerbangan;
    -> end if;
    -> end//
Query OK, 0 rows affected (0.178 sec)
```

Gambar 7.1 Membuat *Trigger* InsertTiket

**Mengapa menggunakan delimiter?** Karena MySQL secara *default* menganggap titik koma (;) sebagai delimiter/pembatas akhir dari suatu perintah/*statement*. Akhirnya pembuatan objek yang memiliki beberapa *statement* tidak akan berjalan sempurna karena "berhenti di tengah jalan".

Solusi terhadap masalah di atas adalah menggunakan delimiter selain tanda titik koma (;) misalkan dengan garis pipa (|). Penggunaannya sangat sederhana, sebelum mendefinisikan objek tersebut kita gunakan *statement* "DELIMITER" diikuti tanda pemisah baru. Setelah di akhir pendefinisian kita kembalikan delimiter lagi kepada tanda titik koma.

2. Tampilkan *table* data_penerbangan sebelum proses *insert*, agar dapat mengetahui perubahan pada data_penerbangan.



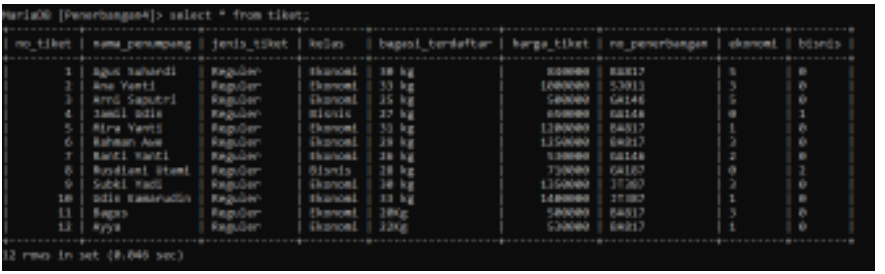
Gambar 7.2 Menampilkan Data Tabel data_penerbangan

3. Lakukan proses *insert* untuk kejadian ‘kurang’ ber*field* “Ekonomi” untuk membuat variasi data dan tampilkan data Penerbangan dan tiket untuk melihat hasil perubahan.

```
MariaDB [Penerbangan4]> insert into tiket values
    -> ('11', 'Bagus', 'Reguler', 'Ekonomi', '20Kg', '500000', 'BA817', '3', '0'),
    -> ('12', 'Ayya', 'Reguler', 'Ekonomi', '22Kg', '530000', 'BA817', '1', '0');
Query OK, 2 rows affected (0.280 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Gambar 7.3 Menambahkan data pada Tabel Tiket (Ekonomi)

Selanjutnya adalah tampilkan perubahan data yang terjadi di tabel tiket.




Gambar 7.4 Menampilkan Data Tabel Tiket

Selanjutnya perhatikan perubahan yang terjadi pada tabel data penerbangan.

Gambar 7.5 Menampilkan Data Tabel data_penerbangan

4. Membuat *trigger* ber*field* bisnis.

```
MariaDB [penerbangan4]> delimiter //
MariaDB [penerbangan4]> create trigger insertTiketBis
    -> after insert on tiket
    -> for each row
    -> begin
    -> if new.kelas='Bisnis' then
    -> update data_penerbangan set kursi_bis=kursi_bis-new.bisnis
    -> where No_penerbangan=new.no_penerbangan;
    -> end if;
    -> end //
Query OK, 0 rows affected (0.124 sec)
```

Gambar 7.6 Membuat *Trigger* Field Bisnis

Lakukan proses *insert* untuk kejadian **'kurang'** ber*field* **"Bisnis"** untuk membuat variasi data dan tampilkan data Penerbangan dan tiket untuk melihat hasil perubahan.

```
MariaDB [penerbangan4]> insert into tiket values
    -> ("15", "Zainullah", "Regular", "Bisnis", "25 Kg", "800000", "37387", "0", "1")
    -> ;
Query OK, 1 row affected (0.129 sec)
```

Gambar 7.7 Menambahkan data pada tabel tiket

```
MariaDB [penerbangan4]> select * from tiket;
+-----------+-----------------+-------------+---------+-----------------+-------------+----------------+----------+--------+
| no_tiket  | nama_penumpang  | jenis_tiket | kelas   | bagasi_terdaftar | harga_tiket | no_penerbangan | ekonomi  | bisnis |
+-----------+-----------------+-------------+---------+-----------------+-------------+----------------+----------+--------+
|         1 | Agus Saherdi    | Regular     | Ekonomi | 30 kg           |      840000 | GA017          |        5 |      0 |
|         2 | Ana Yanti       | Regular     | Ekonomi | 33 kg           |     1000000 | SJ011          |        2 |      0 |
|         3 | Anni Saputri    | Regular     | Ekonomi | 25 kg           |      980000 | GA144          |        1 |      0 |
|         4 | Jamil Udin      | Regular     | Bisnis  | 27 kg           |      650000 | GA144          |        0 |      1 |
|         5 | Mira Yanti      | Regular     | Ekonomi | 33 kg           |     1300000 | GA017          |        1 |      0 |
|         6 | Rahman Ase      | Regular     | Ekonomi | 28 kg           |     1150000 | GA017          |        1 |      0 |
|         7 | Rantl Yanti     | Regular     | Ekonomi | 28 kg           |      530000 | GA144          |        2 |      0 |
|         8 | Ruslianl staml  | Regular     | Bisnis  | 28 kg           |      750000 | GA107          |        0 |      1 |
|         9 | Sukri Yadi      | Regular     | Ekonomi | 30 kg           |     1150000 | JT107          |        1 |      0 |
|        10 | Udin Kamaruddin | Regular     | Ekonomi | 33 kg           |     1400000 | JT107          |        1 |      0 |
|        11 | Bagas           | Regular     | Ekonomi | 20Kg            |      500000 | GA017          |        2 |      0 |
|        12 | Ayya            | Regular     | Ekonomi | 22Kg            |      530000 | GA017          |        1 |      0 |
|        13 | Zainullah       | Regular     | Bisnis  | 25 Kg           |      800000 | JT107          |        0 |      1 |
+-----------+-----------------+-------------+---------+-----------------+-------------+----------------+----------+--------+
13 rows in set (0.000 sec)
```

Gambar 7.8 Menampilkan Data Tabel Tiket

Gambar 7.9 Menampilkan Data Tabel data_penerbangan

5. Menampilkan status *trigger.*

Gambar 7.10 Menampilkan Status *Trigger*

## 7.3 Soal Latihan

Lakukan uji coba di atas pada *database* dbcaffe sebelumnya. Buatlah sebuah *trigger* delete yang menghubungkan antara tabel menu dengan tabel log_menu. Jika ada salah satu dari setiap data pada tabel menu yang dihapus, maka data tersebut akan muncul di tabel log_menu. Isi *field* pada log_menu sendiri adalah semua *field* yang ada pada tabel menu dan satu *field* baru yaitu tanggal_hapus sebagai keterangan kapan data tersebut dihapus.

