

SISTEM MANAJEMEN BASIS DATA

- 02. TIPE & MODEL DATA
- 03. REVIEW DDL
- 04. Review DML
- 05. Function

Doni Abdul Fatah
github.com/donifaft
Universitas Trunojoyo Madura

Pokok Bahasan

01. Overview SMBD- Entity Diagram

02. Tipe & Model Data

03. Review DDL

04. Review DML

05. Function

06. Transactional SQL

07. View dan User Authorisation

08. Perulangan dan Keputusan

09. Trigger

10. Stored Procedure

11. System Catalog

12. Embedded SQL

13. Basis Data NoSQL

14. UAS

01. SMBD

- 1) Tipe & Model Data
- 2) Review DDL
- 3) Review DML
- 4) Function
- 5) Kontrak Perkuliahan
- 6) Kebutuhan Software
- 7) Contact
- 8) Referensi

3) Tipe & Model Data

1. Atribut Tipe Data
2. Tipe DataNUMERIK
3. Tipe DataSTRING
4. Tipe DataDATE&TIME

Atribut Tipe Data

- ❑ ***Atribut tipe data*** adalah aturan yang kita terapkan untuk sebuah kolom.
- ❑ Atribut tipe data yang paling umum digunakan, yakni: **AUTO_INCREMENT, BINARY, DEFAULT, NOT NULL, NULL, SIGNED, UNSIGNED,** dan **ZEROFILL**.

Atribut Tipe Data (contd-2)

Atribut **AUTO_INCREMENT**

- Untuk tipe data numerik (biasanya tipe data **INT**),
- Sebuah kolom **AUTO_INCREMENT**, maka setiap kali kita menginputkan data, nilai pada kolom ini akan bertambah **1**.
- Nilai pada kolom tersebut juga akan bertambah jika kita input dengan **NULL** atau nilai **0**.
- Pada sebuah tabel, hanya 1 kolom yang dapat dikenai atribut **AUTO_INCREMENT**.
- Setiap kolom **AUTO_INCREMENT** juga akan dikenakan atribut **NOT NULL** secara otomatis.
- Kolom **AUTO_INCREMENT** juga harus digunakan sebagai **KEY** (biasanya **PRIMARY KEY**)

Atribut Tipe Data (contd-2)

Atribut **BINARY**

- Untuk tipe data huruf, seperti **CHAR** dan **VARCHAR**. Tipe data **CHAR**, **VARCHAR** dan **TEXT** tidak membedakan antara huruf besar dan kecil (*case-insensitive*), namun jika diberikan atribut **BINARY**, maka kolom tersebut akan membedakan antara huruf besar dan kecil (*case-sensitive*)

Atribut Tipe Data (contd-3)

Atribut **BINARY** - **DEFAULT**

- Atribut **BINARY** Untuk tipe data huruf, seperti **CHAR** dan **VARCHAR**. Tipe data **CHAR**, **VARCHAR** dan **TEXT** tidak membedakan antara huruf besar dan kecil (*case-insensitive*), namun jika diberikan atribut **BINARY**, maka kolom tersebut akan membedakan antara huruf besar dan kecil (*case-sensitive*)
- Atribut **DEFAULT** dapat digunakan pada hampir semua tipe data.
- Fungsinya untuk *menyediakan nilai bawaan* untuk kolom seandainya tidak ada data yang diinput kepada kolom tersebut.

Atribut Tipe Data (contd-4)

Atribut NOT NULL - NULL

- Atribut **NOT NULL** dapat digunakan pada hampir semua tipe data, Fungsinya untuk memastikan bahwa nilai pada kolom tersebut tidak boleh kosong.
- Jika kita menginput data, namun tidak memberikan nilai untuk kolom tersebut, akan menghasilkan *error* pada MySQL.
- Atribut **NULL** berkebalikan dengan **NOT NULL**, dimana jika sebuah kolom didefinisikan dengan **NULL**, maka kolom tersebut *tidak harus* berisi nilai.

Atribut Tipe Data (contd-5)

Atribut SIGNED, UNSIGNED, ZEROFILL

- Atribut **SIGNED** digunakan untuk tipe data numerik.
 - **UNSIGNED** berfungsi agar kolom dapat menampung nilai negatif.
 - Atribut **SIGNED** biasanya dicantumkan hanya untuk menegaskan bahwa kolom tersebut mendukung nilai negatif, karena MySQL sendiri telah menyediakan nilai negatif secara *default* untuk seluruh tipe numerik.
- Atribut **UNSIGNED** digunakan untuk tipe data numerik, namun berbeda sifatnya untuk tipe data **INT,DECIMAL** dan **FLOAT**.
 - Untuk tipe data **INT**, atribut **UNSIGNED** berfungsi *mengorbankan* nilai negatif, untuk mendapatkan jangkauan nilai positif yang lebih tinggi.
 - Namun untuk tipe data **DECIMAL** dan **FLOAT**, atribut **UNSIGNED** hanya akan menghilangkan nilai negatif, tanpa menambah jangkauan data.
- Atribut **ZEROFILL** digunakan untuk tipe data numerik, dimana berfungsi untuk tampilan format data yang akan mengisi nilai 0 di sebelah kanan dari data.
 - Jika kita menggunakan atribut**ZEROFILL** untuk suatu kolom, secara otomatis kolom tersebut juga dikenakan atribut **UNSIGNED**.

Contoh Atribut Tipe Data (contd-6)

Contoh query untuk penggunaan attribut :

```
CREATE TABLE contoh_atribut (no int AUTO_INCREMENT, nama  
VARCHAR(30) NOT NULL, umur TINYINT UNSIGNED DEFAULT  
'10', kodepos CHAR(5) NULL, PRIMARY KEY (no));
```

1 DESCRIBE contoh_atribut

COLUMNS (6x4)					
Field	Type	Null	Key	Default	Extra
no	int(11)	NO	PRI	(NULL)	auto_increment
nama	varchar(30)	NO		(NULL)	
umur	tinyint(3) unsigned	YES		10	
kodepos	char(5)	YES		(NULL)	

```
INSERT INTO contoh_atribut VALUES  
(NULL, 'Aku', NULL, 20155),  
(0, 'Dia', 23, 62115), →  
(1, 'Kita', 19, 62806),  
(2, 'Mereka', 20, 67002),  
(3, 'Kamu', 25, 69028);
```

Duplicat
Primary
key

```
INSERT INTO contoh_at VALUES  
(NULL, 'Aku', NULL, 20155),  
(0, 'Dia', 23, 62115),  
(3, 'Kita', 19, 62806),  
(4, 'Mereka', 20, 67002),  
(5, 'Kamu', 25, 69028);
```

Contoh Atribut Tipe Data (contd-7)

```
1 SELECT * from contoh_at
2
```

contoh_at (4x5)

no	nama	umur	kodepos
1	Aku	(NULL)	20155
2	Dia	23	62115
3	Kita	19	62806
4	Mereka	20	67002
5	Kamu	25	69028

Coba Inputkan data berikut :

```
INSERT INTO contoh_at VALUES (9,NULL,32,10099);
```

Bagaimana Hasilnya?

