



**Kampus
Merdeka**
INDONESIA JAYA



MySQL Database



Dosen Pengampu ➤

Ikhwan Alfath Nurul Fathony





ERD

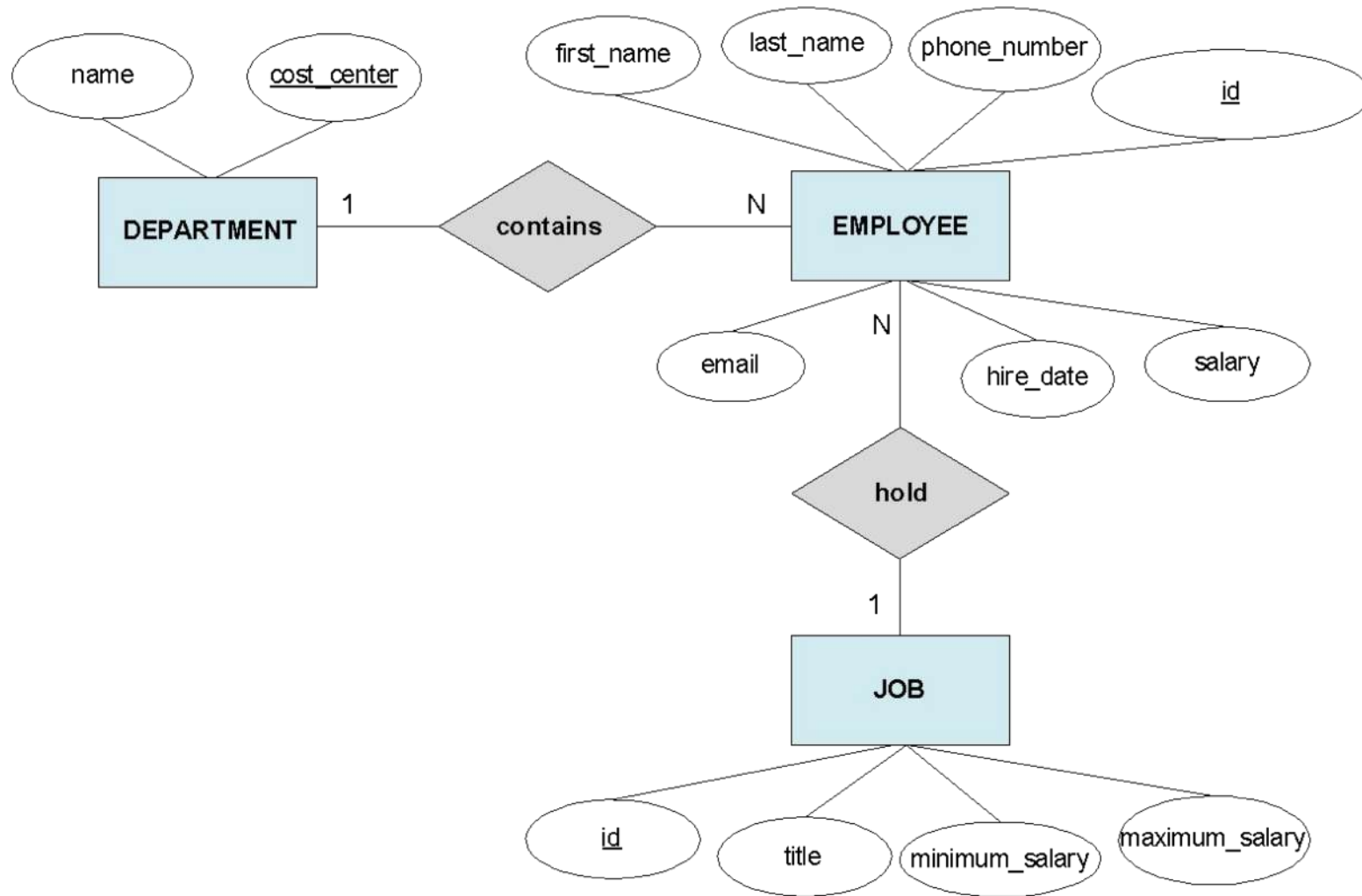
Web Developer

Prodi Teknologi Informasi
Fakultas Teknik
Universitas Tidar Magelang

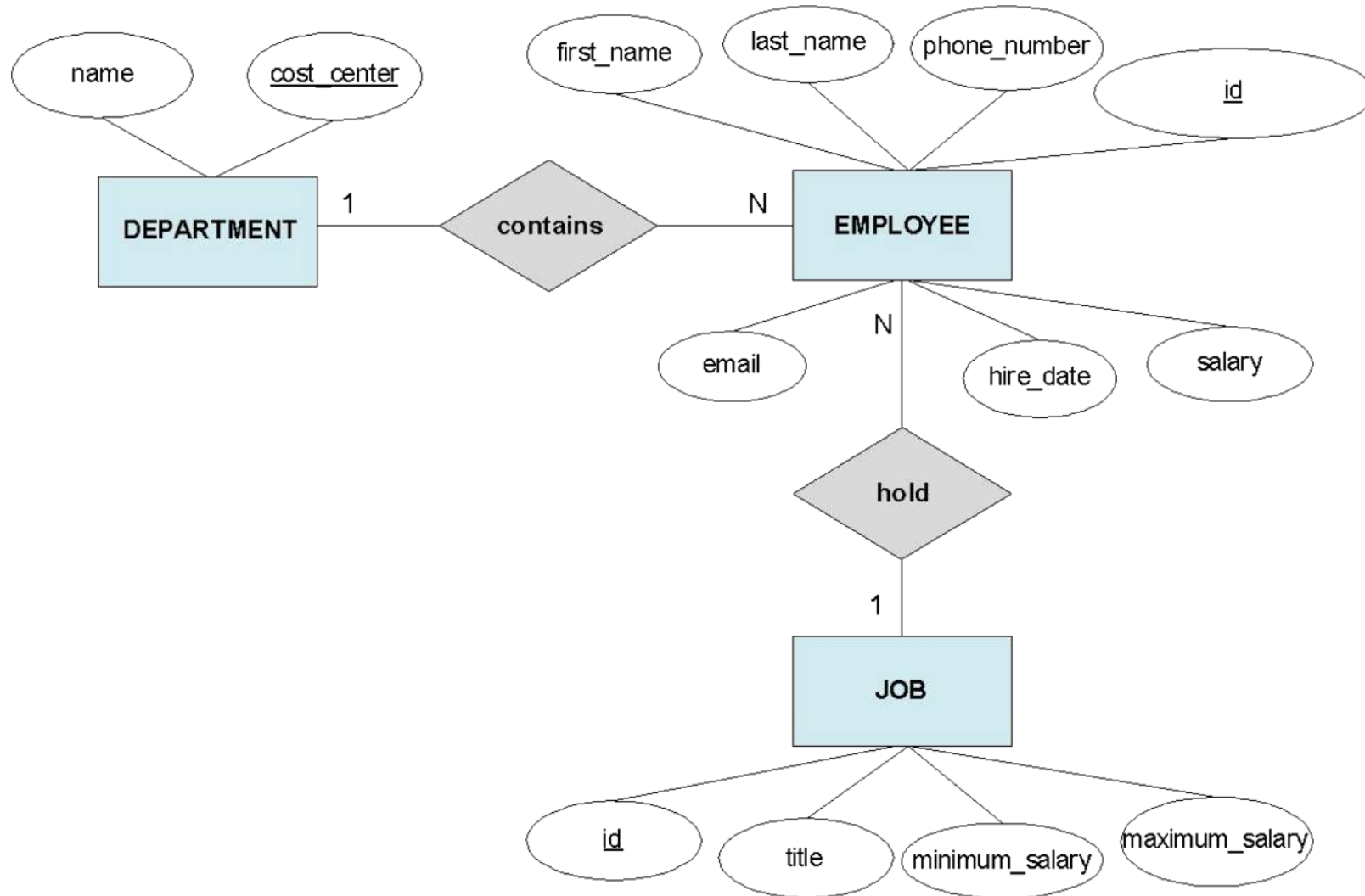
Topik

- ERD

BAGAIMANA BENTUK ERD?



KOMPONEN ERD





ENTITY (ENTITAS)

Entitas adalah:

- Konsep dasar dalam pemodelan basisdata berupa individu yang mewakili sesuatu yang nyata dan dapat dibedakan dari sesuatu yang lain.
- Nama dari benda yang dapat Anda list
- Biasanya kata benda

Contoh:

DEPARTMENT

EMPLOYEE

JOB

Karakteristik entitas meliputi:

- Disimbolkan dalam bentuk kotak persegi panjang
- Memiliki nama yang unik, biasanya kata benda
- Nama UPPERCASE (dalam huruf kapital)
- Tidak ada tanda penghubung atau *underscore*

ENTITY (ENTITAS)

Beberapa contoh entitas meliputi:

- ORANG : agen, karyawan, pelanggan
- TEMPAT : negara bagian, negara, kota
- BARANG: persediaan barang, kendaraan, produk
- KONSEP : kebijakan, resiko, cakupan, pekerjaan
- ORGANISASI: agensi, departemen
- KEJADIAN : permintaan layanan, klaim, pemilihan umum

ENTITAS & INSTANCE

Entitas mengandung *instance*.

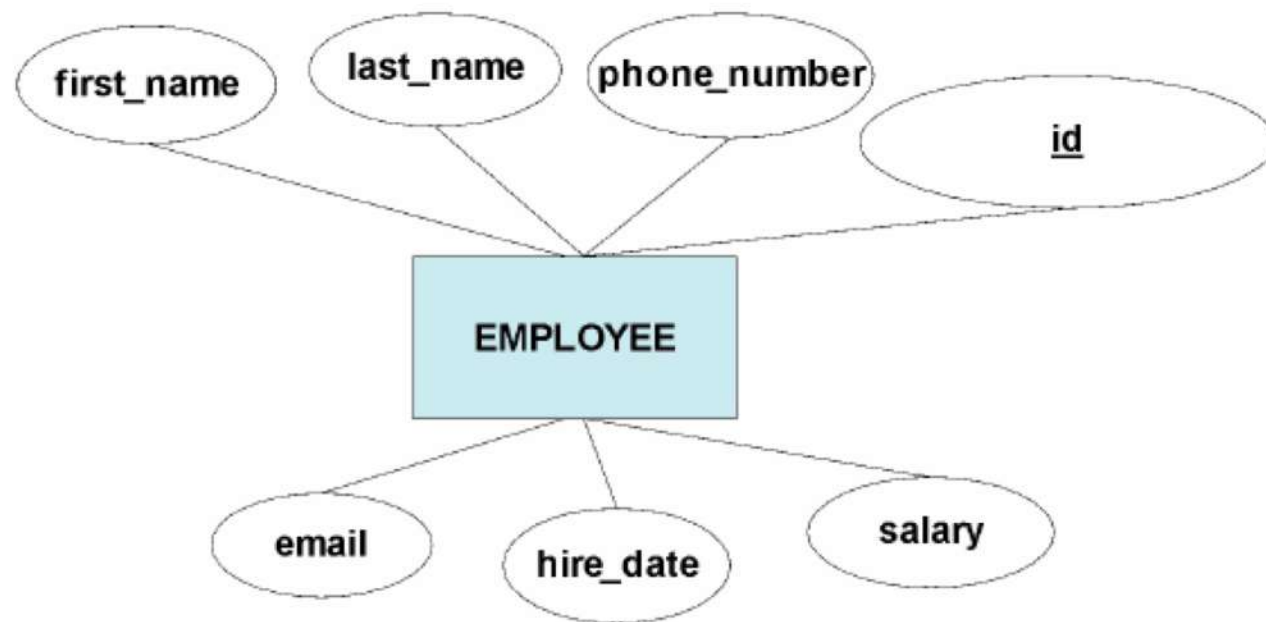
Entitas	<i>Instance</i>
PERSON	John Smith
PRODUCT	2.5 x 35 mm copper nail
PRODUCT TYPE	Nail
JOB	Violinist
SKILL LEVEL	Fluent
DVD TYPE	DVD-R

2

Attribute (Atribut)

Merupakan karakteristik/properti dari entitas yang menyajikan penjelasan detail mengenai entitas tersebut.