1. Membuat *trigger* tg_logmenu untuk tabel menu. *Trigger* ini menyimpan setiap baris data yang dihapus pada tabel menu dan akan ditampilkan pada tabel log_menu dengan keterangan tanggal data tersebut dihapus.
2. Melihat isi data awal pada tabel menu sebelum menghapus salah satu data.

3. Menghapus data pada tabel menu. Yaitu penghapusan menu pasta dengan kd_menu M21.
4. Melihat perubahan isi data pada tabel menu. kd_menu M21 sudah dihapus.
5. Melihat perubahan isi data pada tabel log_menu. kd_menu M21 yang sudah dihapus tadi akan muncul ditabel ini dengan keterangan tanggal kapan data itu dihapus.
6. Melihat status dari *trigger*.

# BAB 8

## VIEW/INDEX

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL *view* dan *index* pada DBMS MySQL.
2. Mampu mengimplementasikan pembuatan *view* dan *index* pada DBMS MySQL.

### 8.1 Perintah *View*

*View* adalah perintah *query* yang disimpan pada *database* dengan suatu nama tertentu, sehingga bisa digunakan setiap saat untuk melihat data tanpa menuliskan ulang *query* tersebut.

**1. Kegunaan *VIEW* adalah sebagai berikut.**

a. Menyembunyikan Kolom atau Baris Fungsi *built-in layer*.
b. Menampilkan hasil dari perhitungan.
c. Menyediakan level isolasi antara data tabel dan *View* data pengguna.
d. Memberikan *trigger* berbeda pada *view* yang berbeda dari tabel yang sama.
e. Memberikan proses *permission* yang berbeda untuk *view* yang berbeda dari tabel yang sama.
f. Menyembunyikan Sintak SQL yang rumit.

**2. Keuntungan *VIEW* adalah sebagai berikut.**

a. Membatasi akses data.
b. Menyediakan data yang independen.
c. Menampilkan *view* yang berbeda-beda dengan data yang sama.
d. Memudahkan *query* yang kompleks.

Cara membuat *View* yaitu sebagai berikut.

*CREATE VIEW view_name AS*
*SELECT column_name(s)*
*FROM table_name*
*WHERE condition*
Untuk menampilkan *Query* yang telah dibuat :
*SELECT * FROM nama_view*
Untuk menghapus *Query view* yang kita buat :
*DROP nama_view*

## 8.2 Uji Coba *View*

Studi kasus yang terjadi pada *database* penerbangan yaitu menampilkan data pemesanan tiket pelanggan (pemesanan.no_pemesanan, tiket.no_tiket, pelanggan.email_pel, data_penerbangan.no_penerbangan) dengan 4 tabel yang digabungkan dan menampilkan data cek jadwal (pemesanan.no_pemesanan, pelanggan.email_pel, data_penerbangan.no_penerbangan) dengan 3 tabel yang digabungkan.

1. Membuat *View* Pemesanan Tiket Pelanggan.

   Untuk dapat membuat tabel pemesanan tiket kita dapat menggunakan perintah '**create view**' dan untuk menampilkan tabel tersebut menggunakan perintah '**select'.**

2. Buat *view* pada tabel pemesanan, tiket, pelanggan dan data penerbangan, sesuaikan *field*-nya pada setiap tabel seperti di bawah ini.

```
mysql> use penerbangan;
Database changed
mysql> create view pemesanan_tiket as
    -> select pemesanan.no_pesanan, tiket.no_tiket, pelanggan.email_pel, data_penerbangan.no_penerbangan
    -> from pemesanan, tiket, pelanggan, data_penerbangan
    -> where (pemesanan.no_tiket=tiket.no_tiket) and (data_penerbangan.no_penerbangan=tiket.no_penerbangan)
    -> and (pelanggan.email_pel=pemesanan.email_pel);
Query OK, 0 rows affected (0.05 sec)
```

Gambar 8.1 Membuat *View* Tabel pemesanan_tiket

3. Lalu tampilkan hasil *view* yang dibuat.

```
mysql> select * from pemesanan_tiket;
+------------+----------+-----------------+----------------+
| no_pesanan | no_tiket | email_pel       | no_penerbangan |
+------------+----------+-----------------+----------------+
| 30315865   |        7 | ranti@gmail.com | GA146          |
| 30316543   |       10 | udin@gmail.com  | JT387          |
| 30318035   |        1 | agus@gmail.com  | BA817          |
| 30323415   |        2 | ana@gmail.com   | SJ011          |
| 30325216   |        6 | rahman@gmail.com| BA817          |
| 30325867   |        5 | nira@gmail.com  | BA817          |
| 30351256   |        4 | jamil@gmail.com | GA146          |
| 30352367   |        3 | arni@gmail.com  | GA146          |
| 30353532   |        9 | subki@gmail.com | JT387          |
| 30363921   |        8 | rusdi@gmail.com | GA187          |
+------------+----------+-----------------+----------------+
10 rows in set (0.00 sec)
```

Gambar 8.2 Menampilkan Hasil *View*

4. Membuat *View* Cek Jadwal

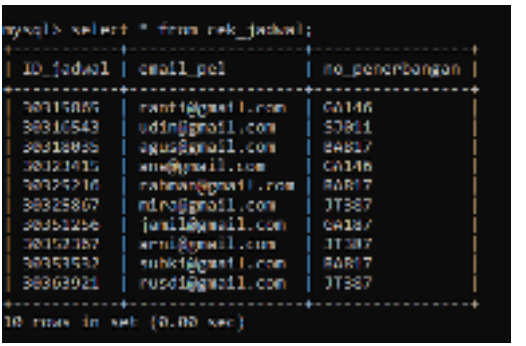
Untuk dapat membuat tabel pemesanan tiket kita dapat menggunakan perintah '**create view**' dan untuk menampilkan tabel tersebut menggunakan perintah '**select'**

5. Buat *view* pada *table* pemesanan, pelanggan dan data penerbangan, sesuaikan *field*nyac pada setiap tabel seperti di bawah ini.

```
mysql> create view cek_jadwal as
    -> select no_pesanan as ID_jadwal, pelanggan.email_pel, data_penerbangan.no_penerbangan
    -> from pemesanan, pelanggan, data_penerbangan
    -> where (pemesanan.email_pel=pelanggan.email_pel)
    -> and (data_penerbangan.no_penerbangan=pelanggan.no_penerbangan);
Query OK, 0 rows affected (0.05 sec)
```

Gambar 8.3 Membuat *View* Tabel cek_jadwal

6. Lalu tampilkan hasil *view* yang telah dibuat.




Gambar 8.4 Menampilkan Hasil *View*

## 8.3 Membuat *Index*

*Index* sering disebut sebagai peningkat performa dari *database*. Penggunaan tabel *index* ini mirip dengan indeks yang ada di bagian belakang sebuah buku. *Index* adalah sebuah tabel spesial dalam sistem *database* yang dapat mempercepat pencarian (*query*) data. *Index* merupakan objek struktur data tersendiri yang tidak bergantung pada struktur tabel. Setiap *index* terdiri dari nilai kolom dan penunjuk (atau ROWID) ke baris yang berisi nilai tersebut.