Pembagian Type Data Secara umum

1. **Numeric Values** yaitu angka atau bilangan seperti 10; 123; 100.50; -10; 1.2E+17; 2.7e-11; dan sebagainya.
 - **Bilangan Bulat (Integer)** dan **Bilangan Pecahan (Floating-point)**.
 - **Bilangan bulat** : bilangan **tanpa tanda desimal** sedangkan **bilangan pecahan** adalah bilangan dengan **tanda desimal**.
 - Kedua jenis bilangan dapat bernilai **positif (+)** dan **negatif (-)**.
 - Jika bilangan menggunakan **tanda positif (+)** atau **(-)**, disebut **SIGNED**.
 - Bila **tanpa tanda** apapun disebut **UNSIGNED**. Karena tanda positif (+) dapat diabaikan penulisannya maka pada bilangan yang bernilai positif disebut **UNSIGNED**.

Pembagian Type Data Secara umum (Contd-2)

2. **String/Character Values** adalah semua karakter (atau teks) yang penulisannya selalu diapit tanda kutip baik kutip tunggal ('') atau kutip ganda ("").
 - Berlaku juga pada angka jika ditulis menggunakan tanda kutip akan menjadi karakter atau string.
3. **Date and Time Values** yaitu tanggal dan waktu.
 - Format standar (default) penulisan
 - Tanggal : “**tahun-bulan-tanggal**”, Misalnya **04 Maret 2019** dituliskan “**2019-03-04**”.
 - Waktu : “**jam-menit-detik**”. Contoh, “**19:31:07**”.
 - Data tanggal dan waktu digabung menjadi “**2019-03-04 19:31:07**”.
3. **NULL**.
 - NULL** bukan data, tapi mewakili sesuatu yang “tidak pasti”, “tidak diketahui” atau “belum ada nilainya”.
 - Contoh** Kita melakukan survei Pemilihan presiden **2019** antara paslon **01** dan **02**, tentang jumlah pemilih milenia yang memilih antara paslon **01** dan **02**. Selama survei belum selesai maka data pastinya belum dapat diketahui. Oleh sebab itu, data tersebut dapat diwakili dengan **NULL**, alias **belum diketahui**.

Tipe data NUMERIK

- **Integer** adalah tipe data untuk angka bulat (misalnya: 1, 6, 59, -533, 1449). MySQL menyediakan beberapa tipe data untuk integer, perbedaannya lebih kepada jangkauan yang juga berpengaruh terhadap ukuran tipe data tersebut.
- **BIT**
A bit-field, from 1 to 64 bits wide. (Prior to MySQL 5 BIT was functionally equivalent to TINYINT)
- **BIGINT**
Integer value, supports numbers from -9223372036854775808 to 9223372036854775807 (or 0 to 18446744073709551615 if UNSIGNED)
- **BOOLEAN (or BOOL)**
Boolean flag, either 0 or 1, used primarily for on/off flags
- **DECIMAL (or DEC)**
Floating point values with varying levels of precision
- **DOUBLE**
Double-precision floating point values
- **FLOAT**
Single-precision floating point values

Tipe data NUMERIK (Contd-2)

- INT (or INTEGER)**

Integer value, supports numbers from -2147483648 to 2147483647 (or 0 to 4294967295 if UNSIGNED) 4 byte (32 bit)

- MEDIUMINT**

Integer value, supports numbers from -8388608 to 8388607 (or 0 to 16777215 if UNSIGNED) 3 byte (24 bit)

- REAL/ DOUBLE**

4-byte floating point values (-1.79...E+308 s/d -2.22...E-308, 0, dan 2.22...E-308 s/d 1.79...E+308.) 8 byte (64 bit)

- SMALLINT**

Integer value, supports numbers from -32768 to 32767 (or 0 to 65535 if UNSIGNED) Ukuran : 2 byte (16 bit).

- TINYINT**

Integer value, supports numbers from -128 to 127 (or 0 to 255 if UNSIGNED) Ukuran : 1 byte (8 bit)

Tipe data NUMERIK - Integer (Contd-3)

Tipe Data	Jangkauan SIGNED	Jangkauan UNSIGNED	Ukuran
TINYINT	-128 to 127	0 to 255	1 byte
SMALLINT	-32,768 to 32,767	0 to 65,535	2 bytes
MEDIUMINT	-8,388,608 to 8,388,607	0 to 16,777,215	3 bytes
INT	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295	4 bytes
BIGINT	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615	8 bytes

Tipe data NUMERIK - Integer (Contd-3)

- **Format query** untuk tipe data **integer** adalah:
- INT[(M)] [UNSIGNED] [ZEROFILL]
- M : menunjukkan lebar karakter maksimum. Nilai M maksimum adalah 255
- tanda [dan] berarti pemakaianya adalah optional
- Contoh query :

```
CREATE TABLE contoh_int (mini TINYINT, kecil SMALLINT  
UNSIGNED, sedang MEDIUMINT(4) ZEROFILL, biasa INT(4) UNSIGNED,  
besar BIGINT(6) UNSIGNED ZEROFILL);
```

```
mysql> DESC contoh_int;
```

Field	Type	Null	Key	Default	Extra
mini	tinyint(4)	YES		NULL	
kecil	smallint(5) unsigned	YES		NULL	
sedang	mediumint(4) unsigned zerofill	YES		NULL	
biasa	int(4) unsigned	YES		NULL	
besar	bigint(6) unsigned zerofill	YES		NULL	

```
INSERT INTO contoh_int VALUES  
((100), (100), (100), (100), (100));
```

```
mysql> INSERT INTO contoh_int values ((122), (122), (122),  
(122), (122));
```

```
Query OK, 1 row affected (0.07 sec)
```

```
mysql> SELECT * FROM contoh_int;
```

mini	kecil	sedang	biasa	besar
122	122	0122	122	000122

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

Tipe data NUMERIK - Integer (Contd-3)

```
INSERT INTO contoh_int VALUES ((100), (100), (100), (100), (100));
```

```
INSERT INTO contoh_int VALUES  
((1000), (1000), (1000), (1000), (1000));
```

```
INSERT INTO contoh_int VALUES  
((10000), (10000), (10000), (10000), (10000));
```

1 SELECT * from contoh_int				
contoh_int (5x4)				
mini	kecil	sedang	biasa	besar
10	10	0010	10	000010
100	100	0100	100	000100
127	1.000	1.000	1.000	001000
127	10.000	10.000	10.000	010000

→ Kenapa???