Contoh:



Karakteristik Atribut

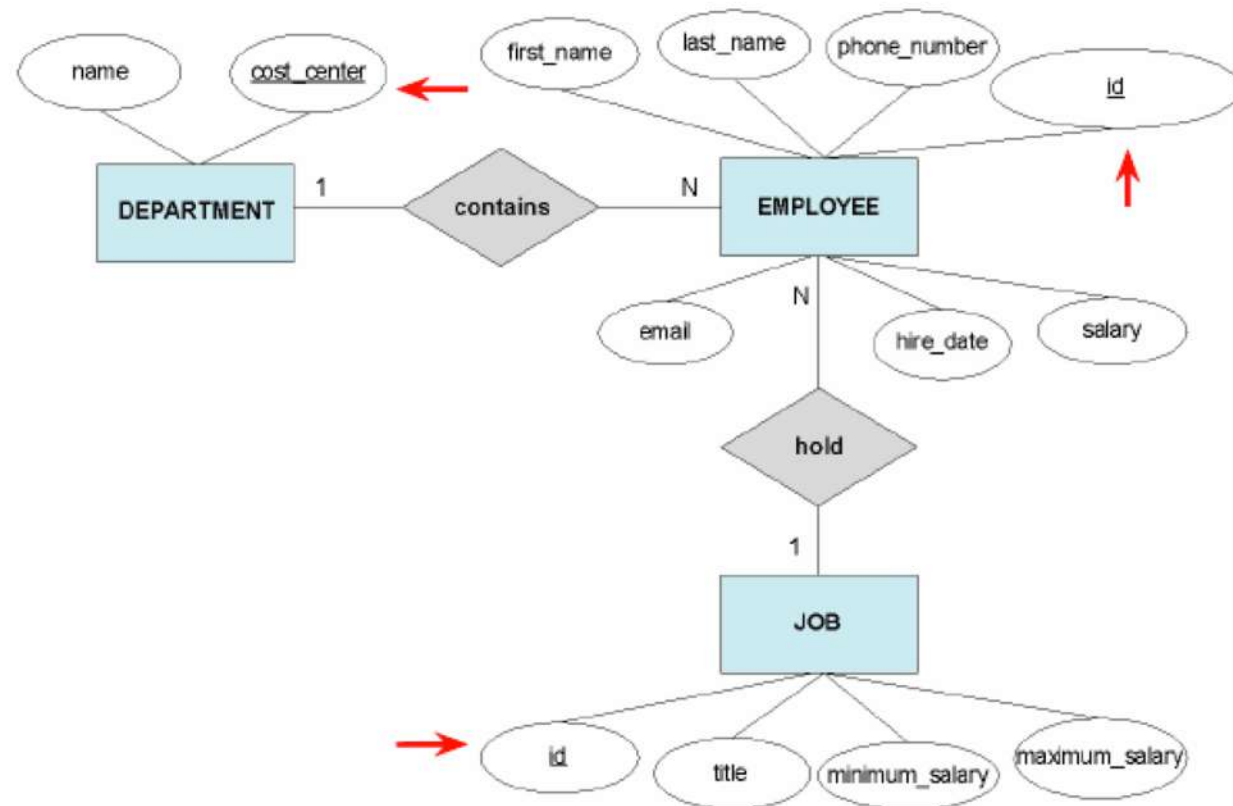
Atribut memiliki karakteristik sebagai berikut:

- Atribut dilambangkan dengan simbol elips.
- Nama-nama atribut adalah tunggal dan ditampilkan dalam huruf campuran atau huruf kecil.
- Nama dari atribut seharusnya tidak memuat nama dari entitas, karena atribut masuk ke dalam kualifikasi dari nama entitas.
- Atribut diklasifikasikan sebagai:
 - Not null
 - Nulls allowed

3

Unique Identifier (UID)

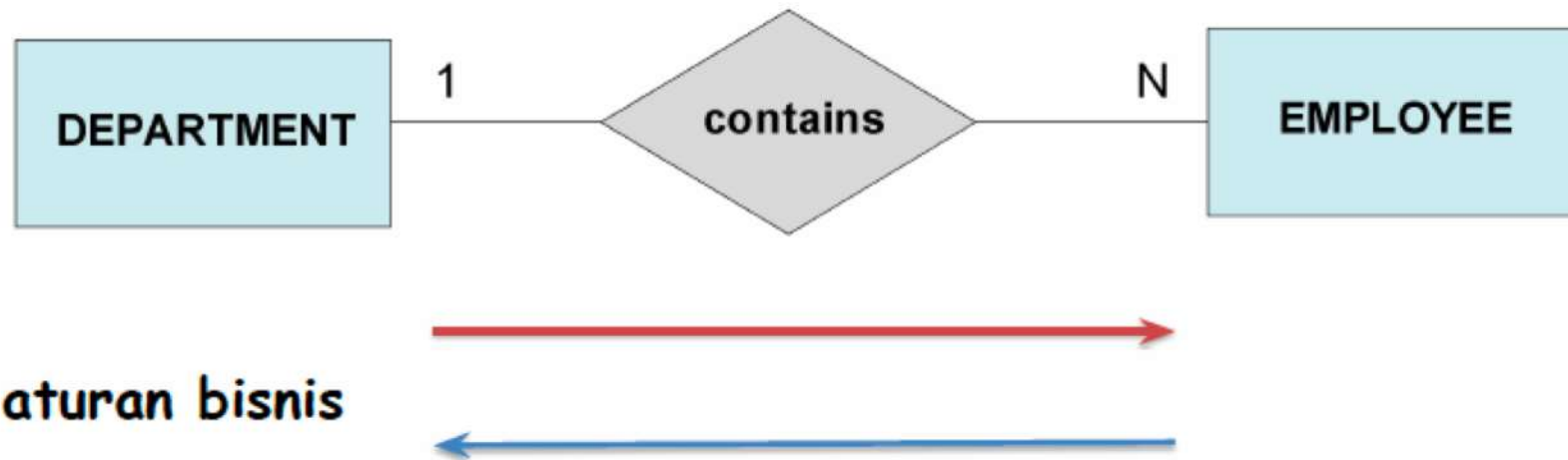
Suatu atribut khusus yang secara unik mengidentifikasi kejadian tertentu dari suatu entitas data.



4

RELATIONSHIP

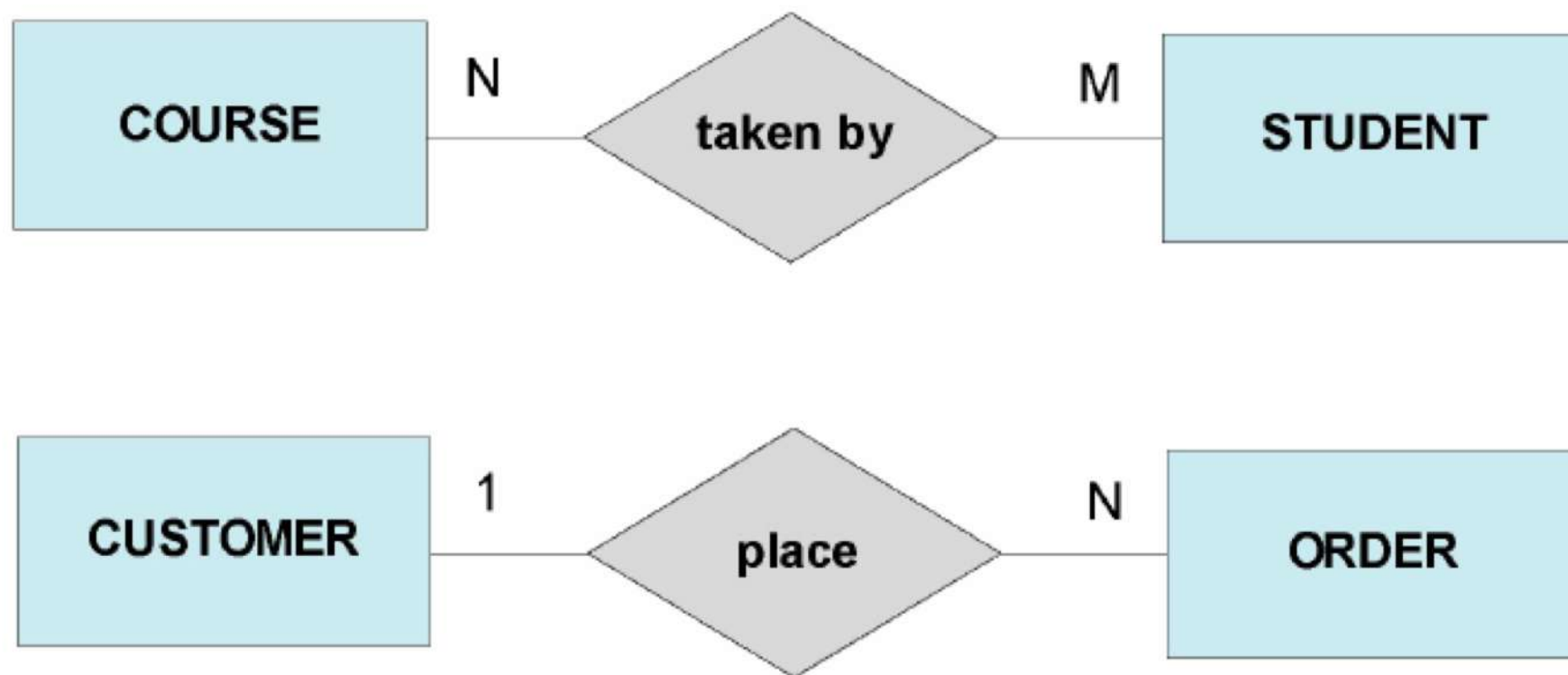
- Sebuah relationship adalah hubungan dua arah, asosiasi signifikan antara dua entitas atau lebih, atau antara sebuah entitas dengan dirinya sendiri.
- Merepresentasikan aturan bisnis yang menghubungkan entitas yang satu dengan entitas yang lainnya.



Karakteristik Relationship

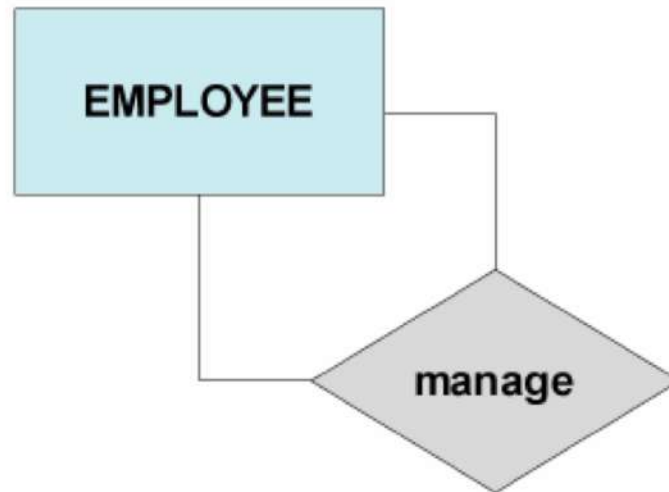
- Disimbolkan dengan belah ketupat/diamond.
- Penamaan umumnya dengan menggunakan kata kerja.
- Umumnya ditulis dengan huruf kecil.

Relationship : Contoh-contoh Tambahan



Relationship Rekursif

Relationship sebuah entitas dengan dirinya sendiri.



Aturan Bisnis:

- **EMPLOYEE mengelola EMPLOYEE.** ? pegawai (misalnya: manajer) mengelola pegawai-pegawai yang lain.
- **EMPLOYEE dikelola oleh EMPLOYEE.** ? pegawai (staf) dikelola oleh pegawai (manajer).