*Index* ini bertujuan untuk mempercepat pencarian data berdasarkan kolom tertentu. Apabila pada suatu tabel tidak ada *index*, maka dalam proses pencarian data membutuhkan waktu yang cukup lama (apabila data dalam jumlah yang besar).

Berikut ini adalah beberapa alasan mengapa *index* diperlukan.

1. Kolom sering digunakan dalam klausa WHERE atau dalam kondisi JOIN.
2. Kolom berisi nilai dengan jangkauan yang luas.
3. Kolom berisi banyak nilai *null*.
4. Tabel berukuran besar.

Adapun beberapa kondisi di mana tidak diperlukannya *index*, yaitu ketika:

1. tabel berukuran kecil,

2. kolom jarang digunakan sebagai kondisi dalam pencarian (*query*), dan
3. tabel sering di-*update*.

Cara membuat *Index* yaitu mengikuti tahapan berikut ini.

1. Bentuk perintah *index* unik.
```
CREATE UNIQUE INDEX nama_index ON nama_tabel
nama_kolom_yang_diindex);
```
2. *Index* lebih dari satu kolom.
```
CREATE UNIQUE INDEX nama_index ON nama_tabel
nama_kolom_yang_diindex1,
nama_kolom_yang_diindex2);
```
3. *Index* yang nilai kolom dalam sejumlah *record* boleh sama.
```
CRETAE INDEX nama_index ON nama_tabel
nama_kolom_yang_diindex);
```

Cara melihat hasil *index* yaitu seperti tahapan berikut ini.

1. Melihat *index* dengan satu kolom
```
SELECT nama_kolom_index FROM nama_tabel
WHERE nama_field(primary key) ;
```
2. Melihat *index* lebih dari satu kolom
```
SELECT nama_kolom_index1, nama_kolom_index2 FROM
nama_tabel WHERE" nama_field(primary key)";
```
Cara mengahapus *index* yaitu sebagai berikut.
```
DROP INDEX nama_index ON nama_tabel;
```

## 8.4 Uji Coba *Index*

1. Proses *Index* untuk satu kolom

Pada proses *index*, apabila pada tabel Pemesanan memiliki data yang sangat banyak, dan kita ingin mencari harga_total sesuai dengan no_pesanan pelanggan maka akan membutuhkan waktu yang cukup lama (*buffering*). Jadi, dengan menggunakan *index*, waktu yang diperlukan menjadi lebih singkat dan kita dapat melihat satu kolom *index* yaitu harga_total.

a. Membuat *index* yang hanya berfokus pada tabel pemesanan.

```
MariaDB [penerbangan]> create index harga_total on pemesanan (harga_total);
Query OK, 0 rows affected (0.339 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Gambar 8.5 Membuat *Index* harga_total

b. Menampilkan harga_total pada tabel pemesanan yang no_pesanan nya "30363921".

```
MariaDB [penerbangan]> select harga_total from pemesanan
    -> where no_pesanan = 30363921;
+-------------+
| harga_total |
+-------------+
|     1450000 |
+-------------+
1 row in set (0.044 sec)
```

Gambar 8.6 Menampilkan Hasil *Index*

2. Proses *index* unique lebih dari satu kolom

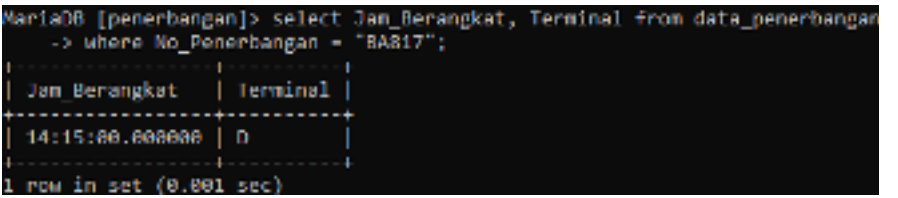
Pada proses *index unique* lebih dari satu kolom ini kita dapat melihat dua kolom *index* pada tabel data_penerbangan yaitu kolom Jam_Berangkat dan Terminal. Sehingga lebih mempermudah kita untuk mencari informasi.

a. Membuat *index unique* yang hanya berfokus pada tabel data_penerbangan.

```
MariaDB [penerbangan]> create unique index lihat_jadual on data_penerbangan
    -> (Jam_Berangkat, Terminal);
Query OK, 0 rows affected (0.246 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Gambar 8.7 Membuat *Index Unique* lihat_jadwal

b. Menampilkan Jam_Berangkat dan Terminal pada tabel data_penerbangan yang No_Penerbangan “BA817”.




Gambar 8.8 Menampilkan Hasil *Index Unique*

## 8.5 Soal Latihan

1. Proses *View*
   a. Buat *view* dari tabel menu pilih nm_menu, dari tabel transaksi pilih total_harga, id_pelanggan dan tgl_trx.
   b. Tampilkan *view* yang telah dibuat.
2. Proses *Index*
   a. Buat *index* untuk melihat harga pada tabel menu.
   b. Tampilkan harga di tabel menu yang memiliki kd_menu M10.

# BAB 9

# TRANSACTION

**Capaian Pembelajaran**

1. Mampu memahami lebih lanjut tentang perintah SQL *transact* SQL pada DBMS MySQL.
2. Mampu mengimplementasikan pembuatan *transact* SQL yang diperlukan pada DBMS MySQL.

## 9.1 Transaction

*Transaction* atau transaksi adalah sekelompok berurutan operasi manipulasi *database* yang dilakukan seolah-olah sebagai satu kesatuan unit tunggal, dengan kata lain transaksi tidak akan lengkap jika semua operasi dalam kelompok berhasil. jika ada salah satu operasi yang gagal maka seluruh transaksi akan gagal. Transaksi adalah implementasi dari suatu operasi yang dapat mengakses *database* dan juga dapat mengubah data dari *database* tersebut.