Tipe data NUMERIK - DECIMAL (Contd-4)

- Tipe data fixed point adalah tipe data angka pecahan (desimal), dimana jumlah angka pecahan (angka di belakang koma) sudah di tentukan dari awal.
- Query fixed point adalah:
- **DECIMAL [(M[,D])] [UNSIGNED] [ZEROFILL]**
- DECIMAL merupakan kata kunci untuk mendefiniskan suatu kolom sebagai fixed point.
- *Opsional query [M,D]* dimana **M =65** adalah total jumlah digit keseluruhan, dan **D=30** adalah jumlah digit dibekang koma (pecahan).
- Dengan syarat, nilai D tidak boleh lebih besar dari nilai M. Jika tidak menyertakan M dan D default **DECIMAL**, M set 10. Dan D default 0.

Deklarasi	Jangkauan
DECIMAL (4,1)	-999,9 to 999,9
DECIMAL (6,2)	-9999,99 to 9999,99
DECIMAL (3,2)	-9,99 to 9,99
DECIMAL (8,2)	-999999,99 to 999999,99

Tipe data NUMERIK - DECIMAL (Contd-5)

```
mysql> CREATE TABLE contoh_dec (satuan DECIMAL(3,2), puluhan DECIMAL(4,2),  
ribuan DECIMAL(5,2), normal DECIMAL, cantik DECIMAL(8,2) ZEROFILL);
```

Query OK, 0 rows affected (0.13 sec)

```
mysql> DESCRIBE contoh_dec;
```

Field	Type	Null	Key	Default	Extra
satuan	decimal(3,2)	YES		NULL	
puluhan	decimal(4,2)	YES		NULL	
ribuan	decimal(5,2)	YES		NULL	
normal	decimal(10,0)	YES		NULL	
cantik	decimal(8,2) unsigned zerofill	YES		NULL	

5 rows in set (0.08 sec)

```
INSERT INTO contoh_dec  
values ((1.3), (45.32),  
(455.77), (4552.3),  
(45000.45));
```

1 SELECT * from contoh_dec					
contoh_dec (5x1)					
satuan	puluhan	ribuan	normal	tambah	
1,30	45,32	455,77	4.552	045000,45	

```
CREATE TABLE contoh_dec  
(satuan DECIMAL(3,2),  
puluhan DECIMAL(4,2),  
ribuan DECIMAL(5,2),  
normal DECIMAL, tambah  
DECIMAL(8,2) ZEROFILL);
```

```
mysql> INSERT INTO contoh_dec values ((1.3), (55.32),  
(523.77), (7832.3), (150000.45));
```

Query OK, 1 row affected, 1 warning (0.07 sec)

```
mysql> SELECT * FROM contoh_dec;
```

satuan	puluhan	ribuan	normal	cantik
1.30	55.32	523.77	7832	150000.45

1 row in set (0.00 sec)

Kenapa hasilnya???

Tipe data NUMERIK - FLOAT dan DOUBLE (Contd-6)

- Pada tipe **fixed point** kita mendefiniskan suatu kolom dengan nilai pecahan yang tetap, untuk tipe **floating point**, nilai pecahan yang dapat diinput bisa berbeda-beda.
- Untuk tipe floating point, MySQL menyediakan 2 jenis tipe data, yaitu **FLOAT** (4 byte (32 bit)) dan **DOUBLE** (8 byte (64 bit)).
- Perbedaan keduanya terletak pada **presisi** (ketelitian) pembulatan. **FLOAT** menggunakan **single-precision**, sedangkan **DOUBLE** menggunakan **double-precision**.

Tipe Data	Jangkauan	Ukuran
FLOAT	-3.402823466E+38 to 3.402823466E+38	4 bytes
DOUBLE	-1.7976931348623157E+308 to 1.7976931348623157E+308	8 bytes

Tipe data NUMERIK - FLOAT dan DOUBLE (Contd-7)

- Untuk tipe data **FLOAT** dan **DOUBLE**, format querynya adalah:
- **FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]**
- **DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]**
- Opsional query **[M, D]** dimana **M** adalah total jumlah digit keseluruhan, dan **D** adalah jumlah digit dibekang koma (pecahan).

```
mysql> CREATE TABLE contoh_float (satuan FLOAT(3,2), puluhan FLOAT(4,2),
ribuan FLOAT(5,2), positif DOUBLE ZEROFILL UNSIGNED,
cantik DOUBLE(8,2) ZEROFILL);
```

Query OK, 0 rows affected (1.21 sec)

```
mysql> DESCRIBE contoh_float;
```

Field	Type	Null	Key	Default	Extra
satuan	float(3,2)	YES		NULL	
puluhan	float(4,2)	YES		NULL	
ribuan	float(5,2)	YES		NULL	
positif	double unsigned zerofill	YES		NULL	
cantik	double(8,2) unsigned zerofill	YES		NULL	

5 rows in set (0.04 sec)

```
CREATE TABLE contoh_float (satuan
FLOAT(3,2), puluhan FLOAT(4,2),ribuan
FLOAT(5,2), positif DOUBLE ZEROFILL
UNSIGNED,tambah DOUBLE(8,2) ZEROFILL);
```

```
mysql> INSERT INTO contoh_float values ((1.3), (55.32), (523.77),
(7832.3), (150000.45));
```

Query OK, 1 row affected (0.15 sec)

```
mysql> SELECT * FROM contoh_float;
```

satuan	puluhan	ribuan	positif	cantik
1.30	55.32	523.77	00000000000000007832.3	150000.45

1 row in set (0.00 sec)

```
INSERT INTO contoh_float values
((1.3), (45.32), (455.77),
(4552.3), (45000.45));
```

```
1 SELECT * from contoh_float
```

contoh_float (5x2)

satuan	puluhan	ribuan	positif	tambah
1,30	45,32	455,77	00000000000000004552,3	45.000,45
1,30	55,32	523,77	00000000000000007832,3	01500,45