5

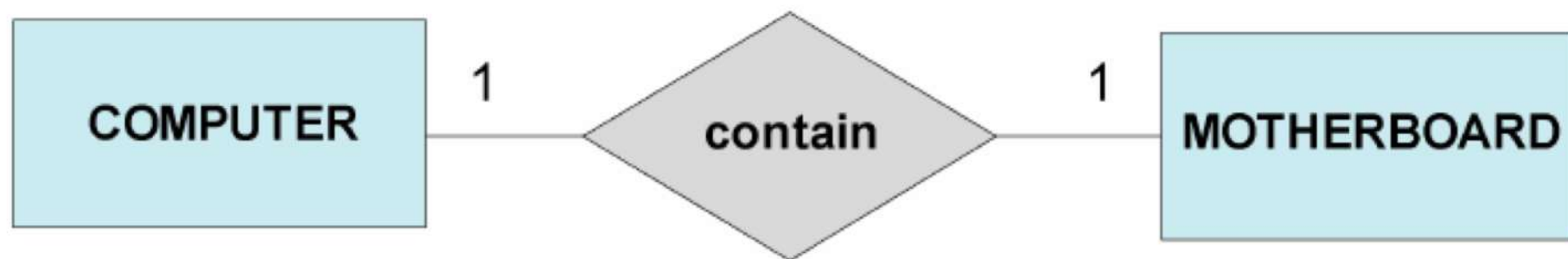
Cardinality (Kardinalitas)

- ❑ Merepresentasikan banyaknya instance suatu entitas yang berelasi dengan instance entitas lainnya.
- ❑ Semua relationship merepresentasikan kebutuhan informasi dan aturan-aturan bisnis.
 - One to one (1:1)
 - One to many (1:N)
 - Many to one (N:1)
 - Many to many (N:M)



One to One (1:1)

Kardinalitas one to one (1:1) memiliki makna relasi **satu dan hanya satu** di kedua arahnya.



Aturan Bisnis:

- **Setiap** COMPUTER harus berisi **satu dan hanya satu** MOTHERBOARD.
- **Setiap** MOTHERBOARD harus terkandung dalam **satu dan hanya satu** COMPUTER.

One to Many (1:N)

Kardinalitas one to many (1:N) memiliki makna relasi **satu atau lebih** pada satu arah dan **satu dan hanya satu** pada arah yang lain.



Aturan Bisnis:

- **Setiap** CUSTOMER dapat memesan **satu atau lebih** ORDER.
- **Setiap** ORDER dapat dipesan oleh **satu dan hanya satu** CUSTOMER.

Many to Many (N:M)

Kardinalitas many to many (N:M) memiliki makna relasi **satu atau lebih** di kedua arahnya.



Aturan Bisnis:

- **Setiap** COURSE diambil oleh **satu atau lebih** STUDENT.
- **Setiap** STUDENT mengambil **satu atau lebih** COURSE.



SQL: Introduction + CRUD

Web Developer

Prodi Teknologi Informasi
Fakultas Teknik
Universitas Tidar Magelang

Learning Outcome

Setelah menyelesaikan sesi pelatihan ini, mahasiswa:

- Memahami konsep SQL
- Memahami elemen SQL
- Memahami penggunaan DDL
- Memahami penggunaan DML
- Memahami penggunaan DRL

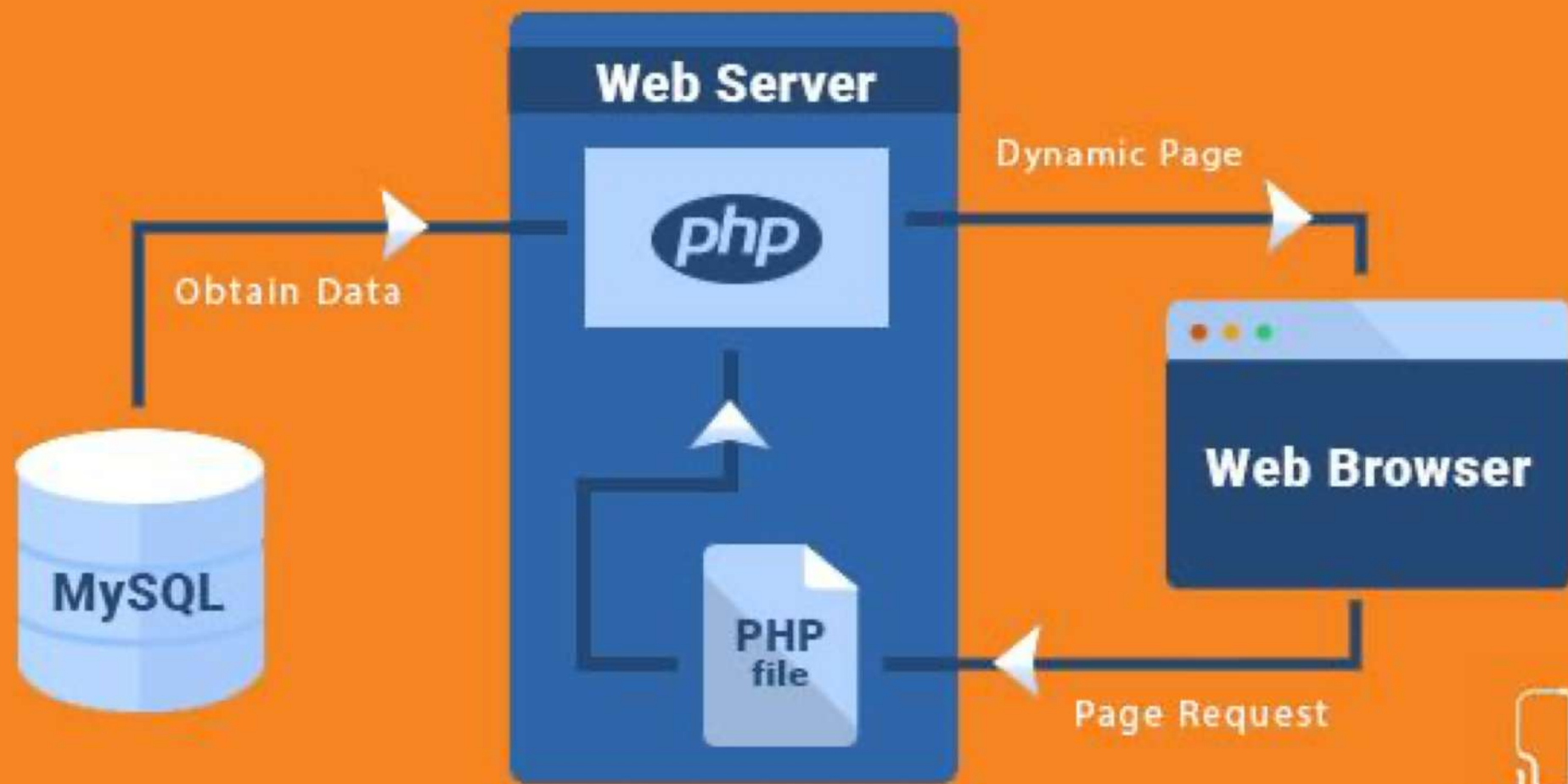
Topik

- Pengertian SQL
- DDL
- DML
- DRL

Apa Itu SQL??

“SQL (Structured Query Language) adalah bahasa query yang dirancang untuk pengambilan informasi tertentu dari *database*.”





Dasar SQL

- SQL = Structured Query Language
- Digunakan untuk mengakses basis data relasional
- Bersifat standar >> bisa dipakai untuk basis data relasional lainnya
- Perintah SQL yang biasa digunakan dibagi menjadi DDL dan DML

5 Bagian Utama SQL

- DDL: Bahasa yang digunakan untuk mendefinisikan data.
Contoh : create, drop
- DML: Bahasa yang digunakan untuk memanipulasi data.
Contoh : insert, update
- Retrieving Data: Perintah untuk menampilkan data dari database. Contoh : select
- DCL : Bahasa untuk kontrol pengendalian akses data ke database. Contoh : grant, revoke
- DTL: Bahasa untuk mengelola transaksi di database. Contoh : commit transaction, rollback transaction

Perintah DDL

- DDL = Definition Data Language
- Digunakan untuk kepentingan penciptaan database, tabel, hingga penghapusan database atau tabel
- Contoh :
 - ☐ CREATE DATABASE
 - ☐ CREATE TABLE
 - ☐ DROP TABLE
 - ☐ ALTER TABLE

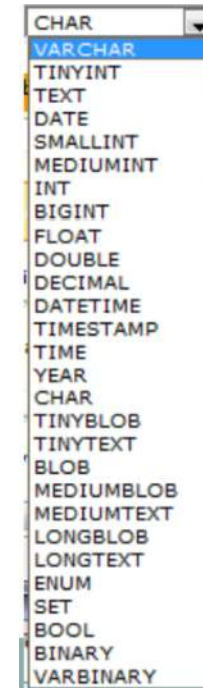
Perintah DML

- DML = Data Manipulation Language
- Digunakan untuk memanipulasi data
- Contoh :
 - ☐ SELECT mengambil data
 - ☐ DELETE menghapus data
 - ☐ INSERT menyisipkan data
 - ☐ UPDATE mengubah data

Tipe Data di MySQL

- Dalam bahasa SQL pada umumnya informasi tersimpan dalam tabel-tabel yang secara logik merupakan struktur dua dimensi terdiri dari baris (row atau record) dan kolom (column atau field). Sedangkan dalam sebuah database dapat terdiri dari beberapa table.
- Beberapa tipe data dalam MySQL yang sering dipakai:

Tipe data	Keterangan
INT(M) [UNSIGNED]	Angka -2147483648 s/d 2147483647
FLOAT(M,D)	Angka pecahan
DATE	Tanggal Format : YYYY-MM-DD
DATETIME	Tanggal dan Waktu Format : YYYY-MM-DD HH:MM:SS
CHAR(M)	String dengan panjang tetap sesuai dengan yang ditentukan. Panjangnya 1-255 karakter
VARCHAR(M)	String dengan panjang yang berubah-ubah sesuai dengan yang disimpan saat itu. Panjangnya 1 – 255 karakter
BLOB	Teks dengan panjang maksimum 65535 karakter
LONGBLOB	Teks dengan panjang maksimum 4294967295 karakter



Penjelasan Tipe Data

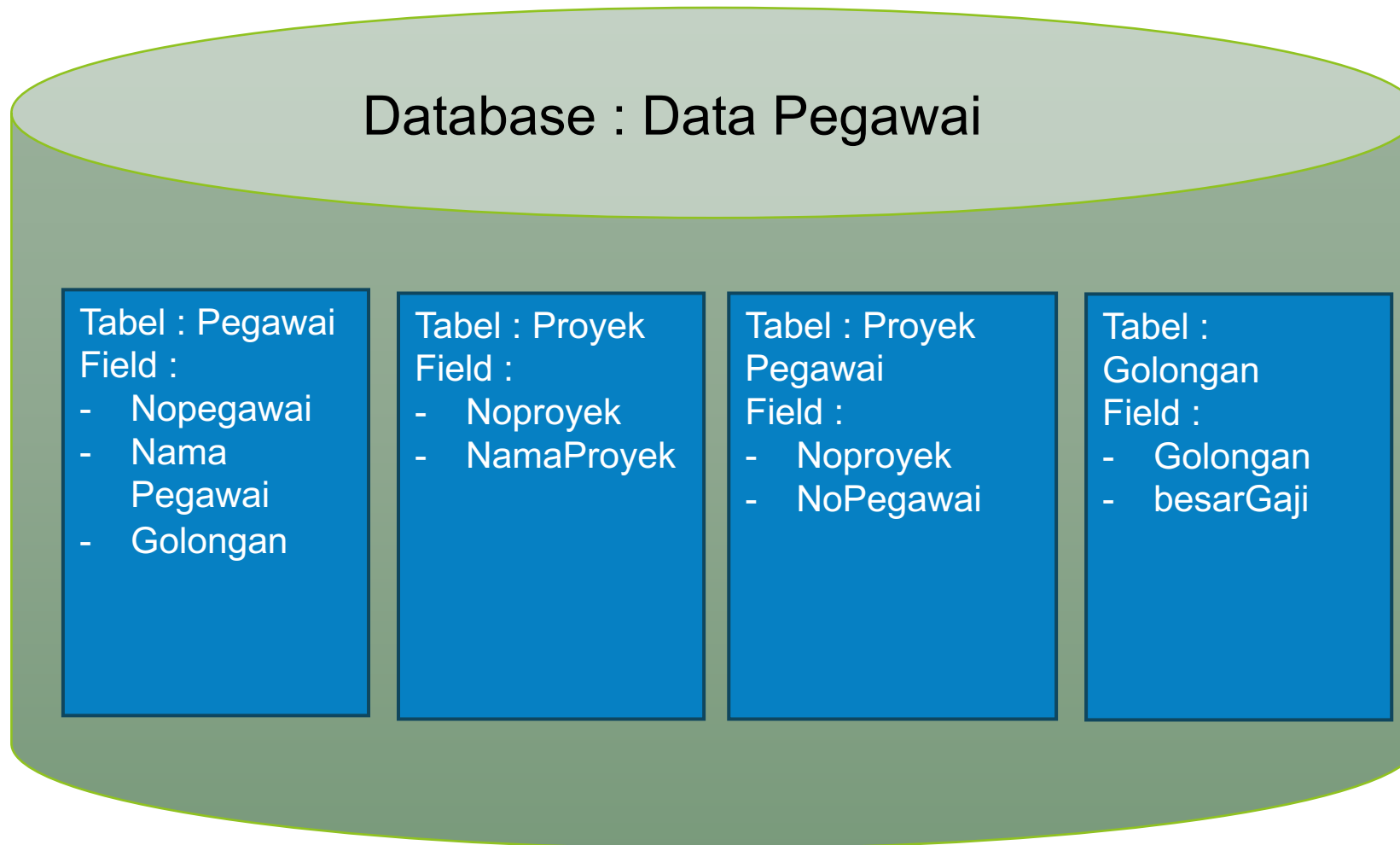
- **Karakter**

- CHAR : Teks dengan maksimal 255 karakter
- VARCHAR : Teks maksimal 255 karakter dan bersifat variabel
- TEXT : Teks dengan panjang maksimal 65535