DBMS atau "*Database Management System*" atau yang digunakan untuk membangun basis data yang berbasis komputerisasi harus mendukung operasi transaksi yang kita gunakan. Artinya operasi transaksi harus dikerjakan secara menyeluruh (tidak parsial), karena dapat mengakibatkan basis data yang tidak konsisten

### 9.1.1 Tujuan Transaksi

Adapun tujuan daripada Transaksi adalah untuk menghindari suatu masalah pada data yang ada di *database* kita, seperti data yang rusak atau bahkan data yang sampai hilang. Dan untuk mendukung Transaksi tersebut agar dapat tetap konsisten, maka Transaksi wajib mempunyai beberapa sifat-sifat sebagai berikut.

1. *Atomicity*, adalah keseluruhan dari tindakan yang harus diselesaikan atau dibatalkan.

2. *Consistency*, yakni menunjukkan konsistensi data yang ada setelah terjadi transaksi.
3. *Isolation*, data yang sedang di lakukan perubahan tidak boleh diakses oleh lebih dari satu operasi.
4. *Durability*, memastikan data yang telah disimpan (*committed data*) disimpan oleh sistem sebagaimana keadaannya, bahkan jika dalam keadaan kegagalan sistem dan *restart* sistem, data tersebut tersedia dalam tahapan dan keadaan yang benar.

Dari suatu transaksi akan menghasilkan dua kemungkinan sebagai berikut. Kemungkinan pertama adalah jika transaksi tersebut dilakukan dengan baik dalam arti keseluruhan maka dapat dikatakan bahwa transaksi itu dapat di-*commit* dan *database* tersebut menjadi konsisten baru. Jika transaksi yang sudah pasti di *commit* maka transaksi tersebut tidak dapat dikembalikan lagi, kecuali melakukan transaksi ulang seperti mengambilkan ke transaksi ke sebelumnya menggunakan beberapa operasi. Kemungkinan kedua adalah jika transaksi tersebut tidak dilakukan atau dibatalkan maka dapat dikatakan bahwa *database* yang melakukan transaksi tersebut menjadi seperti keadaan konsisten sebelumnya (*rollback*).

### 9.1.2 Status Transaksi

Adapun status dari transaksi adalah sebagai berikut.

1. Aktif (*active*) adalah status awal (*initial state*) dari suatu operasi transaksi yang menandakan bahwa transaksi tersebut dapat dieksekusi.
2. Berhasil sebagian (*partially committed*) adalah status di mana operasi transaksi yang bertepatan pada operasi terakhir dalam transaksi tersebut sudah diselesaikan.
3. Gagal (*failed*) adalah status di mana operasi transaksi berada pada terhentinya pengeksekusian, padahal operasi belum selesai.
4. Batal (*aborted*) adalah status di mana operasi transaksi tidak jadi dikerjakan dan pastinya sebelum itu data sudah dikembalikan ke bentuk semulanya.
5. Berhasil sempurna (*committed*) adalah status di mana operasi transaksi dianggap telah selesai dalam keseluruhan sehingga

*database* dapat menampilkan perubahan-perubahan yang kita inginkan pada operasi transaksi tersebut.

Operasi Transaksi memiliki beberapa *state* yang bertujuan untuk mengetahui kapan suatu transaksi itu mulai dan kapan suatu transaksi berhenti sebagai berikut.

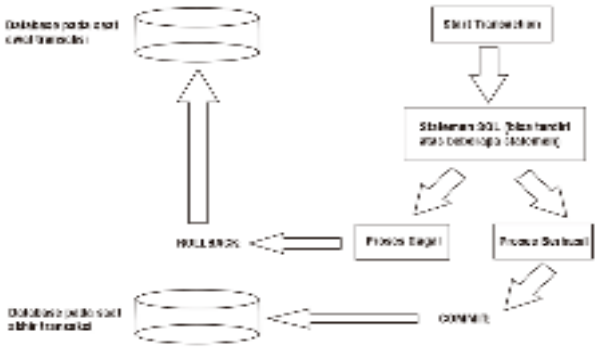
1. BEGIN_TRANSACTION digunakan untuk memulai transaksi.
2. READ/WRITE digunakan untuk operasi-operasi pada operasi transaksi.
3. END_ TRANSACTION digunakan untuk mengakhir transaksi, dan kemudian operasi transaksi akan dicek, jika ada yang ingin dibuat permanen atau digagalkan.
4. COMMIT_TRANSACTION digunakan untuk memberikan *signal* jika operasi transaksi telah sukses.
5. ROLLBACK (*abort*) digunakan untuk memberikan *signal* jika operasi transaksi ada yang gagal dan perubahan pada *database* harus dihapuskan.

Untuk melakukan suatu transaksi digunakan *syntax* berikut.

```
BEGIN {TRAN | TRANSACTION }
[ transaction_name |@tran_name_variable]
[ WITH MARK [ 'description'] ]
[ database_operation ]
COMMIT { TRAN | TRANSACTION }
[ transaction_name |
@tran_name_variable ]
```

### 9.1.3 Alur Operasi Transaksi

Adapun contoh alur dari pembuatan operasi transaksi sebagai berikut.



Gambar 9.1 Alur Operasi Transaksi

Dalam membuat dan menjalankan *transaction* ada beberapa hal yang perlu diperhatikan sebagai berikut.

1. **Mesin Penyimpanan**

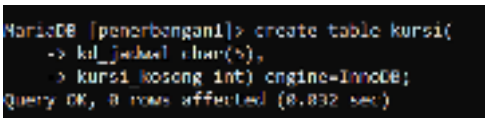
MySQL menyiapkan berbagai jenis penyimpanan, yang akan menetapkan bagaimana sebenarnya suatu *table* disimpan. Secara tipe yang dipakai untuk *table* adalah MyISAM. Mengenai jenis mesin penyimpan lain yang didukung oleh MySQL antara lain *Memory* (*Heap*), BDB (BerkeleyDB), dan InnoDB.

Secara khusus tipe InnoDB akan dibahas karena mendukung fitur transaksi. Fitur ini sangat bermanfaat karena mengharuskan penanganan *Commit* dan *Rollback*, yang biasanya tersedia pada DBMS komersial.