Tipe data STRING (Text)

- Beberapa tipe data string: **CHAR**, **VARCHAR**, **BINARY**, **VARBINARY**, **TEXT** dan **BLOB**.
- Perbedaan dari tipe data ini adalah dari *ukuran*, *cara penyimpanan*, dan dukungan *case-sensitif* (*perbedaan huruf besar dan kecil*).
- CHAR** menyimpan data string ukuran tetap
 - Fixed-length string from 0 to 255 chars long. Its size must be specified at create time, or MySQL assumes CHAR(1)
- ENUM** (kumpulan data)
 - Accepts one of a predefined set of up to 64K strings
- LONGTEXT** menyimpan data text
 - Same as TEXT, but with a maximum size of 4GB, 0 s/d 2³² – 1 karakter
- MEDIUMTEXT** untuk menyimpan data text
 - Same as TEXT, but with a maximum size of 16K, 0 s/d 2²⁴ – 1 karakter
- SET** (himpunan data)
 - Accepts zero or more of a predefined set of up to 64 strings Sampai dengan 255 string anggota

Tipe data STRING (Contd-2)

- TEXT**
Variable-length text with a maximum size of 64K, 0 s/d 65.535 ($2^{16} - 1$) karakter
- TINYTEXT** menyimpan data text.
Same as TEXT, but with a maximum size of 255 bytes, 0 s/d 65.535 (versi 5.0.3)
- VARCHAR** menyimpan data string ukuran dinamis.
Same as CHAR, but stores just the text. The size is a maximum, not a minimum. 0 s/d 65.535 (versi 5.0.3)
- BLOB** menyimpan data biner.
Blob with a maximum length of 64K, $2^{16} - 1$ byte
- MEDIUMBLOB** menyimpan data biner.
Blob with a maximum length of 16MB, $2^{24} - 1$ byte
- LONGBLOB**, menyimpan data biner.
Blob with a maximum length of 4GB, $2^{32} - 1$ byte
- TINYBLOB** menyimpan data biner.
Blob with a maximum length of 255 bytes, 255 byte

Tipe data STRING - CHAR dan VARCHAR (Contd-3)

- Format query tipe data CHAR dan VARCHAR:
- CHAR [(M)]
- VARCHAR [(M)]

Data	CHAR(5)	Ukuran Penyimpanan	VARCHAR(5)	Ukuran Penyimpanan
''	' '	5 byte	''	1 byte
'du'	'du'	5 byte	'du'	3 byte
'dunia'	'dunia'	5 byte	'dunia'	6 byte
'duniailkom'	'dunia'	5 byte	'dunia'	6 byte

Tipe data STRING - CHAR dan VARCHAR (Contd-4)

```
CREATE TABLE contoh_cha (cha CHAR(5), varcha VARCHAR(5));
```

1 desc contoh_cha					
COLUMNS (6x2)					
Field	Type	Null	Key	Default	Extra
cha	char(5)	YES		(NULL)	
varcha	varchar(5)	YES		(NULL)	

```
INSERT INTO contoh_cha VALUES  
( 'a ', 'a ' ),  
( 'dunia', 'dunia' ), ( 'dunia' ,  
  'dunia' );
```

1 SELECT * from contoh_cha	
contoh_cha (2x3)	
cha	varcha
a	a
dunia	dunia
dunia	dunia

```
mysql> SELECT * FROM contoh_cha;  
+-----+-----+  
| cha | varcha |  
+-----+-----+  
| a   | a      |  
| dunia | dunia |  
+-----+-----+
```

Tipe data STRING - BINARY dan VARBINARY (Contd-5)

- dimana tipe data binary akan **disimpan secara biner** (bit per bit), bukan secara karakter seperti CHAR. Sederhananya, hal ini akan berefek pada **casesensitif** data (perbedaan penggunaan huruf besar dan huruf kecil).
- Format query tipe data BINARY dan VARBINARY:
 - BINARY [(M)]
 - VAR BINARY [(M)]

Tipe data STRING - BINARY dan VARBINARY (Contd-6)

```
mysql> CREATE TABLE contoh_bin (bin BINARY(5),
varbin VARBINARY(5));
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO contoh_bin values ('dunia','dunia');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM contoh_bin;
+-----+-----+
| bin  | varbin |
+-----+-----+
| dunia | dunia |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM contoh_bin where bin='dunia';
+-----+-----+
| bin  | varbin |
+-----+-----+
| dunia | dunia |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM contoh_bin where bin='DUNIA';
Empty set (0.00 sec)
```

1 SELECT * FROM contoh_bin;	
contoh_bin (2x1)	
bin	varbin
dunia	dunia

```
CREATE TABLE contoh_bin (bin
BINARY(5),
varbin VARBINARY(5));

INSERT INTO contoh_bin values
('dunia','dunia');
```

1 SELECT * FROM contoh_bin where bin='dunia';	
contoh_bin (2x1)	
bin	varbin
dunia	dunia

```
SELECT * FROM contoh_bin
where bin='DUNIA';
```

Kenapa tidak ada hasilnya?

1 SELECT * FROM contoh_bin where bin='DUNIA';	
contoh_bin (2x0)	
bin	varbin

Tipe data STRING – TEXT (Contd-7)

- Untuk data string yang lebih besar, MySQL menyediakan tipe data **TEXT**, yang terdiri dari : **TINYTEXT**, **TEXT**, **MEDIUMTEXT**, dan **LONGTEXT**.