- **Bilangan**

- TINYINT : Bilangan 1 byte
- SMALLINT : Bilangan 2 byte
- INT atau INTEGER : Bilangan 4 byte
- BIGINT : Bilangan 8 byte
- FLOAT : Bilangan pecahan (4 byte)
- DOUBLE atau REAL : Bilangan pecahan (8 byte)
- DECIMAL(M,D) atau NUMERIC(M,D) : Bilangan pecahan

Pembuatan Database dan Tabel



DDL

Data Definition Language

Data definition language (DDL)

- `CREATE/DROP DATABASE dbname;`
- `SHOW DATABASES;`
- `USE dbname;`
- `CREATE TABLE table_name (field_name type,..., constraints,...);`
- `SHOW TABLES;`
- `SHOW COLUMNS FROM table name;`
- `SHOW COLUMNS FROM table_name;`
- `DROP TABLE table_name;`

Membuat Database dan Tabel

- Cara untuk membuat sebuah database baru adalah dengan perintah:

```
create database namadatabase;
```

Contoh: **create database privatdb;**

- Untuk membuka sebuah database dapat menggunakan perintah berikut ini:

```
use namadatabase;
```

Contoh: **use privatdb;**

- Perintah untuk membuat tabel baru adalah:

```
create table namatabel (  
    struktur  
);
```

Membuat Database dan Tabel

- Berikan perintah pada prompt mysql :

```
create database pegawai;
```

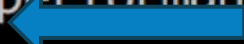
```
mysql> create database pegawai;  
Query OK, 1 row affected (0.00 sec)  
  
mysql> █
```

Menampilkan Database

- Perintah untuk menampilkan seluruh database yang ada di sistem

`show databases;`

```
mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema      |
| mysql                   |
| pegawai                  |
| performance_schema      |
| sys                     |
+-----+
5 rows in set (0.00 sec)
```



Memilih Database

- Untuk memilih database mana yang akan digunakan, berikan perintah berikut ini :

USE namadatabase;

Contoh : **USE pegawai;**

Pegawai adalah nama database yang akan digunakan.

Menghapus database

- Untuk menghapus database dapat menggunakan perintah :

```
drop database nama_database;
```

Contoh : bila ingin menghapus database pegawai karena sudah tidak digunakan, maka perintahnya sebagai berikut :

```
drop database pegawai;
```

Menampilkan Isi Database

- Isi dari database adalah berupa kumpulan tabel-tabel
- Untuk melihat tabel apa saja yang ada didalam database, maka berikan perintah berikut :

```
show tables;
```

Membuat Tabel Baru

- Untuk membuat tabel baru didalam sebuah database, maka berikan perintah :

```
create table nama_tabel (nama_kolom tipe_data  
(panjang data) key)
```

- Menampilkan struktur table

```
describe nama_tabel;
```

- Menghapus tabel

```
drop table nama_tabel;
```


Contoh : Membuat Tabel Pribadi

```
Create tabel pribadi (  
    nip char(5) not null primary key,  
    nama varchar (35) not null,  
    tgl_lahir date,  
    sex enum('p','w')  
    alamat varchar (35),  
    kota varchar (15));
```

Constraints

- Not Null
tidak boleh berisi NULL (kosong)
- UNIQUE
satu data dengan data lainnya tidak boleh sama
- PRIMARY KEY
- FOREIGN KEY
sebagai relasi antara 2 tabel
- AUTO_INCREMENT
nilai naik secara otomatis tanpa diisi

Penambahan Data

- Penambahan data dilakukan dengan menggunakan pernyataan INSERT
- Bentuk dasar :

```
INSERT INTO nama_tabel(nama_field1, nama_field2,...)  
values (nilai1, nilai2,...);
```

Contoh :

```
INSERT INTO pribadi(nip,nama,tgl_lahir,sex,alamat,kota)  
values ('001','yoyon','1965/10/10','P','jl. Kutisari  
67','surabaya');
```

Melihat Isi Tabel

- Untuk melihat isi dari tabel yang sudah diinputkan, dapat menggunakan perintah SELECT

select * from nama_tabel

contoh :

select * from pribadi

```
mysql> select * from pribadi;
+-----+-----+-----+-----+-----+-----+
| nip | nama | tgl_lahir | sex | alamat | kota |
+-----+-----+-----+-----+-----+-----+
| 001 | yoyon | 1965-10-10 | p | jl. kutisari 67 | surabaya |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql> █
```

Melihat Struktur Tabel

- Gunakan perintah :

DESC nama_tabel;

Contoh :

DESC pribadi;

```
mysql> desc pribadi;
```

Field	Type	Null	Key	Default	Extra
nip	char(5)	NO	PRI	NULL	
nama	varchar(35)	NO		NULL	
tgl_lahir	date	YES		NULL	
sex	enum('p','w')	YES		NULL	
alamat	varchar(35)	YES		NULL	
kota	varchar(15)	YES		NULL	

6 rows in set (0.01 sec)

Mengganti Nama field

- Perintah yang digunakan adalah **ALTER TABLE**

Contoh :

```
ALTER TABLE Pribadi CHANGE sex kelamin ENUM('p','w');
```

```
mysql> desc pribadi;
```

Field	Type	Null	Key	Default	Extra
nip	char(5)	NO	PRI	NULL	
nama	varchar(35)	NO		NULL	
tgl_lahir	date	YES		NULL	
kelamin	enum('P','W')	YES		NULL	
alamat	varchar(35)	YES		NULL	
kota	varchar(15)	YES		NULL	

6 rows in set (0.00 sec)

Mengganti Ukuran / Tipe Field

- Perintah yang digunakan adalah **alter table**
- Contoh :

```
ALTER TABLE pribadi MODIFY kota VARCHAR(20);
```

```
mysql> desc pribadi;
```

Field	Type	Null	Key	Default	Extra
nip	char(5)	NO	PRI	NULL	
nama	varchar(35)	NO		NULL	
tgl_lahir	date	YES		NULL	
kelamin	enum('P','W')	YES		NULL	
alamat	varchar(35)	YES		NULL	
kota	varchar(20)	YES		NULL	


6 rows in set (0.01 sec)

Menambahkan Default

- Default pada struktur tabel digunakan untuk memberikan nilai bawaan pada suatu field kalau nilai bersangkutan tidak dimasukkan
- Contoh :

```
ALTER TABLE pribadi CHANGE kelamin kelamin ENUM('P','W')DEFAULT 'P';
```

```
mysql> desc pribadi;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nip        | char(5)       | NO   | PRI | NULL    |       |
| nama       | varchar(35)   | NO   |     | NULL    |       |
| tgl_lahir  | date          | YES  |     | NULL    |       |
| kelamin    | enum('P','W') | YES  |     | P       |       |
| alamat     | varchar(35)   | YES  |     | NULL    |       |
| kota       | varchar(20)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```



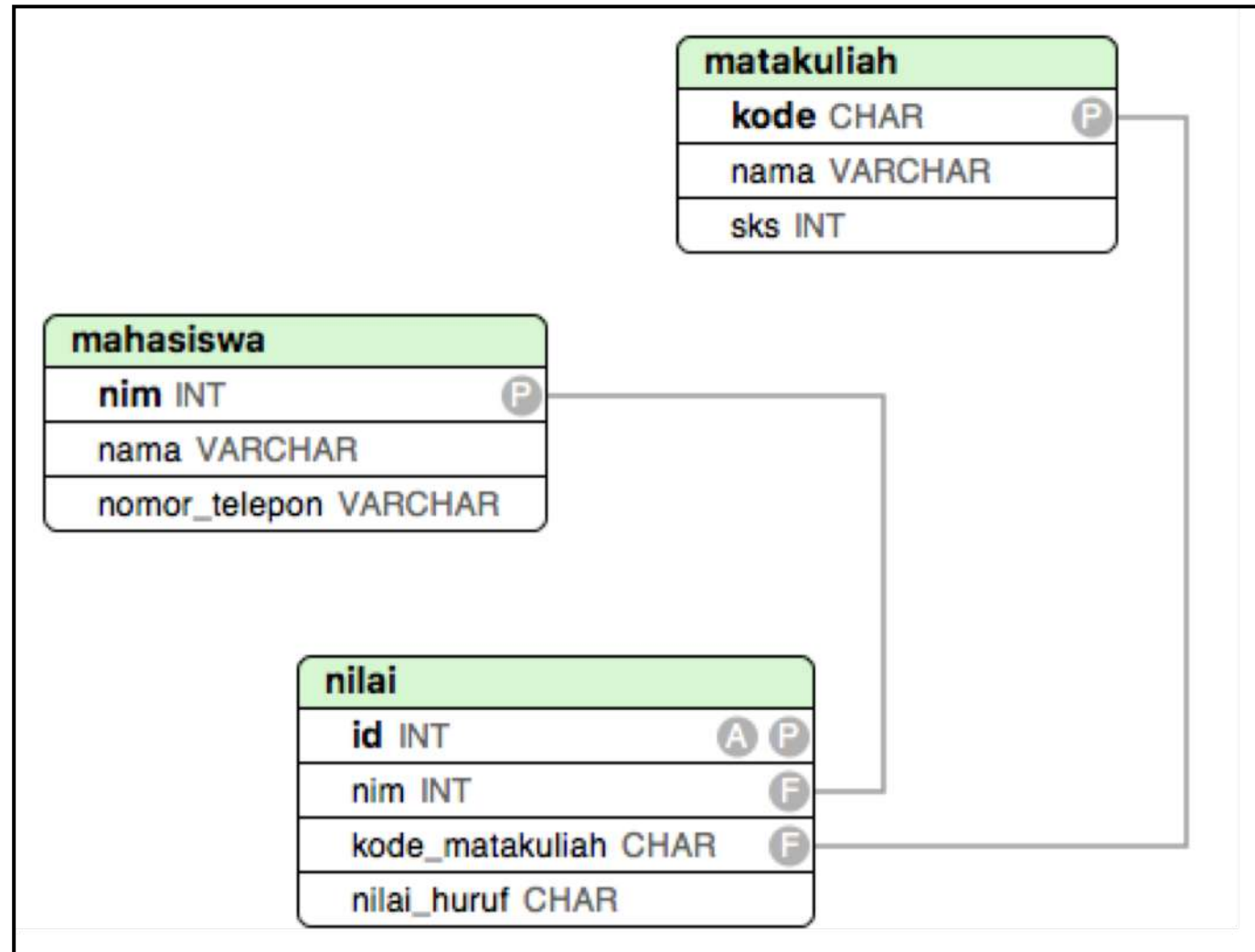
DML

Data Manipulation Language

DML vs DDL

- Adalah BAHASA yang digunakan untuk memerintahkan DBMS agar melakukan operasi-operasi yang sifatnya **MENGUBAH** nilai-nilai data pada (**ISI**) **tabel**.
- Yang diubah oleh:
 - DDL ☐ **Struktur** tabel.
 - DML ☐ **Isi** tabel.
- Perspektif:
 - DDL ☐ **TABEL**.
 - DML ☐ **BARIS**/Row/Record/Tuple.
- Ada 3 klausa utama:
 - INSERT : **Menambahkan** suatu BARIS baru.
 - UPDATE : **Mengganti** nilai pada suatu BARIS.
 - DELETE : **Menghapus** suatu BARIS.
- Dan 1 klausa syarat (filtering): **WHERE**

Database Akademik



INSERT

- Digunakan untuk **menambahkan RECORD/Baris** baru pada suatu tabel.
- Klausa pembentuk: **INSERT, INTO, VALUES**
- Format:
 1. **INSERT INTO** nama_tabel (kolom1, kolom2, ...dst.) **VALUES** (nilai_kolom1, nilai_kolom2, ...dst.);
 2. **INSERT INTO** nama_tabel **VALUES** (nilai_kolom1, nilai_kolom2, ...dst.);

INSERT