Perlu diketahui, walau InnoDB mendukung transaksi, bukan berarti InnoDB lebih baik daripada tipe seperti MyISAM. Bila Anda memang tidak memerlukan transaksi akan lebih baik kalau menggunakan tipe seperti MyISAM dikarenakan kinerja menjadi lebih cepat. Selain itu, perlu juga diketahui bahwa implementasi InnoDB masih memiliki beberapa batasan, antara lain tidak mendukung tipe kolom spasial dan proses untuk menghitung semacam SELECT COUNT(*) FROM *table* berjalan lebih lama dari pada pada MyISAM karena implementasi InnoDB secara internal tidak tercatat jumlah baris dalam tabel.

**2. Menciptakan Tabel Bertipe InnoDB**

Untuk menciptakan tabel bertipe InnoDB, perlu penambahan ENGINE=InnoDB di bagian belakang pernyataan CREATE *TABLE*. Untuk mempraktikkan hal ini, berikan perintah berikut terlebih dulu untuk mengaktifkan *database test*.




Gambar 9.2 Penerapan perintah membuat tabel dengan *engine* InnoDB

Dengan perintah di atas maka mesin penyimpanan akan mengubah tabel barang menjadi tabel dengan tipe InnoDB

**3. Memahami Transaksi**

Transaksi adalah sederetan operasi yang berkedudukan sebagai satu kesatuan proses. Dalam dunia nyata, proses pengambilan uang melalui ATM merupakan contoh sebuah transaksi, yang mencakup pemasukan kartu ATM, pemasukan nomor PIN (*Personal Identification Number*), penentuan jumlah uang yang akan diambil, hingga pengambilan uang itu sendiri. Dalam proses transaksi seperti itu, ada dua kemungkinan yang seharusnya dipenuhi, yaitu:

a. transaksi dianggap berhasil jika semua proses berjalan lancar dan
b. transaksi dianggap gagal kalau ada salah satu bagian proses yang gagal.

**4. Cara Menggunakan Transaksi**

Setiap klien MySQL yang melakukan koneksi ke MySQL server menggunakan mode AUTOCOMMIT, yang berarti setiap perintah SQL dengan sendirinya akan mengalami proses COMMIT. COMMIT sendiri berarti "menyetujui perintah untuk segera dimutakhirkan ke *database*".

Untuk mematikan mode AUTOCOMMIT, maka dapat menggunakan perintah SET AUTOCOMMIT=0. Berdasarkan kondisi tersebut suatu perintah SQL yang melaksanakan perubahan data di

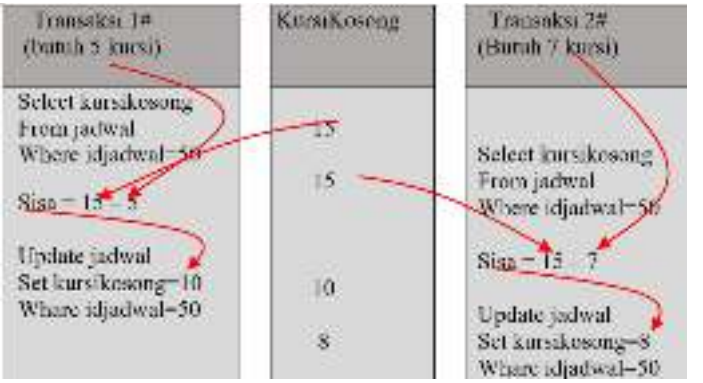
*database* dapat disetujui untuk mengeksekusi perintah simpan secara menetap dengan menggunakan perintah COMMIT atau dapat pula dibatalkan dengan menggunakan perintah ROLLBACK.

**5. Problem pada *Multiuser***

Sedangkan lingkungan yang *multiuser*, akses terhadap *database* dilakukan sejumlah orang. Pada keadaan seperti ini, muncul masalah yang mungkin terjadi. Berikut adalah masalah yang perlu diketahui.

a. Pemutakhiran yang hilang (*lost update*).

Masalah operasi *update* yang sukses dari seorang pengguna kemudian diikuti oleh operasi *update* dari pengguna lain.

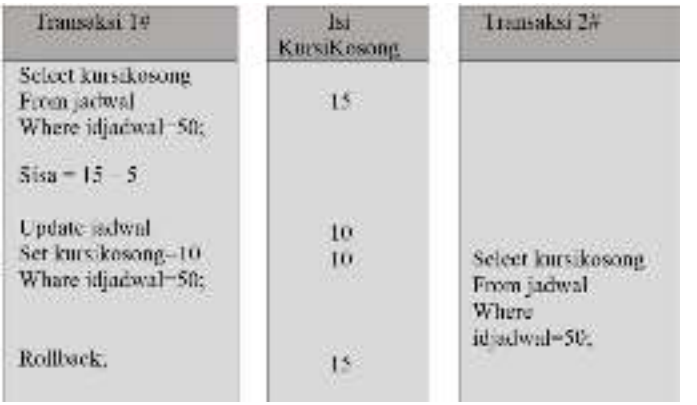



Gambar 9.3 Ilustrasi Masalah Pemutakhiran yang Hilang

Berdasarkan ilustrasi di atas tampak bahwa dari transaksi pertama menghasilkan *update* kursikosong sebanyak 10 kursi yang seharusnya setelah dikurangi dengan *update* perintah yang kedua yaitu sisa kursikosongnya adalah sebanyak 3 kursi, tetapi dikarenakan terjadi kesalahan dikarenakan eksekusi proses yang bersamaan menyebabkan data menjadi tidak valid, dan ini memiliki pengaruh yang sangat besar dalam suatu proses bisnis.

b. Pembacaan Kotor.

Masalah terjadi saat suatu transaksi membaca data dari transaksi lain yang belum di-*commit*.

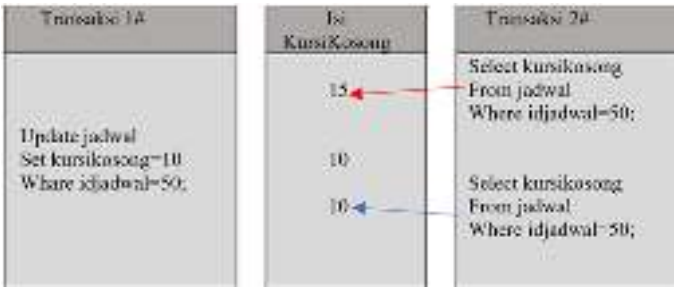



Gambar 9.4 Permasalahan akibat pembacaan kotor

Dari ilustrasi Gambar 9.4 di atas terlihat bahwa proses transaksi kedua melihat kolom kursikosong bernilai 10, tetapi ternyata beberapa saat kemudian nilai kolom tersebut telah berubah menjadi 15 yang diakibatkan perintah ROOLBACK dari transaksi pertama.

c. Pembacaan yang Tidak Sama.