Tipe Data	Ukuran Maksimum	Jumlah Karakter Maksimum
TINYTEXT	255 byte	255
TEXT	65.535 byte (64 KB)	6.5535
MEDIUMTEXT	16.777.215 byte (16MB)	16.777.215
LONGTEXT	4.294.967.295 (4GB)	4.294.967.295

Tipe data STRING - TEXT (Contd-8)

```
mysql> CREATE TABLE contoh_text (tin TINYTEXT, tex TEXT,  
lon LONGTEXT);  
Query OK, 0 rows affected (0.05 sec)  
  
mysql> DESC contoh_text;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type   | Null | Key  | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| tin   | tinytext | YES  |       | NULL    |       |  
| tex   | text    | YES  |       | NULL    |       |  
| lon   | longtext| YES  |       | NULL    |       |  
+-----+-----+-----+-----+-----+-----+  
3 rows in set (0.09 sec)  
  
mysql> INSERT INTO contoh_text values ('duniailkom',  
'duniailkom','duniailkom.com');  
Query OK, 1 row affected (0.04 sec)  
  
mysql> SELECT * FROM contoh_text;  
+-----+-----+-----+  
| tin   | tex   | lon  |  
+-----+-----+-----+  
| duniailkom | duniailkom | duniailkom.com |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

```
CREATE TABLE contoh_text (tin  
TINYTEXT, tex TEXT,  
lon LONGTEXT);  
  
INSERT INTO contoh_text values  
('duniailkom',  
'duniailkom','duniailkom.com');
```

Tipe data STRING - BLOB (Contd-9)

- tipe data *versi binary* dari **TEXT**, dimana karakter akan disimpan dalam **bit**. Dan untuk karakter huruf, huruf besar dan kecil akan dibedakan ('A' tidak sama dengan 'a'). Sama seperti **TEXT**, **BLOB** juga memiliki beberapa tipe : **TINY BLOB**, **BLOB**, **MEDIUM BLOB**, dan **LONGBLOB**.

Tipe Data	Ukuran Maksimum	Jumlah Karakter Maksimum
TINYBLOB	255 byte	255
BLOB	65.535 byte (64 KB)	65.535
MEDIUMBLOB	16.777.215 byte (16MB)	16.777.215
LONGBLOB	4.294.967.295 (4GB)	4.294.967.295

Tipe data STRING - BLOB (Contd-10)

```
mysql> CREATE TABLE contoh_blob (tin TINYBLOB, blo BLOB,  
lon LONGBLOB);  
Query OK, 0 rows affected (0.14 sec)
```

```
mysql> DESCRIBE contoh_blob;  
+-----+-----+-----+-----+-----+  
| Field | Type   | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+  
| tin   | tinyblob | YES  |     | NULL    |       |  
| blo   | blob    | YES  |     | NULL    |       |  
| lon   | longblob | YES  |     | NULL    |       |  
+-----+-----+-----+-----+-----+  
3 rows in set (0.01 sec)
```

```
mysql> INSERT INTO contoh_blob values ('duniailkom',  
'duniailkom','duniailkom.com');  
Query OK, 1 row affected (0.05 sec)
```

```
mysql> SELECT * FROM contoh_blob;  
+-----+-----+-----+  
| tin   | blo   | lon  |  
+-----+-----+-----+  
| duniailkom | duniailkom | duniailkom.com |  
+-----+-----+-----+  
1 row in set (0.01 sec)
```

```
CREATE TABLE contoh_blob  
(tin TINYBLOB, blo BLOB,  
lon LONGBLOB);
```

```
INSERT INTO contoh_blob  
values ('duniailkom',  
'duniailkom','duniailkom.co  
m');
```

Tipe data DATE & TIME

- tipe data date (tanggal)**, tipe data ini digunakan untuk menyimpan data yang berkaitan dengan tanggal dan waktu. Tipe data **date**, terdiri dari: **DATE**, **TIME**, **DATETIME**, **TIMESTAMP**, dan **YEAR**.
- DATE**
Date from 1000-01-01 to 9999-12-31 in the format YYYY-MM-DD, 3 byte.
- TIME**
Time in the format HH:MM:SS, 3 byte.
- YEAR**
A 2 or 4 digit year, 2 digit years support a range of 70 (1970) to 69 (2069), 4 digit years support a range of 1901 to 2155, 1 byte.
- DATETIME**
A combination of DATE and TIME, 1000-01-01 00:00:00' s/d '9999-12-31 23:59:59, 8 byte.
- TIMESTAMP**
Functionally equivalent to DATETIME (but with a smaller range)

Tipe data DATE & TIME - DATE (Contd-1)

- **DATE, TIME, DATETIME, TIMESTAMP, dan YEAR.**
Perbedaan dari tipe-tipe tersebut terletak pada format penyimpanan data.

Tipe Data	Jangkauan	Ukuran	Zero Value
DATE	□1000-01-01□ to □9999-12-31□	3 byte	□0000-00-00□
DATETIME	□1000-□01-01 00:00:01□ to □9999-12-31 23:59:59□	8 byte	□0000-00-00 00:00:00□
TIMESTAMP	□1970-01-01 00:00:00□ to □2038-01-18 22:14:07□	4 byte	□0000-00-00 00:00:00□
TIME	□□838:59:59□ to □838:59:58□	3 byte	□00:00:00□
YEAR(2)	00 to 99	1 byte	□00□
YEAR(4)	1901 to 2155	1 byte	□0000□

Tipe data DATE & TIME - DATE (Contd-2)

Tipe Data	Format Input
DATETIME	'CCYY-MM-DD hh:mm:ss'
TIMESTAMP	'YY-MM-DD hh:mm:ss'
	'CCYYMMDDhhmmss'
	'YYMMDDhhmmss'
	CCYYMMDDhhmmss
	YYMMDDhhmmss
DATE	'CCYY-MM-DD'
	'YY-MM-DD'
	'CCYYMMDD'
	'YYMMDD'
	CCYYMMDD
	YYMMDD
TIME	'hh:mm:ss'
	'hhmmss'
	hhmmss
YEAR	'CCYY'
	'YY'
	CCYY
	YY

- **CCYY** : input untuk tahun, dimana **YY** berupa tahun **2 digit**, seperti 98, 78, dan 00, sedangkan untuk **CCYY** adalah tahun dengan **4 digit**, seperti 2001, 1987, 2012. Untuk tahun dengan 2 digit, MySQL mengkonversinya dengan aturan 70-99 menjadi 1970-1999 dan 00-69 menjadi 2000-2069.
- **MM**: bulan dalam format dua digit, seperti 05,07,dan 12.
- **DD**: tanggal dalam format dua digit, seperti 14, 06 dan 30.
- **hh**: jam dalam format 2 digit, seperti 06,09, dan 12.
- **mm**: menit, dalam format 2 digit, seperti 15, 45, dan 59.
- **ss**: detik, dalam format 2 digit, seperti 10, 40, dan 57.

Tipe data DATE & TIME - DATE (Contd-3)

```
mysql> CREATE TABLE contoh_date (dat DATE, tim TIME, dattim DATETIME,  
timestamp TIMESTAMP, yea YEAR);  
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> DESCRIBE contoh_date;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| dat | date | YES | NULL | NULL | NULL |  
| tim | time | YES | NULL | NULL | NULL |  
| dattim | datetime | YES | NULL | NULL | NULL |  
| timestamp | timestamp | NO | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |  
| yea | year(4) | YES | NULL | NULL | NULL |  
+-----+-----+-----+-----+-----+-----+
```