```
INSERT INTO mahasiswa (nim, nama) VALUES (1, 'Ani Rahmawati');
```

nim	nama	nomor_telepon
1	Ani Rahmawati	NULL

```
INSERT INTO mahasiswa VALUES (2, 'Budi Raharjo', '0858776453');
```

nim	nama	nomor_telepon
1	Ani Rahmawati	NULL
2	Budi Raharjo	0858776453

INSERT

```
INSERT INTO mahasiswa VALUES  
  (3, 'Charlie Setiabudi', '0859767553'),  
  (4, 'Diandra Paramita', '0858998745');
```

nim	nama	nomor_telepon
1	Ani Rahmawati	NULL
2	Budi Raharjo	0858776453
3	Charlie Setiabudi	0859767553
4	Diandra Paramita	0858998745

Klausa 'WHERE'

- **WHERE** digunakan pada statement-statement UPDATE, DELETE, dan SELECT sebagai **filter/pembatas** terhadap **hasil** yang dikembalikan.
- Format:
 - `[Statement Utama] WHERE kolom_patokan [operator_perbandingan] nilai_patokan;`
 - `[Statement Utama] WHERE kolom_patokan1 [operator_perbandingan1] nilai_patokan1 [operator_logika1] kolom_patokan2 [operator_perbandingan2] nilai_patokan2 [operator_logika2] ...dst.;`
- **Operator perbandingan/comparison operator** dapat berupa:
 - `=, <, >, <=, >=, <>`
- **Operator logika** dapat berupa:
 - **AND, OR**
- Contoh:
 - `SELECT * FROM matakuliah WHERE kode = 'ASD' ;`
 - `UPDATE matakuliah SET sks = 2 WHERE nama = 'Kecerdasan Buatan';`
 - `DELETE FROM matakuliah WHERE kode = 'SPK';`
 - `DELETE FROM matakuliah WHERE kode = 'SPK' OR kode = 'ASD';`

UPDATE

- Digunakan untuk **mengubah/mengganti** nilai **RECORD/Baris** yang sudah ada pada suatu tabel.
- Klausa pembentuk: **UPDATE, SET, WHERE**
- Format:
 1. `UPDATE nama_tabel SET nama_kolom = nilai_baru WHERE nama_kolom_patokan [operator_perbandingan] nilai_patokan;`
 2. `UPDATE nama_tabel SET nama_kolom1 = nilai_baru1, nama_kolom2 = nilai_baru2, ...dst. WHERE nama_kolom_patokan [operator_perbandingan] nilai_patokan;`
- **Operator perbandingan/comparison operator** dapat berupa:
 - `=, <, >, <=, >=, <>`

UPDATE

```
UPDATE mahasiswa SET nomor_telepon = '0857550234'  
WHERE nim = 1;
```

nim	nama	nomor_telepon
1	Ani Rahmawati	0857550234
2	Budi Raharjo	0858776453
3	Charlie Setiabudi	0859767553
4	Diandra Paramita	0858998745

DELETE

- Digunakan untuk **menghapus** suatu **RECORD/Baris** yang sebelumnya ada pada suatu tabel.
- Klausa pembentuk: **DELETE, FROM, WHERE**
- Format:
 1. `DELETE FROM nama_tabel WHERE nama_kolom_patokan [operator_perbandingan] nilai_patokan;`
 2. `DELETE * FROM nama_tabel; atau DELETE FROM nama_tabel;`
- **Operator perbandingan**/comparison operator dapat berupa:
 - `=, <, >, <=, >=, <>`

DELETE

```
DELETE FROM matakuliah WHERE kode = 'BDD';
```

kode	nama	sks
ASD	Algoritma dan Struktur Data	NULL
KCB	Kecerdasan Buatan	NULL
MMT	Multimedia Terapan Tingkat Lanjut	3
PBO	Pemrograman Berorientasi Objek	NULL
SPK	Sistem Pendukung Keputusan	NULL

```
SET SQL_SAFE_UPDATES = 0;  
DELETE FROM matakuliah;  
SET SQL_SAFE_UPDATES = 1;
```

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

```
mysql> █
```

DML

DATABASE AKADEMIK: DDL

```
MariaDB [akademik]> select * from mahasiswa;
```

nim	nama	nomor_telepon
171401	Ani Rahmawati	0858776453
171402	Budi Raharjo	NULL
171403	Charlie Setiabudi	0813543226
171404	Diandra Paramita	NULL

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

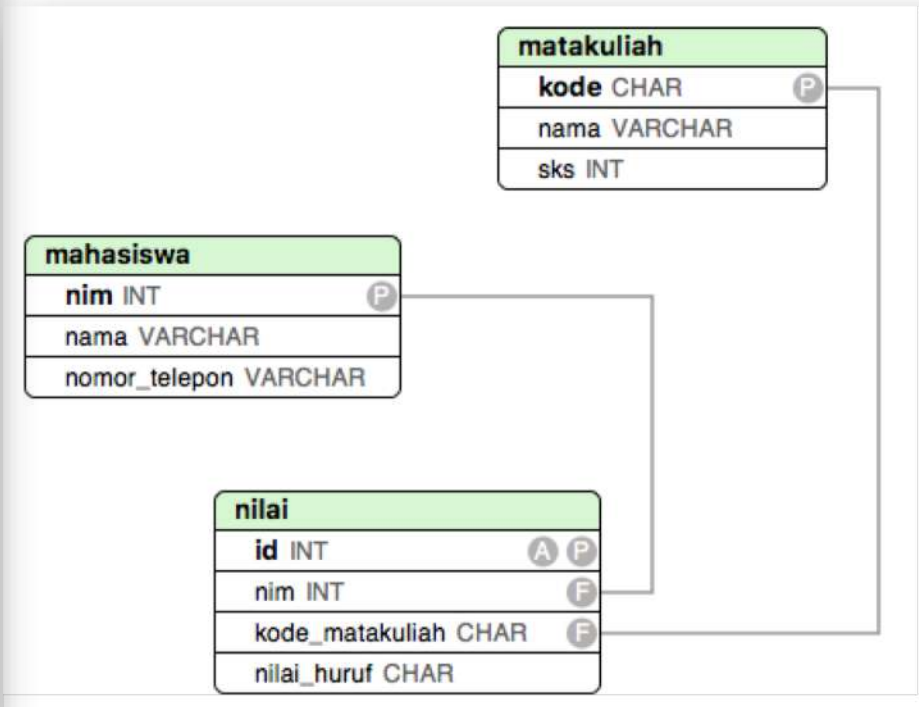
```
MariaDB [akademik]> select * from matakuliah;
```

kode	nama	sks
ASD	Algoritma dan Struktur Data	NULL
BDD	Basis Data Dasar	NULL
KCB	Kecerdasan Buatan	NULL
MMT	Multimedia Terapan	NULL
PBO	Pemrograman Berorientasi Objek	NULL
SPK	Sistem Pendukung Keputusan	NULL

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

```
MariaDB [akademik]> select * from nilai;
```

id	nim	kode_matakuliah	nilai_huruf
1	171402	ASD	A
2	171402	SPK	C
3	171401	ASD	B
4	171401	SPK	B
5	171403	ASD	A
6	171403	SPK	A
7	171403	BDD	A



QUERY

- Query merupakan operasi yang melibatkan satu atau lebih tabel untuk melakukan retrieval data.
- Retrieval data dilakukan dengan beberapa cara berikut :

SELECT * untuk memilih semua kolom

SELECT dengan WHERE untuk menampilkan baris dengan suatu kondisi

SELECT dengan DISTINCT untuk menampilkan data dengan eliminasi data yang sama (duplicate)

SELECT dengan IN untuk menampilkan data yang spesifik

SELECT dengan BETWEEN untuk menampilkan data pada jarak (range) tertentu

SELECT dengan LIKE untuk menampilkan data yang memiliki kemiripan dengan keyword yang diinginkan

SELECT dengan GROUP BY untuk menampilkan susunan data dalam bentuk grup

SELECT dengan ORDER BY untuk menampilkan baris secara spesifik dan terurut maju atau mundur

SELECT dengan AND, OR and NOT untuk menampilkan data dengan kondisi dan, atau, tidak

SELECT dengan UNION, INTERSECT dan EXCEPT untuk menampilkan data dengan operasi himpunan yang melibatkan lebih dari satu tabel

QUERY (SELECT-WHERE)

Format :

```
SELECT column1, column2,...  
FROM table_name  
WHERE condition;
```

```
SELECT nim, kode_matakuliah  
FROM nilai  
WHERE nilai_huruf = 'A'
```

Akan menghasilkan:

nim	kode_matakuliah
171402	ASD
171403	ASD
171403	SPK
171403	BDD

QUERY (SELECT-DISTINCT)

Format :

```
SELECT DISTINCT column FROM table_name;
```

```
SELECT DISTINCT nilai_huruf  
FROM nilai;
```

Akan menghasilkan:

nilai_huruf
A
C
B

QUERY (SELECT-IN)

Format :

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name IN (value1, value2, ...);
```

```
SELECT *  
FROM nilai  
WHERE nilai_huruf IN ('A','C');
```

Akan menghasilkan:

id	nim	kode_matakuliah	nilai_huruf
1	171402	ASD	A
2	171402	SPK	C
5	171403	ASD	A
6	171403	SPK	A
7	171403	BDD	A

QUERY (SELECT-BETWEEN)

Format :

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name BETWEEN value1 AND value2;
```

```
SELECT *  
FROM nilai  
WHERE nilai_huruf  
BETWEEN 'B' AND 'D';
```

Akan menghasilkan:

id	nim	kode_matakuliah	nilai_huruf
2	171402	SPK	C
3	171401	ASD	B
4	171401	SPK	B

QUERY (SELECT-LIKE)

Format :

```
SELECT column1, column2, ...  
FROM table_name  
WHERE columnN LIKE pattern;
```

```
SELECT *  
FROM mahasiswa  
WHERE nama like '%ra%';
```

Akan menghasilkan:

nim	nama	nomor_telepon
171401	Ani Rahmawati	0858776453
171402	Budi Raharjo	NULL
171404	Diandra Paramita	NULL

QUERY (SELECT-GROUP BY)

Format :

```
SELECT column1, column2,...  
condition  
FROM table_name  
GROUP BY column1, column2, ...;
```

```
SELECT nim, kode_matakuliah, nilai_huruf  
FROM nilai  
GROUP BY nilai_huruf
```

Akan menghasilkan:

nim	kode_matakuliah	nilai_huruf
171402	ASD	A
171401	ASD	B
171402	SPK	C

QUERY (SELECT-ORDER BY)

Format :

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... ASC|DESC;
```

```
SELECT kode, nama  
FROM matakuliah  
ORDER BY nama DESC
```

Akan menghasilkan:

kode	nama
SPK	Sistem Pendukung Keputusan
PBO	Pemrograman Berorientasi Objek
MMT	Multimedia Terapan
KCB	Kecerdasan Buatan
BDD	Basis Data Dasar
ASD	Algoritma dan Struktur Data

QUERY (SELECT-AND|OR|NOT)

Format :

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND|OR|NOT condition2 AND|OR|NOT condition3 ...;
```

```
SELECT *  
FROM nilai  
WHERE nim = 171401  
AND kode_matakuliah = 'ASD';
```

Akan menghasilkan:

id	nim	kode_matakuliah	nilai_huruf
3	171401	ASD	B

QUERY

(SELECT-UNION|INTERSECT|EXCEPT)

Format :