Kasus ini terjadi jika seseorang pemakai menemukan nilai yang berubah dalam jangka waktu yang sangat pendek. Contoh dapat dilihat pada gambar berikut:




Gambar 9.5 Permasalahan akibat pembacaan tidak sama

Berdasarkan ilustrasi pada Gambar 9.5 terlihat bahwa kondisi yaitu kolom kursikosong bernilai 15, tetapi beberapa saat terjadi

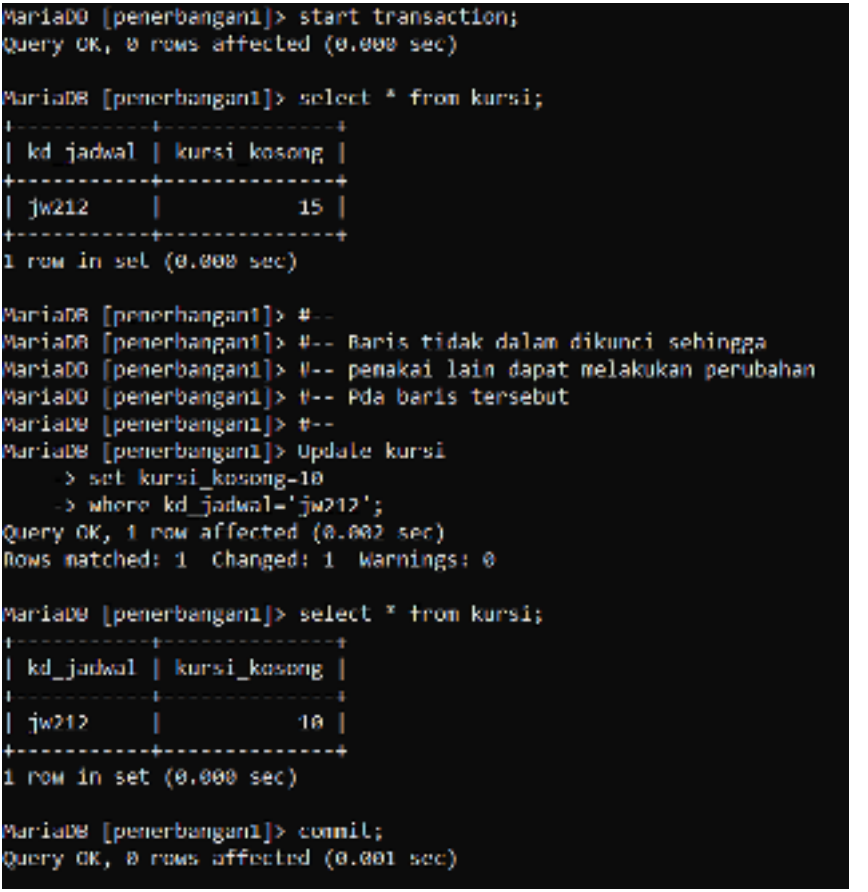
proses *update* data yaitu data kursikosong berubah menjadi 10. Hal seperti ini menjadi masalah bagi transaksi kedua berupa suatu proses yang membaca data dalam waktu yang lama, misalnya membuat sebuah ringkasan

d. Penyisipan yang Tak Dikehendaki.

   Penyisipan yang tidak dikehendaki adalah kasus khusus dari pembacaan yang tidak sama. Hal ini terjadi apabila instruksi INSERT saat pengguna lain sedang melakukan suatu *query* tersebut belum selesai. Penyisipan ini akan menimbulkan perbedaan hasil saat diakses oleh pengguna lain.

**6. Penanganan *Multiuser***

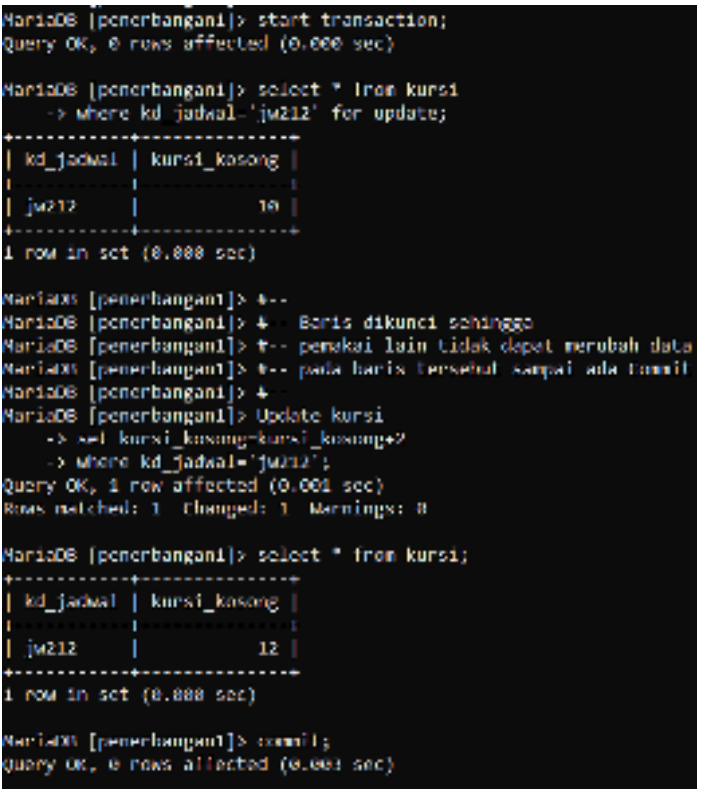
Metode penanganan *multiuser* (yaitu suatu proses pengaksesan basis data secara bersamaan oleh sejumlah pemakai) atau disebut dengan istilah penguncian (*locking*). Penguncian yang dilakukan oleh para pemakai dapat dilakukan terhadap semua baris atau sejumlah baris dalam *database*. Melihat kondisi tersebut perintah bawaan MYSQL akan melakukan penguncian secara otomatis pada baris yang diakses, tetapi ada saatnya penguncian harus dilakukan secara manual.




Gambar 9.6 Contoh aktivitas dalam transaksi

Berdasarkan pada contoh di atas, *user* yang terhubung dengan *database* dapat melakukan operasi untuk mengubah baris tersebut sebelum dilakukan UPDATE. Hasilnya, perintah UPDATE yang mengubah jumlah menjadi 10 dapat memberikan hasil yang salah terhadap kolom tersebut.