```
mysql> INSERT INTO contoh_date values (NOW(),NOW(),NOW(),NOW(),'2012');  
Query OK, 1 row affected, 1 warning (0.05 sec)
```

```
mysql> SELECT * FROM contoh_date;  
+-----+-----+-----+-----+-----+  
| dat | tim | dattim | timestamp | yea |  
+-----+-----+-----+-----+-----+  
| 2012-10-20 | 19:40:45 | 2012-10-20 19:40:45 | 2012-10-20 19:40:45 | 2012 |  
+-----+-----+-----+-----+-----+
```

Tipe data DATE & TIME - DATE (Contd-4)

```
CREATE TABLE contoh_date (dat DATE, tim TIME, dattim DATETIME,  
timestam TIMESTAMP, yea YEAR);
```

```
INSERT INTO contoh_date values (NOW(),NOW(),NOW(),NOW(),'2012');
```

```
1 SELECT * FROM contoh_date;
```

contoh_date (5x1)

dat	tim	dattim	timestam	yea
2019-03-04	22:12:10	2019-03-04 22:12:10	2019-03-04 22:12:10	2012

```
SELECT DATE_FORMAT(dat, '%d/%m/%Y') FROM contoh_date;
```

```
1 SELECT DATE_FORMAT(dat, '%d/%m/%Y') FROM contoh_date;
```

Hasil #1 (1x1)

```
DATE_FORMAT(dat, '%d/%m/%Y')  
04/03/2019
```

Format DATE_FORMAT (Contd-5)

```
SELECT DATE_FORMAT(dat, '%d - %m - %Y') FROM contoh_date;
```

```
SELECT DATE_FORMAT(dat, '%d %M %Y') FROM contoh_date;
```

```
1 SELECT DATE_FORMAT(dat, '%d %M %Y') FROM contoh_date;
```

```
/ contoh_date (5x1) \ Hasil #2 (1x1) \ Hasil #3 (1x1) \
DATE_FORMAT(dat, '%d %M %Y')
04 March 2019
```

```
1 SELECT DATE_FORMAT(dat, '%d - %m - %Y') FROM contoh_date;
```

```
/ Hasil #1 (1x1) \
DATE_FORMAT(dat, '%d - %m - %Y')
04 - 03 - 2019
```

```
SELECT DATE_FORMAT(dattim, '%d - %m - %Y,%k:%i:%s'),DATE_FORMAT
(dat, '%d - %m - %Y' ) FROM contoh_date;
```

```
1 SELECT DATE_FORMAT(dattim, '%d - %m - %Y,%k:%i:%s'),DATE_FORMAT (dat,'%d - %m - %Y' ) FROM contoh_date;
```

```
/ contoh_date (5x1) \ Hasil #2 (2x1) \
DATE_FORMAT(dattim, '%d - %m - %Y,%k:%i:%s')
04 - 03 - 2019,22:12:10  DATE_FORMAT (dat,'%d - %m - %Y')
04 - 03 - 2019
```

Tipe data ENUM (Contd-1)

- Tipe data **ENUM** tipe data khusus untuk **kolom** yang **nilai datanya** sudah kita **tentukan sebelumnya**.
- Pilihan ini dapat berisi **1** sampai dengan **65,535** pilihan **string**.
- Dimana kolom yang didefinisikan sebagai **ENUM** hanya dapat memilih **satu** diantara **pilihan string** yang **tersedia**.
- Contoh : **Kolom** daftar Jurusan **harus dipilih mahasiswa**. **Pilihan** ini harus **sudah tersedia** sebelumnya, dan kita dapat menggunakan **tipe data enum** untuk **memastikan jurusan** yang dipilih adalah **jurusan** yang telah **ditentukan sebelumnya** dan hanya **1 jurusan**.

Tipe data ENUM (Contd-2)

```
mysql> CREATE TABLE jurusan (jur ENUM('Ilmu Komputer','Ekonomi','MIPA','Kedo  
Query OK, 0 rows affected (0.07 sec)

mysql> DESCRIBE jurusan;
+-----+-----+-----+
| Field | Type   | Null | Key |
+-----+-----+-----+
| jur   | enum('Ilmu Komputer','Ekonomi','MIPA','Kedokteran') | YES |   |
+-----+-----+-----+
1 row in set (0.01 sec)

mysql> INSERT INTO jurusan VALUES ('Ilmu Komputer');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO jurusan VALUES ('Kedokteran');
Query OK, 1 row affected (0.04 sec)

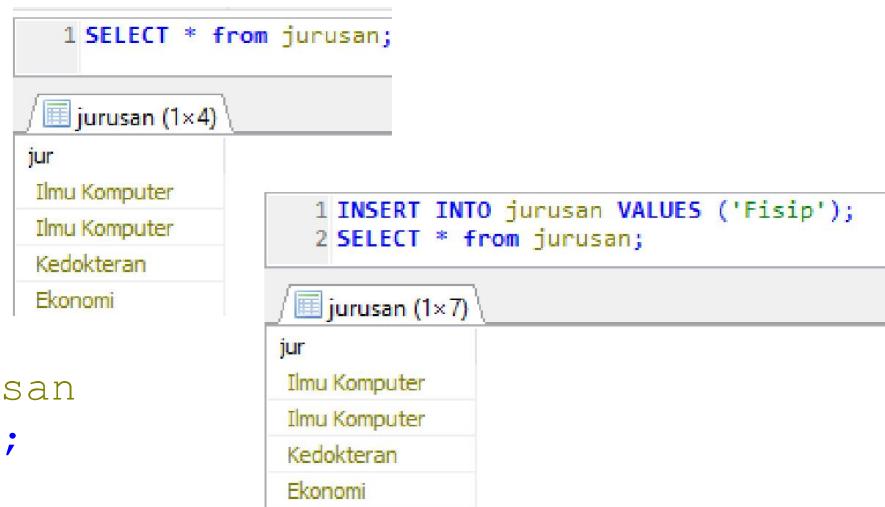
mysql> INSERT INTO jurusan VALUES ('FISIP');
ERROR 1265 (01000): Data truncated for column 'jur' at row 1

mysql> INSERT INTO jurusan VALUES ('Ilmu Komunikasi');
ERROR 1265 (01000): Data truncated for column 'jur' at row 1

mysql> SELECT * FROM jurusan;
+-----+
| jur  |
+-----+
| Ilmu Komputer |
| Kedokteran    |
+-----+
```