```
SELECT column_name(s) FROM table1  
UNION|INTERSECT|EXCEPT  
SELECT column_name(s) FROM table2;
```

```
SELECT kode  
FROM matakuliah  
UNION  
SELECT kode_matakuliah  
FROM nilai;
```

Akan menghasilkan:

kode
ASD
BDD
KCB
MMT
PBO
SPK

QUERY

(SELECT-UNION|INTERSECT|EXCEPT)-ALL

Format :

```
SELECT column_name(s) FROM table1  
UNION|INTERSECT|EXCEPT-ALL  
SELECT column_name(s) FROM table2;
```

```
SELECT kode  
FROM matakuliah  
UNION ALL  
SELECT kode_matakuliah  
FROM nilai;
```

Akan menghasilkan:

kode
ASD
BDD
KCB
MMT
PBO
SPK
ASD
ASD
ASD
BDD
SPK
SPK
SPK

SUB-QUERY

- Sub-query adalah adanya query di dalam query lain. Sub-query juga disebut dengan perintah SELECT bersarang (nested SELECT).
- Retrival data dalam sub-query dilakukan oleh SELECT untuk menemukan data pada SELECT utama.
- Kegunaan utama :
 - Test Keaggotaan
 - Perbandingan Himpunan

SUB-QUERY (TES KEANGGOTAAN)

Format :

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name IN (SELECT STATEMENT);
```

```
SELECT nim, kode_matakuliah, nilai_huruf  
FROM nilai  
WHERE nilai_huruf  
in (SELECT MIN(nilai_huruf) FROM nilai);
```

Akan menghasilkan:

nim	kode_matakuliah	nilai_huruf
171402	ASD	A
171403	ASD	A
171403	SPK	A
171403	BDD	A

SUB-QUERY (PERBANDINGAN HIMPUNAN)

Format :

```
SELECT column_name(s)  
FROM table_name  
operator  
(SELECT STATEMENT);
```

```
SELECT nim, kode_matakuliah, nilai_huruf  
FROM nilai  
WHERE nilai_huruf > ALL  
(SELECT nilai_huruf FROM nilai  
WHERE nilai_huruf = 'A');
```

Akan menghasilkan:

nim	kode_matakuliah	nilai_huruf
171402	SPK	C
171401	ASD	B
171401	SPK	B

referensi

- Dwi Puspitasari, S.Kom, “**Buku Ajar Dasar Basis Data**”, *Program Studi Manajemen Informatika Politeknik Negeri Malang*, 2012.
- Fathansyah, “**Basisdata Revisi Kedua**”, Bandung: Informatika, 2015.
- <https://www.w3schools.com>



Aggregate: Function

Web Developer

Prodi Teknologi Informasi
Fakultas Teknik
Universitas Tidar Magelang

OUTLINE

1. Pengertian Aggregate Function
2. Tipe Aggregate Function
3. Fungsi COUNT
4. Fungsi SUM
5. Fungsi AVG
6. Fungsi MIN
7. Fungsi MAX

Apa Itu Aggregate Function??

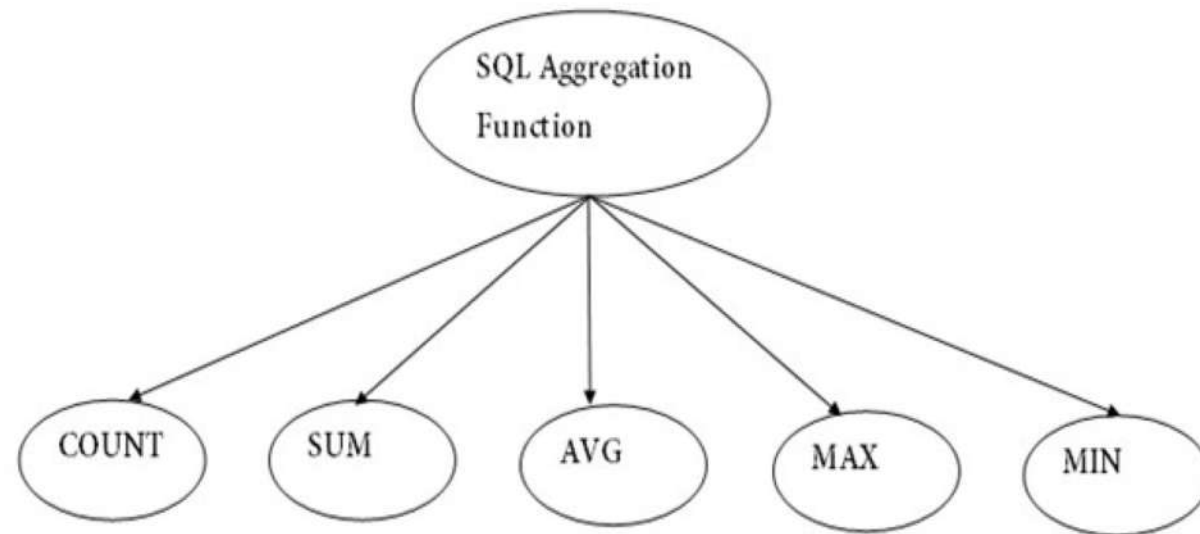
“

Fungsi agregat dalam SQL mengembalikan satu nilai setelah menghitung beberapa nilai kolom. Kita sering menggunakan fungsi agregat dengan klausa GROUP BY dan HAVING dari

”

pernyataan SELECT.

Type Aggregate Function



Fungsi COUNT()

- **Apa itu Fungsi COUNT()**

- Fungsi COUNT() dalam bahasa kueri standar (SQL) adalah fungsi agregat yang mengembalikan jumlah total data yang dikembalikan oleh kueri SELECT berdasarkan kondisi yang ditentukan.
- Fungsi COUNT(*) menghitung semua data.
- Fungsi COUNT() hanya menghitung data non-NULL.
- Fungsi COUNT(), ketika digunakan dengan kata kunci DISTINCT, mengembalikan hitungan hanya data unik dalam kumpulan hasil dari pernyataan SELECT.
- Fungsi mengembalikan 0 jika tidak ada data yang cocok.



Cara penggunaan fungsi COUNT():

```
SELECT COUNT(expression)
FROM table_name;
```

```
SELECT COUNT(DISTINCT column_name)
FROM table_name;
```

```
SELECT count(column_name1), column_name2
FROM table_name
GROUP BY column_name2;
```

Fungsi COUNT()

Contoh 1 - Menggunakan ekspresi (*)

Tabel produk

id_produk	nama_produk	harga_produk	merk_produk
1	Susu	20000	Dancow
2	Susu	30000	Milo
3	Popok	50000	Makuku
4	Popok	75000	Pampers

Code:

```
MariaDB [dts_tsa]> SELECT COUNT(*)  
-> FROM produk;
```

```
MariaDB [dts_tsa]> SELECT COUNT(1)  
-> from produk  
-> ;
```

Output:

COUNT(*)
4

Contoh 2 - Menggunakan nama kolom (ada null)

Tabel produk

id_produk	nama_produk	harga_produk	merk_produk
1	Susu	20000	Dancow
2	Susu	30000	Milo
3	Popok	50000	Makuku
4	Popok	75000	NULL

Code:

```
MariaDB [dts_tsa]> SELECT COUNT(merk_produk)  
-> FROM produk;
```

Output:

COUNT(merk_produk)
3

Fungsi COUNT()

Contoh 3 - Menggunakan nama kolom tanpa null

Tabel produk

id_produk	nama_produk	harga_produk	merk_produk
1	Susu	20000	Dancow
2	Susu	30000	Milo
3	Popok	50000	Makuku
4	Popok	75000	NULL

Code:

```
MariaDB [dts_tsa]> SELECT COUNT(id_produk)  
-> from produk;
```

Output:

COUNT(id_produk)
4

Contoh 4 - Menggunakan nama kolom dan memberikan nama alias

Tabel produk

id_produk	nama_produk	harga_produk	merk_produk
1	Susu	20000	Dancow
2	Susu	30000	Milo
3	Popok	50000	Makuku
4	Popok	75000	NULL

Code:

```
MariaDB [dts_tsa]> SELECT COUNT(merk_produk)  
-> as "jumlah merk"  
-> from produk;
```

Output:

jumlah merk
3

Fungsi SUM()

- **Apa itu Fungsi SUM()**

- SQL SUM() adalah salah satu fungsi agregat yang tersedia di SQL yang membantu kita mengambil nilai total di antara beberapa nilai yang ditentukan dalam nilai kolom catatan, ekspresi yang terdiri dari kolom yang disebutkan.
- Saat kueri digunakan untuk mengambil data yang terkait laporan dan berisi pernyataan kelompok demi kelompok, fungsi SUM() digunakan untuk mendapatkan nilai total kolom atau kolom tertentu berdasarkan fungsi pengelompokan.

Cara penggunaan fungsi SUM():

```
SELECT SUM(expression)
FROM table_name
[WHERE restriction];
```

Fungsi SUM()

Contoh 1 - Menggunakan 1 kolom

Tabel number

num
50
100
150
200
50

Code:

```
SELECT SUM(num) FROM numbers ;
```

Output:

sum(num)
550

Contoh 2 - Menggunakan fungsi distinct

Code:

```
SELECT SUM(DISTINCT(num)) FROM numbers ;
```

Output:

SUM(DISTINCT(num))
500

Fungsi SUM()

Contoh 1 - Menggunakan 1 kolom

Tabel number

num
50
100
150
200
50

Code:

```
SELECT SUM(num) FROM numbers ;
```

Output:

sum(num)
550

Contoh 2 - Menggunakan fungsi distinct

Code:

```
SELECT SUM(DISTINCT(num)) FROM numbers ;
```

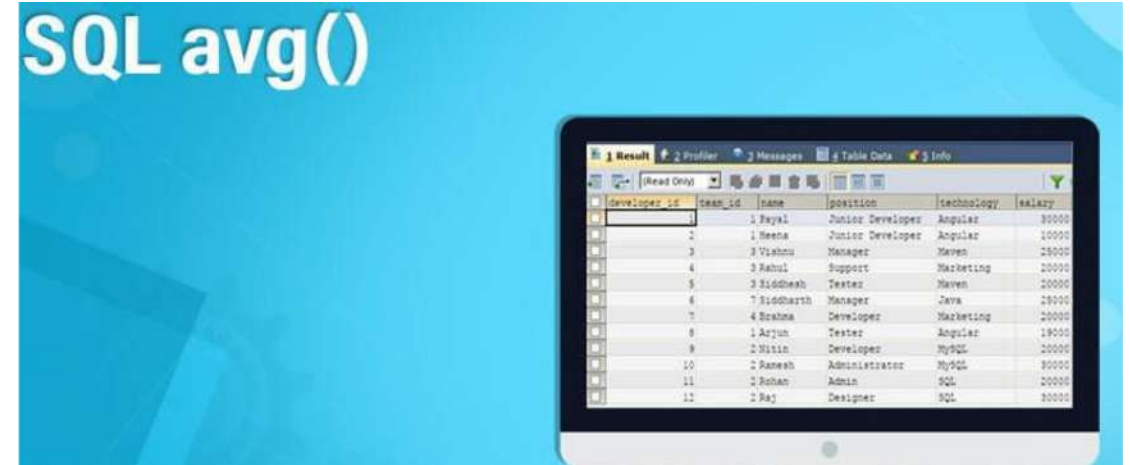
Output:

SUM(DISTINCT(num))
500

Fungsi AVG()

• Apa itu Fungsi AVG()

- SQL avg() adalah salah satu fungsi agregat yang tersedia di SQL yang membantu kita mengambil nilai rata-rata di antara beberapa nilai yang ditentukan dalam nilai kolom, ekspresi yang terdiri dari kolom yang disebutkan.
- Seringkali, ketika kueri digunakan untuk mengambil data yang terkait dengan laporan dan berisi pernyataan kelompok demi kelompok, fungsi avg() digunakan untuk mendapatkan nilai rata-rata kolom atau kolom tertentu berdasarkan fungsi pengelompokan.