Jika dalam pengelolaan *database* Anda tidak mau orang lain dapat mengubah baris tersebut sebelum pemakai pertama memberikan COMMIT, maka dapat menambahkan perintah FOR UPDATE pada perintah SELECT. Perhatikan gambar berikut:




Gambar 9.7 Cara mengunci baris pada database multiuser

Pada contoh di atas, baris dengan **kd_jadwal='jw212'** akan tetap terkunci dan akan terbuka ketika dilaksanakannya atau diberikan perintah COMMIT. Penguncian dilakukan dengan FOR UPDATE pada pernyataan SELECT. Dari ilustrasi gambar 9.7 di atas maka sebelum COMMIT, orang lain tidak dapat mengubah baris (harus menunggu sampai pemakai yang mengunci memberikan COMMIT atau waktu tunggu habis).

Selain FOR UPDATE, perintah LOCK IN SHARE MODE juga bisa dapat digunakan. Kedua perintah tersebut mempunyai kemampuan melindungi dari memungkinkan orang lain melakukan perubahan pada baris yang dikunci. Perbedaannya, SELECT-FOR UPDATE hanya bisa dijalankan oleh satu pengguna, sedangkan SELECT-FOR SHARE MODE bisa dijalankan oleh banyak pengguna.

**7. Penguncian Secara Otomatis**

MySQL menggunakan penguncian secara otomatis terhadap sejumlah perintah yang memperbaharui *database*. Beberapa pernyataan yang menggunakan penguncian secara otomatis adalah sebagai berikut.

a. Pernyataan UPDATE, saat UPDATE dilakukan, semua baris yang diubah akan dikunci. Jika ada yang memperbaharui baris yang sama, maka pengguna yang akan melakukan UPDATE selanjutnya harus menunggu sampai UPDATE yang pertama mengalami COMMIT.
b. Pernyataan INSERT, pernyataan INSERT membuat semua kunci primer atau kunci yang unik akan dikunci. Hal inilah yang memungkinkan pencegahan kunci yang identik.
c. Pernyataan SELECT dengan klausa FOR UPDATE dan LOCK IN SHARE MODE Saat SELECT dengan FOR UPDATE dan LOCK IN SHARE MODE dilakukan, semua baris yang dihasilkan akan dikunci sampai ada COMMIT atau ROLLBACK.

## 9.2 Uji Coba

Ketika pelanggan ingin bepergian menggunakan pesawat, yang pertama kali yang dilakukan adalah memesan tiket, nah di dalam *database* penerbangan memiliki tabel data_penerbangan yang berisi beberapa *field* yang datanya tidak bisa dirubah ketika ada pelanggan telah memesan tiket (terutama di bagian jumlah kursi pesawat yang tidak ter-*update* apakah berkurang atau tidak). Untuk itu pada BAB ini kita akan membahas tentang tabel transaksi, yang nantinya akan berfungsi untuk mengubah dari tabel data_penerbangan jika ada pelanggan memesan tiket.

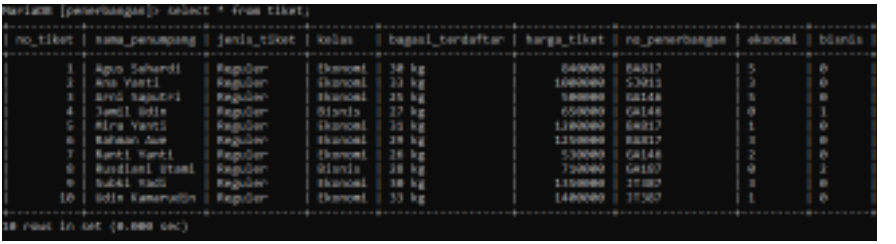
Dari kasus di atas dapat didapat jika kita *menggunakan* tabel transaksi maka tabel tersebut akan melakukan *insert* di *table* tiket dan mengurangi jumlah kursi_eko di tabel data_penerbangan jika no_penerbangan pada tabel tiket sama, namun jika no_penerbangan tidak sama, maka akan melakukan *rollback*. Otomatis karena di data_penerbangan terdapat 2 kelas yakni Ekonomi dan Bisnis maka kita harus membuatkan 2 tabel transaksi yang masing-masing untuk kelas tersebut.

**1. Membuat Tabel Transaksi Ekonomi.**

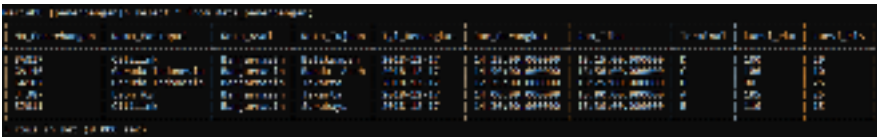
Membuat *transact* untuk kejadian 'pengurangan' dengan nama "insert_kursiEko" yang dikombinasikan dengan *procedure* sesuai studi kasus di atas.

Gambar 9.8 *Procedure Transaction* insert_kursiEko

Menampilkan kondisi tabel tiket dan tabel data_penerbangan agar nantinya kita dapat melihat perubahan di tabel tersebut.



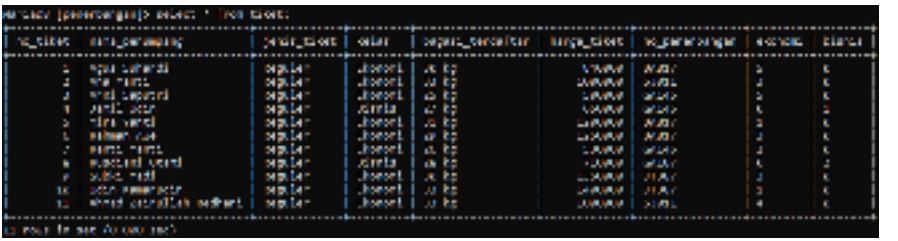
Gambar 9.9 Menampilkan Data Tabel Tiket



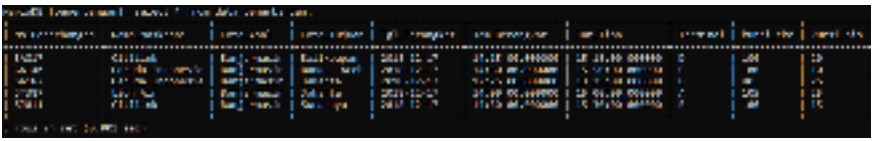
Gambar 9.10 Menampilkan Data Tabel data_penerbangan

Lakukan proses *call* insert_kursiEko dan lihat perubahan stok pada tabel data_penerbangan apakah stok dari kursi_eko akan berkurang.