```
INSERT INTO jurusan  
VALUES ('Fisip');
```

```
CREATE TABLE jurusan  
(jur ENUM('Ilmu  
Komputer','Ekonomi',  
'MIPA','Kedokteran'));  
  
INSERT INTO jurusan  
VALUES ('Ilmu  
Komputer'), ('Kedokteran'), ('Ekonomi');
```



The screenshot shows two separate MySQL sessions in the MySQL Workbench interface.

Session 1:

```
1 SELECT * from jurusan;
```

Results:

jur
Ilmu Komputer
Ilmu Komputer
Kedokteran
Ekonomi

Session 2:

```
1 INSERT INTO jurusan VALUES ('Fisip');  
2 SELECT * from jurusan;
```

Results:

jur
Ilmu Komputer
Ilmu Komputer
Kedokteran
Ekonomi
Fisip

Tipe data SET (Contd-3)

- Untuk kolom **SET** kita **dapat memilih satu atau lebih** nilai yang tersedia dari **1 sampai 64** pilihan string yang tersedia.
- Contoh : Data **SET** adalah untuk **data** tentang **hobby** seseorang, karena seseorang **memiliki hobi** yang **lebih dari 1**, namun **kita ingin memilih** dari kumpulan **list yang telah kita buat sebelumnya**.

Tipe data SET (Contd-4)

```
mysql> CREATE TABLE hobi (hob SET('Membaca','Menulis','Menggambar','Main Musi
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> DESCRIBE hobi;
+-----+-----+-----+
| Field | Type   | Null | Key |
+-----+-----+-----+
| hob   | set('Membaca','Menulis','Menggambar','Main Musik') | YES  |   |
+-----+-----+-----+
1 row in set (0.01 sec)
```

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

```
mysql> INSERT INTO hobi VALUES ('Membaca,Main Musik');
Query OK, 1 row affected (0.07 sec)
```

```
mysql> INSERT INTO hobi VALUES ('Menggambar,Main Musik');
Query OK, 1 row affected (0.04 sec)
```

```
mysql> INSERT INTO hobi VALUES ('Belajar,Main Musik');
ERROR 1265 (01000): Data truncated for column 'hob' at row 1
```

```
mysql> SELECT * FROM hobi;
+-----+
| hob |
+-----+
| Membaca |
| Membaca,Main Musik |
| Menggambar,Main Musik |
+-----+
```

**INSERT INTO hobi VALUES
('Menggambar,Main Musik');**

**INSERT INTO hobi VALUES
('Memancing,Main Musik');**

Dari query, Untuk menginput lebih dari 1 nilai, kita memisahkan dengan **tanda koma** dan tetap dalam **tanda kutip**, juga seandainya kita coba input hobi '**belajar**', yang memang **tidak ada** sebelumnya, maka MySQL akan mengeluarkan pesan **error**.

```
CREATE TABLE hobi (hob
SET ('Membaca','Menulis','Menggambar','Main Musik'));
```