Cara penggunaan fungsi AVG():

```
SELECT AVG(expression)
FROM table_name
[WHERE restriction];
```

- **Ekspresi** dapat berupa nama kolom tabel atau rumus yang dibuat menggunakan nama kolom dan nilai atau variabel literal statis.
- **Nama_tabel** adalah nama tabel tempat Anda ingin mengambil catatan dan menghitung nilai rata-rata dari salah satu kolomnya.
- Satu hal yang opsional adalah penggunaan **klausula where** untuk menyebutkan kondisi dan batasan yang harus dipenuhi oleh catatan tabel untuk mempertimbangkan nilai kolom catatan itu untuk menghitung nilai

Fungsi AVG()

Contoh 1 - Menggunakan 1 kolom

Tabel produk

Code:

num
50
100
150
200
50

```
MariaDB [dts_tsa]> SELECT AVG(num)
-> FROM number;
```

Output:

AVG(num)
110.0000

Contoh 2 - Menggunakan fungsi distinct

Code:

```
MariaDB [dts_tsa]> SELECT AVG(DISTINCT(num))
-> from number;
```

Output:

AVG(DISTINCT(num))
125.0000

Fungsi AVG()

Contoh 3 - Menggunakan Formula

Tabel number

num
50
100
150
200
50

Code:

```
MariaDB [dts_tsa]> SELECT AVG((num * 10) + 1)  
-> FROM number;
```

Output:

AVG((num * 10) + 1)
1101.0000

Contoh 4 - Menggunakan group by

id_produk	nama_produk	harga_produk	merk_produk
1	Susu	20000	Dancow
2	Susu	30000	Milo
3	Popok	50000	Makuku
4	Popok	75000	Pampers

Code:

```
MariaDB [dts_tsa]> SELECT  
-> nama_produk,  
-> AVG(harga_produk)  
-> FROM  
-> produk  
-> GROUP BY nama_produk;
```

Output:

nama_produk	AVG(harga_produk)
Popok	62500
Susu	25000

Fungsi MIN()

- **Apa itu Fungsi MIN()**

- Fungsi MIN() dalam bahasa kueri standar (SQL) adalah fungsi agregat yang mengembalikan nilai terkecil atau minimum dari kolom tertentu yang diperoleh dalam kumpulan hasil kueri SELECT.
- Fungsi MIN() dapat digunakan sendiri dalam kueri SELECT atau digunakan dengan klausa GROUP BY dan HAVING untuk menyiapkan tabel ringkasan dalam analitik data.



Cara penggunaan fungsi MIN():

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

- **column_name**: Bidang atau kolom yang ingin kita kembalikan nilai minimumnya.
- **table_name**: Tabel database dari mana nama_kolom harus diambil.
- **Kondisi WHERE**: Klausa pemfilteran apa pun untuk baris jika diperlukan.

Fungsi MIN()

Contoh 1 - Menggunakan fungsi dasar MIN()

Tabel produk

	product_id	sale_date	sale_amount	salesperson	store_state
1	1	06.05.2020 00:00:00	2300	X	DL
2	2	06.05.2020 00:00:00	5300	Y	DL
3	3	06.05.2020 00:00:00	300	X	MH
4	4	07.05.2020 00:00:00	4200	Y	MH
5	5	07.05.2020 00:00:00	900	Y	MH
6	6	05.05.2020 00:00:00	600	X	DL
7	7	05.05.2020 00:00:00	1450	Y	MH
8	8	05.05.2020 00:00:00	987	X	MH

Code:

```
SELECT MIN(sale_amount)
FROM sales;
```

Output:

	(No column name)
1	300

Fungsi MIN()

Contoh 2 - Menggunakan klausa WHERE

Tabel produk

	product_id	sale_date	sale_amount	salesperson	store_state
1	1	06.05.2020 00:00:00	2300	X	DL
2	2	06.05.2020 00:00:00	5300	Y	DL
3	3	06.05.2020 00:00:00	300	X	MH
4	4	07.05.2020 00:00:00	4200	Y	MH
5	5	07.05.2020 00:00:00	900	Y	MH
6	6	05.05.2020 00:00:00	600	X	DL
7	7	05.05.2020 00:00:00	1450	Y	MH
8	8	05.05.2020 00:00:00	987	X	MH

Code:

```
SELECT MIN(sale_amount) as "Minimum sale"  
FROM sales  
WHERE store_state = 'DL';
```

Output:

	Minimum sale
1	600

Fungsi MIN()

Contoh 3 - Menggunakan klausa GROUP BY

Tabel produk

	product_id	sale_date	sale_amount	salesperson	store_state
1	1	06.05.2020 00:00:00	2300	X	DL
2	2	06.05.2020 00:00:00	5300	Y	DL
3	3	06.05.2020 00:00:00	300	X	MH
4	4	07.05.2020 00:00:00	4200	Y	MH
5	5	07.05.2020 00:00:00	900	Y	MH
6	6	05.05.2020 00:00:00	600	X	DL
7	7	05.05.2020 00:00:00	1450	Y	MH
8	8	05.05.2020 00:00:00	987	X	MH

Code:

```
SELECT store_state, MIN(sale_amount) as "Minimum sale"  
FROM sales  
GROUP BY store_state;
```

Output:

	store_state	Minimum sale
1	DL	600
2	MH	300

Fungsi MIN()

Contoh 4 - Mencari nilai terendah dari masing-masing sales person

Tabel produk

	product_id	sale_date	sale_amount	salesperson	store_state
1	1	06.05.2020 00:00:00	2300	X	DL
2	2	06.05.2020 00:00:00	5300	Y	DL
3	3	06.05.2020 00:00:00	300	X	MH
4	4	07.05.2020 00:00:00	4200	Y	MH
5	5	07.05.2020 00:00:00	900	Y	MH
6	6	05.05.2020 00:00:00	600	X	DL
7	7	05.05.2020 00:00:00	1450	Y	MH
8	8	05.05.2020 00:00:00	987	X	MH

Code:

```
SELECT salesperson, MIN(sale_amount) as "Minimum sale"  
FROM sales  
GROUP BY salesperson;
```

Output:

	salesperson	Minimum sale
1	X	300
2	Y	900

Contoh 5 - Menggunakan Klausa HAVING

Tabel produk

	product_id	sale_date	sale_amount	salesperson	store_state
1	1	06.05.2020 00:00:00	2300	X	DL
2	2	06.05.2020 00:00:00	5300	Y	DL
3	3	06.05.2020 00:00:00	300	X	MH
4	4	07.05.2020 00:00:00	4200	Y	MH
5	5	07.05.2020 00:00:00	900	Y	MH
6	6	05.05.2020 00:00:00	600	X	DL
7	7	05.05.2020 00:00:00	1450	Y	MH
8	8	05.05.2020 00:00:00	987	X	MH

Fungsi MIN()

Code:

```
SELECT salesperson, SUM(sale_amount) as "Total sales"  
FROM sales  
GROUP BY salesperson  
HAVING MIN(sale_date) = '2020-05-05';
```

Output:

	salesperson	Total sales
1	X	4187
2	Y	11850

Fungsi MIN()

Contoh 6 - Menggunakan Klausa ORDER BY

Tabel produk

	product_id	sale_date	sale_amount	salesperson	store_state
1	1	06.05.2020 00:00:00	2300	X	DL
2	2	06.05.2020 00:00:00	5300	Y	DL
3	3	06.05.2020 00:00:00	300	X	MH
4	4	07.05.2020 00:00:00	4200	Y	MH
5	5	07.05.2020 00:00:00	900	Y	MH
6	6	05.05.2020 00:00:00	600	X	DL
7	7	05.05.2020 00:00:00	1450	Y	MH
8	8	05.05.2020 00:00:00	987	X	MH

Code:

```
SELECT sale_date as "Date of sale",  
salesperson, MIN(sale_amount) as "Minimum sale"  
FROM sales  
GROUP BY sale_date, salesperson  
ORDER BY MIN(sale_amount);
```

Output:

	Date of sale	salesperson	Minimum sale
1	06.05.2020 00:00:00	X	300
2	05.05.2020 00:00:00	X	600
3	07.05.2020 00:00:00	Y	900
4	05.05.2020 00:00:00	Y	1450
5	06.05.2020 00:00:00	Y	5300

Fungsi MAX()

• Apa itu Fungsi MAX()

- SQL MAX() adalah salah satu fungsi agregat yang tersedia di SQL yang membantu kita mengambil nilai terbesar di antara beberapa nilai yang ditentukan dalam nilai kolom catatan, ekspresi yang terdiri dari kolom yang disebutkan.
- Saat kueri digunakan untuk mengambil data yang laporan terkait dan berisi grup dengan pernyataan, fungsi MAX() digunakan untuk mendapatkan nilai terbesar dari kolom atau kolom tertentu berdasarkan fungsi pengelompokan.



Cara penggunaan fungsi MIN():

```
SELECT MAX(expression)
FROM table_name
[WHERE restriction];
```

- **ekspresi** dapat berupa nama kolom tabel atau rumus yang dibuat menggunakan nama kolom dan nilai atau variabel literal statis,
- **nama_tabel** adalah nama tabel tempat Anda ingin mengambil catatan dan menghitung nilai terbesar dari salah satu kolom mereka.
- Satu hal opsional adalah penggunaan **klausula where** untuk menyebutkan kondisi dan batasan yang harus dipenuhi oleh catatan tabel untuk mempertimbangkan nilai kolom catatan itu untuk mengambil nilai terbesar.

Fungsi MAX()

Contoh 1 - Menggunakan fungsi dasar MAX()

Tabel numbers

	num
1	50
2	100
3	150
4	200

Code:

```
SELECT MAX(num) FROM numbers ;
```

Output:

	(No column name)
1	200

Fungsi MAX()

Contoh 2 - Menggunakan fungsi distinct

Tabel numbers

Code:

```
INSERT INTO numbers(num) VALUES (350), (800), (150), (300),(450), (100), (250);  
select * from numbers;
```

```
SELECT MAX(DISTINCT(num)) FROM numbers ;
```

Output:

	(No column name)
1	800

	num
1	50
2	100
3	150
4	200
5	350
6	800
7	150
8	300
9	450
10	100
11	250

Fungsi MAX()

Contoh 3 - Menggunakan formula

Tabel numbers

Code:

	num
1	50
2	100
3	150
4	200
5	350
6	800
7	150
8	300
9	450
10	100
11	250

```
SELECT MAX((num * 10) + 1) FROM numbers ;
```

Output:

	(No column name)
1	8001

Fungsi MAX()

Code:

Contoh 4 - Menggunakan GROUP BY

Tabel workers

	developer_id	team_id	name	position	technology	salary
1	1	1	Payal	Junior Developer	Angular	30000
2	2	1	Heena	Junior Developer	Angular	10000
3	3	3	Vishnu	Manager	Maven	25000
4	4	3	Rahul	Support	Marketing	20000
5	5	3	Siddhesh	Tester	Maven	20000
6	6	7	Siddharth	Manager	Java	25000
7	7	4	Brahma	Developer	Marketing	20000
8	8	1	Arjun	Tester	Angular	19000
9	9	2	Nitin	Developer	MySQL	20000
10	10	2	Ramesh	Administrator	MySQL	30000
11	11	2	Rohan	Admin	SQL	20000
12	12	2	Raj	Designer	SQL	30000

```
SELECT
team_id,
MAX(salary)
FROM
workers
GROUP BY team_id ;
```

Output:

	team_id	(No column name)
1	1	30000
2	2	30000
3	3	25000
4	4	20000
5	7	25000

referensi

- Dwi Puspitasari, S.Kom, “**Buku Ajar Dasar Basis Data**”, *Program Studi Manajemen Informatika Politeknik Negeri Malang*, 2012.
- Fathansyah, “**Basisdata Revisi Kedua**”, Bandung: Informatika, 2015.
- <https://www.w3schools.com>



SQL: View + Stored Procedure

Web Developer

Prodi Teknologi Informasi
Fakultas Teknik
Universitas Tidar Magelang

Learning Outcome

Setelah menyelesaikan sesi pelatihan ini, mahasiswa:

- Memahami konsep View di MySQL
- Memahami konsep Stored Procedure di MySQL
- Mengetahui cara membuat, mengubah, dan menghapus View
- Mengetahui cara membuat, menjalankan, dan menghapus Stored Procedure
- Menggunakan parameter IN, OUT, dan INOUT dalam Stored Procedure

Topik

- Konsep View
- Konsep Stored Procedure
- Penerapan View
- Penerapan Stored Procedure
- Penerapan Parameter In, OUT di Stored Procedure

Pengenalan **View**

- View adalah tabel virtual yang berisi hasil query SQL.
- View menyederhanakan query kompleks.
- View tidak menyimpan data fisik, hanya menyimpan definisi query.