Gambar 9.11 Proses *Call* insert_kursiEko



Gambar 9.12 Menampilkan Data Tabel Tiket



Gambar 9.13 Menampilkan Data Tabel data_penerbangan

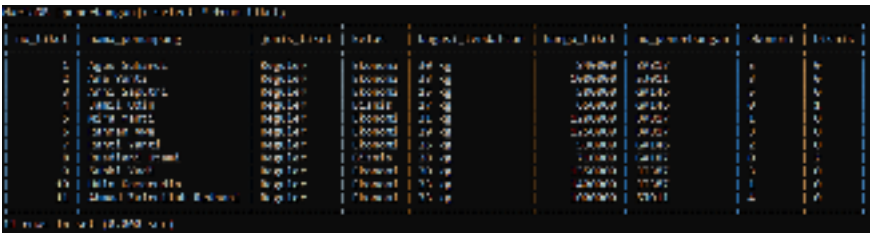
Bisa dilihat perbedaan setelah kita memanggil insert_kursiEko maka akan berubah pula di kedua tabel di atas.

**2. Membuat Tabel Transaksi Bisnis.**

Membuat *transact* untuk kejadian 'pengurangan' dengan nama "insert_kursiBis" yang dikombinasikan dengan *procedure* sesuai studi kasus di atas.

Gambar 9.14 Membuat *Procedure Transaction* insert_kursiBis

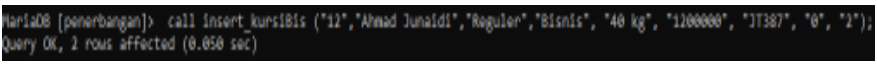
Menampilkan kondisi tabel tiket dan tabel data_penerbangan agar nantinya kita dapat melihat perubahan ditabel tersebut.



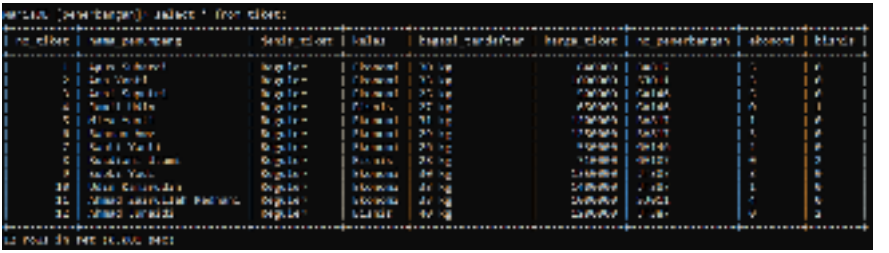
Gambar 9.15 Menampilkan Data Tabel Tiket

Gambar 9.16 Menampilkan Data Tabel data_penerbangan

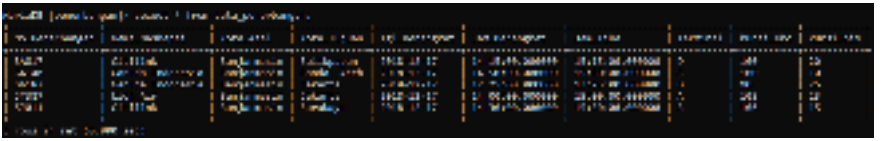
Lakukan proses *call* insert_kursiBis dan lihat perubahan stok pada tabel data_penerbangan apakah stok dari kursi_bis akan berkurang.




Gambar 9.17 Proses *Call* insert_kursiBis



Gambar 9.18 Menampilkan Data Tabel Tiket



Gambar 9.19 Menampilkan Data Tabel data_penerbangan

Bisa dilihat perbedaan setelah kita memanggil inser_kursiBis maka akan berubah pula di kedua tabel di atas.

## 9.3 Soal Latihan

Berdasarkan *database* dbcaffe buatlah transaksi dengan ketentuan sebagai berikut.

1. Buat *transact* untuk kejadian ‘tambah’ dengan nama “insert_tambah” yang dikombinasikan dengan *procedure*.
2. Lakukan proses *call* insert_tambah dan lihat perubahan stok *table* tb_buku apakah stok bertambah.

# DAFTAR PUSTAKA


Date, C.J. 2006. *An Introduction to Database Sistems 8th.* Pearson Education.

Korth, Henry F., Abraham Silberschatz. 2011. *Database Sistem Concepts 6th Edition*. McGraw-Hill.

Mulyanto, Agus. 2009. *Teori Client Server*. Universitas Komputer Indonesia, Bandung.

Raharjo, Budi. 2015. *Belajar Otodidak Teknik Pembuatan dan Pengelolaan Database*. Informatika, Bandung.

Ramakrishnan, Raghu and Johannes Gehrke. 2003. *Database Management Sistems Third Edition*. McGraw-Hill.

Rozaq, Abdul. 2019. *Sistem Basis Data MySQL Pada Konsep Jaringan Klien Server.* Banjarmasin: Poliban Press.

Solihin, Achmad. 2010. *MySQL 5: Dari Pemula Hingga Mahir*. Universitas Budi Luhur, Jakarta.

Ullman, Jeffrey, Jennifer Widom, and Hector Garcia-Molina. 2013. *Database Sistems*. Pearson New International Edition: The Complete Book.

# BIOGRAFI SINGKAT PENULIS

**Rahimi Fitri, S.Kom., M.Kom.** lahir dari orang tua bernama Dr. H. Karyono Ibnu Ahmad dan Hj. Sumiatun, S.Pd., M.A.P. di Banjarmasin pada tanggal 22 Juli 1982. Meraih gelar sarjana di bidang Teknik Informatika Universitas Komputer Indonesia pada tahun 2004 dan meraih gelar master bidang Teknik Informatika di Institut Teknologi Sepuluh November pada tahun 2011. Saat ini, mengajar di Program Studi D-3 Teknik Informatika Politeknik Negeri Banjarmasin dengan mata kuliah yang diajarkan adalah mata kuliah Sistem Basis Data, Basis Data 2 dengan Menggunakan MySql, Rekayasa Perangkat Lunak dan Metode Numerik.

# Pemrograman Basis Data

## *Menggunakan* MySQL

RAHIMI FITRI

Penulisan buku **Pemrograman Basis Data Menggunakan MySQL** ini adalah dalam rangka melengkapi perangkat pembelajaran mata kuliah Basis Data Lanjut pada program studi Teknik Informatika, Politeknik Negeri Banjarmasin.

**Capaian Pembelajaran:**

1. Mampu memahami, menguasai dan mampu mengimplementasi teori, konsep dan prinsip pemrograman database MySQL.
2. Mampu menginstal perangkat pendukung pemrograman basis data menggunakan MySQL.

Penerbit Politeknik Negeri
Alamat :
Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry, Pangeran, Kota Baru, Banjarmasin 70123, Kalimantan Selatan
Telp. x (0511) 3305052
Email : press@poliban.ac.id