Manfaat View

- Meningkatkan keamanan data dengan memberikan akses terbatas
- Menyederhanakan query kompleks
- Meningkatkan keterbacaan dan pemeliharaan kode
- Mengabstraksi logika bisnis dari tabel fisik

Mengisolasi perubahan dan memudahkan perubahan, missal ada perubahan pada model penampilan data anggota yang meminjam buku, cukup diubah pada definisi view ini

```
SELECT a.nama, b.judul, pa.tanggal_pinjam FROM anggota a  
JOIN peminjaman_anggota pa ON a.id_anggota = pa.id_anggota  
JOIN buku b ON pa.id_buku = b.id_buku  
WHERE pa.tanggal_pinjam >= '2024-01-01';
```

CREATE VIEW **anggota_peminjaman_buku** AS

```
SELECT a.nama, b.judul, pa.tanggal_pinjam FROM anggota a  
JOIN peminjaman_anggota pa ON a.id_anggota = pa.id_anggota  
JOIN buku b ON pa.id_buku = b.id_buku  
WHERE pa.tanggal_pinjam >= '2024-01-01';
```

Menyederhanakan query

```
SELECT * FROM anggota_peminjaman_buku
```

Dengan membaca nama view ini, lebih mudah mudah terbaca

Deklarasi View

Menggunakan
perintah

CREATE VIEW

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Contoh:

```
CREATE VIEW view_buku AS  
SELECT judul, pengarang, tahun_terbit  
FROM  
buku;
```

Deklarasi View

- Contoh:

```
CREATE VIEW view_high_salaries AS  
SELECT name, salary  
FROM employees  
WHERE salary > 5000;
```

```
CREATE VIEW view_buku_tertebal AS  
SELECT judul, pengarang, jumlah_halaman  
FROM buku  
WHERE jumlah_halaman = (SELECT MAX(jumlah_halaman) FROM  
buku)  
;
```

Query View

- Setelah view dibuat, selanjutnya bisa dilakukan query terhadapnya
- Format query sama dengan query ada umumnya
- Contoh:

SELECT	*	FROM	`view_buku_tertebal`
SELECT	*	FROM	`view_high_salaries`

Mengubah View

Menggunakan

CREATE OR REPLACE VIEW

```
CREATE OR REPLACE VIEW view_high_salaries AS  
SELECT name, salary, department  
FROM employees  
WHERE salary > 5000;
```

```
CREATE OR REPLACE VIEW view_buku AS  
SELECT judul, pengarang  
FROM buku  
WHERE tahun_terbit >= 2020;
```


Menghapus View

Menggunakan
perintah

DROP VIEW

```
DROP VIEW view_buku_tertebal;
```

```
DROP VIEW view_buku;
```

```
DROP VIEW view_high_salaries;
```

Deklarasi View dengan JOIN

View juga bisa dibuat dari query yang melibatkan klausa JOIN dari beberapa tabel

```
CREATE VIEW view_peminjaman_buku AS SELECT p.peminjam,  
p.tanggal_pinjam, p.tanggal_kembali, bu.judul, bu.pengarang  
FROM peminjaman p INNER JOIN buku bu ON p.id_buku=bu.id_buku  
ORDER BY p.tanggal_pinjam ASC;
```

Deklarasi View dengan Fungsi AGREGAT

View juga bisa dibuat dari query yang melibatkan klausa fungsi Agregat

```
CREATE VIEW view_halaman_terbanyak_per_tahunterbit AS
```

SELECT tahun_terbit,	MAX(jumlah_halaman)	FROM	buku	GROUP	BY
tahun_terbit ORDER BY	tahun_terbit ASC;				

Pengertian **Stored Procedure**

- Stored Procedure adalah kumpulan perintah SQL yang disimpan dan dapat dijalankan di server database.
- Digunakan untuk menjalankan operasi yang berulang atau kompleks.

Manfaat Stored Procedure

- Meningkatkan kinerja dengan mengurangi lalu lintas jaringan
 - Stored procedure menggabungkan perintah-perintah SQL dalam satu blok di server dan dieksekusi di server, klien cukup memanggil nama prosedurnya daripada mengirimkan perintah SQL yang Panjang, dan tidak perlu bolak-balik kirim perintah ke server
- Meningkatkan keamanan data dengan mengabstraksi logika bisnis
 - Logika bisnis yang kompleks terenkapsulasi dalam sebuah nama prosedur, dan melalui nama tersebut yang akan berinteraksi dengan klien
- Mengurangi duplikasi kode dan memudahkan pemeliharaan
 - Jika menginginkan menjalankan proses, cukup memanggil prosedur, tanpa harus menulis ulang perintah SQL
- Memungkinkan penggunaan parameter untuk fleksibilitas

Deklarasi Stored Procedure

```
DELIMITER //
```

Secara default, delimiter adalah ; tetapi delimiter ; digunakan sebagai akhir perintah SQL di tubuh prosedur. Sehingga perlu didefinisikan delimiter baru sebagai penanda berakhirnya blok prosedur

```
CREATE PROCEDURE procedure_name (parameters)
```

```
BEGIN
```

```
-- Perintah SQL di sini
```

```
END //
```

Delimiter digunakan disini

```
DELIMITER ;
```

Mengembalikan definisi delimiter ke karakter awalnya yaitu ;

Deklarasi Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE getEmployee(IN emp_id INT)
```

```
BEGIN
```

```
    SELECT * FROM employees WHERE id = emp_id;
```

```
END //
```

```
DELIMITER ;
```

Parameter



Deklarasi Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE
```

```
BEGIN t_all_buku()
```

```
    SELECT * FROM buku;
```

```
END //
```

```
DELIMITER ;
```


Pemanggilan Stored Procedure

- Pemanggilan stored procedure tanpa parameter `CALL select_all_buku();`
- Pemanggilan stored procedure dengan parameter `CALL getEmployee(1);`

Mengubah Stored Procedure

- Format

```
DROP PROCEDURE IF EXISTS procedure_name;  
CREATE PROCEDURE procedure_name (parameters)  
BEGIN  
    -- Perintah SQL baru di sini  
END;
```

Menghapus Stored Procedure

- Format

```
DROP PROCEDURE procedure_name;
```

Parameter dalam Stored Procedure

- IN Parameter: Input parameter yang nilainya dikirimkan ke stored procedure.
- OUT Parameter: Output parameter yang nilainya dikembalikan dari stored procedure.
- INOUT Parameter: Parameter yang bisa berfungsi sebagai input dan output.

Parameter IN

```
DELIMITER //
```

```
CREATE PROCEDURE getEmployeeById(IN emp_id INT)
```

```
BEGIN
```

```
    SELECT * FROM employees WHERE id = emp_id;
```

```
END
```

```
//
```

```
DELIMITER ;
```

```
CALL getEmployeeById(1);
```

Parameter OUT

```
DELIMITER //
```

```
CREATE PROCEDURE getEmployeeNameById(IN emp_id INT, OUT emp_name  
VARCHAR(100))
```

```
BEGIN
```

```
    SELECT name INTO emp_name FROM employees WHERE id = emp_id;
```

```
END //
```

```
DELIMITER ;
```

```
CALL getEmployeeNameById(1, @name);
```

```
SELECT @name;
```

Parameter INOUT

```
DELIMITER //
```

```
CREATE PROCEDURE incrementSalary(INOUT emp_salary DECIMAL(10,2))  
BEGIN  
    SET emp_salary = emp_salary + 1000;  
END //
```

```
DELIMITER ;
```

```
SET @salary = 5000;  
CALL incrementSalary(@salary);  
SELECT @salary;
```

Stored Procedure dengan Fungsi Agregat

```
DELIMITER //
```

```
CREATE PROCEDURE employeeStatistics(OUT total_employees INT, OUT avg_salary  
DECIMAL(10,2)  
,  
BEGIN
```

```
    SELECT COUNT(*) INTO total_employees FROM
```

```
    SELECT AVG(salary) INTO avg_salary FROM
```

```
END //employees;
```

```
DELIMITER
```

```
;
```

```
CALL employeeStatistics(@total,
```

```
SELECT @total,
```

```
@avg;
```


Stored Procedure dengan Seleksi Kondisi

```
DELIMITER //
```

```
CREATE PROCEDURE manageBooks(IN action CHAR(6), IN book_id INT, IN book_title VARCHAR(255),  
IN book_price DECIMAL(10,2))
```

```
BEGIN
```

```
    IF action = 'INSERT' THEN
```

```
        INSERT INTO buku (id_buku, judul, harga) VALUES (book_id, book_title, book_price);
```

```
    ELSEIF action = 'UPDATE' THEN
```

```
        UPDATE buku SET judul = book_title, harga = book_price WHERE id_buku = book_id;
```

```
    ELSEIF action = 'DELETE' THEN
```

```
        DELETE FROM buku WHERE id_buku = book_id;
```

```
    END IF;
```

```
END //
```

```
DELIMITER ;
```

```
CALL manageBooks('INSERT', 1, 'Buku A', 50000);
```

```
CALL manageBooks('UPDATE', 1, 'Buku B', 75000);
```

```
CALL manageBooks('DELETE', 1, NULL, NULL)
```

Stored Procedure dengan Seleksi Kondisi

CALL	manageBooks('INSERT' ,	1,	'Buku A' ,	50000);
CALL	manageBooks('UPDATE' ,	1,	'Buku B' ,	75000);
CALL	manageBooks('DELETE' ,	1,	NULL,	NULL)

Latihan - Deklarasi View

```
CREATE VIEW view_buku_termahal AS  
SELECT judul, harga  
FROM buku  
WHERE harga = (SELECT MAX(harga) FROM buku);
```

Penggunaan View

```
SELECT * FROM view_buku_termahal;
```

Mengubah View

```
CREATE OR REPLACE VIEW view_buku_termahal AS  
SELECT judul, harga, penerbit  
FROM buku  
WHERE harga = (SELECT MAX(harga) FROM buku);
```

Menghapus View

```
DROP VIEW view_buku_termahal;
```

Deklarasi Stored Procedure (1)

```
DELIMITER //
```

```
CREATE PROCEDURE getBookById(IN book_id INT)
```

```
BEGIN
```

```
    SELECT * FROM buku WHERE id_buku = book_id;
```

```
END //
```

```
DELIMITER ;
```

Deklarasi Stored Procedure (2)

```
DELIMITER //
```

```
CREATE PROCEDURE getBookTitleById(IN book_id INT,  
OUT book_title VARCHAR(255))  
BEGIN
```

```
    SELECT judul INTO book_title FROM buku WHERE  
id buku = book_id;  
END //
```

```
DELIMITER ;
```


Deklarasi Stored Procedure (3)

```
DELIMITER //
```

```
CREATE PROCEDURE updateBookPrice(INOUT book_price  
DECIMAL(10,2))
```

```
BEGIN
```

```
    SET book_price = book_price * 1.1;
```

```
END //
```

```
DELIMITER ;
```

Pemanggilan Stored Procedure

```
CALL getBookById(1);
```

```
CALL getBookTitleById(1, @judul);
```

```
SELECT @judul;
```

```
SET @harga = 1000;
```

```
CALL updateBookPrice(@harga);
```

```
SELECT @harga;
```

Perubahan Stoed Procedure

```
DROP PROCEDURE IF EXISTS getJumlahBukuTahun;
```

```
DELIMITER //
```

```
CREATE PROCEDURE getJumlahBukuTahun(IN tahun INT)
```

```
BEGIN
```

```
    SELECT COUNT(*) AS jumlah_buku, SUM(harga) AS total_harga
```

```
    FROM buku
```

```
    WHERE tahun_terbit = tahun;
```

```
END
```

```
//  
DELIMITER ;
```

Menghapus Stored Procedure

```
DROP PROCEDURE getJumlahBukuTahun;
```

referensi

- Dwi Puspitasari, S.Kom, “**Buku Ajar Dasar Basis Data**”, *Program Studi Manajemen Informatika Politeknik Negeri Malang*, 2012.
- Fathansyah, “**Basisdata Revisi Kedua**”, Bandung: Informatika,

2015.
- <https://www.w3schools.com>



Terima kasih

Universitas Tidar