```
1 INSERT INTO hobi VALUES ('Memancing,Main Musik');
2 SELECT * FROM hobi;
```

hobi (1x2)	
hob	
Menggambar,Main Musik	
Main Musik	

4) Review DDL

1. Struktur SQL
2. CREATE/DROP basisdata
3. CREATE tabel
4. DROP tabel
5. RENAME tabel
6. ALTER tabel

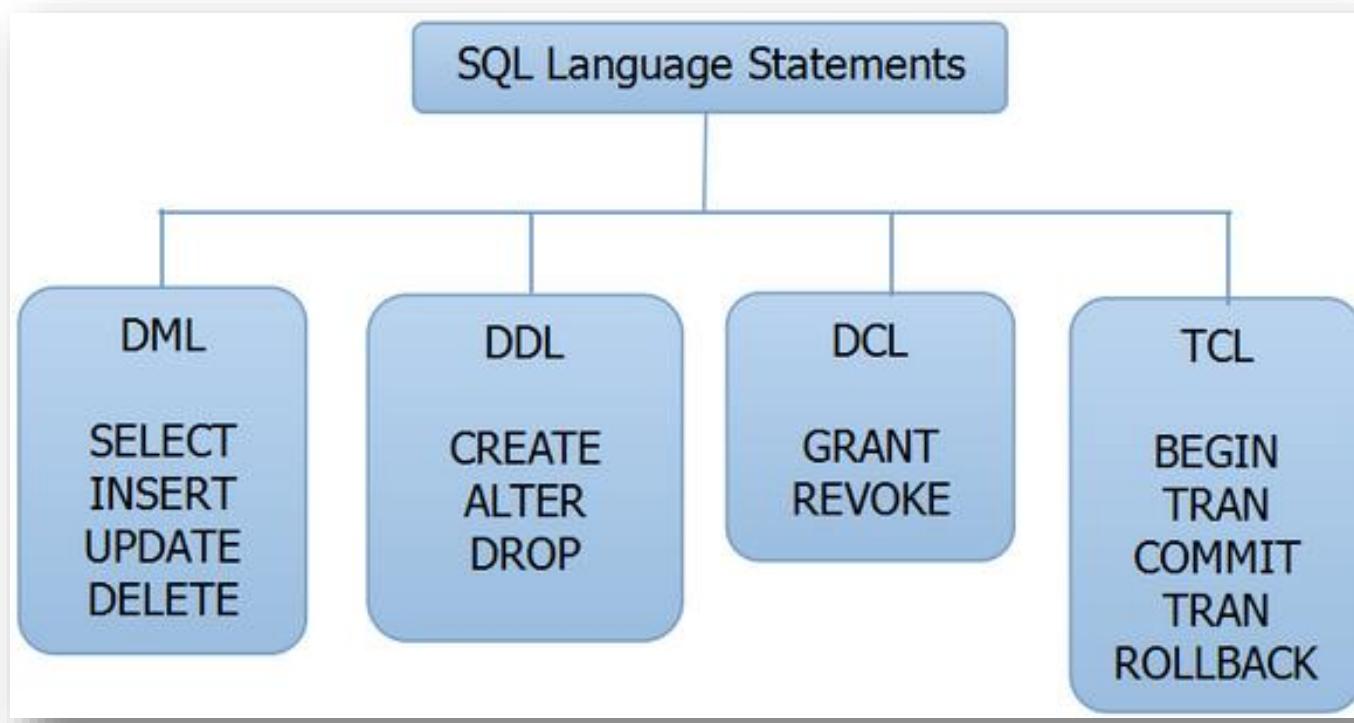
SQL - Structured Language Query

Adalah bahasa standar yang digunakan untuk **memanipulasi basisdata relasional**

- Terdiri dari:
 - **Data Definition Language (DDL):**
 - CREATE tables, indexes, views, Establish primary / foreign keys, DROP / ALTER tables etc
 - **Data Manipulation Language (DML):**
 - INSERT / UPDATE / DELETE, SELECT etc.
 - **Data Control Language (DCL):**
 - COMMIT / ROLLBACK work, GRANT / REVOKE etc

SQL

b) DDL DML

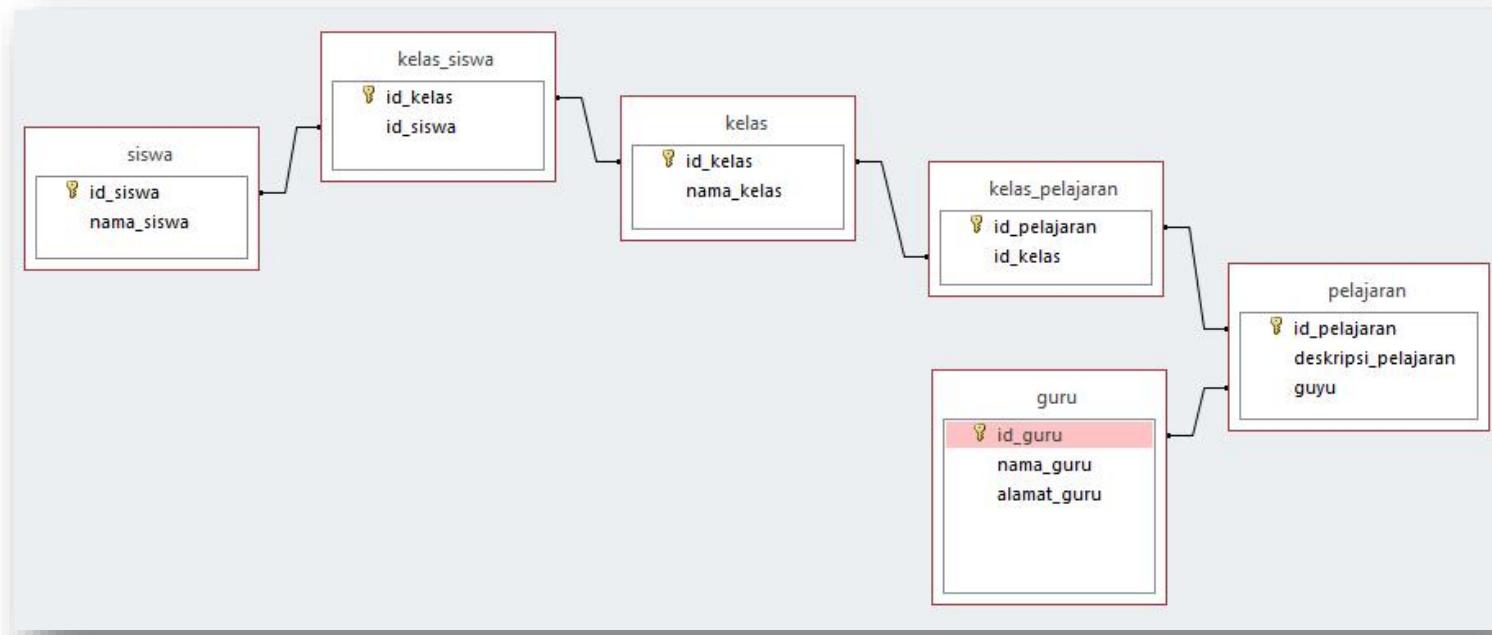


- Data Definition Language (DDL)
- Data Manipulation Language (DML)

SQL

b) DDL

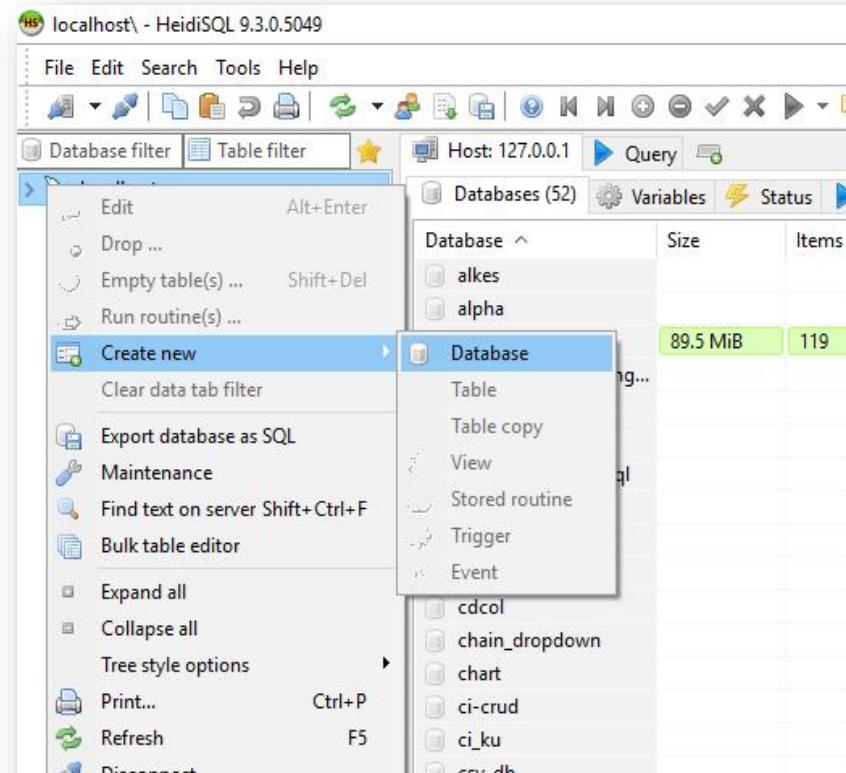
- Pengelolaan pembuatan database dan tabel.
- Dengan berdasar pada relasi table seperti gambar berikut, (nama database = **sekolah**)



SQL

b) DDL Script – buat database

GUI



CLI

```
CREATE DATABASE `sekolah`;
```

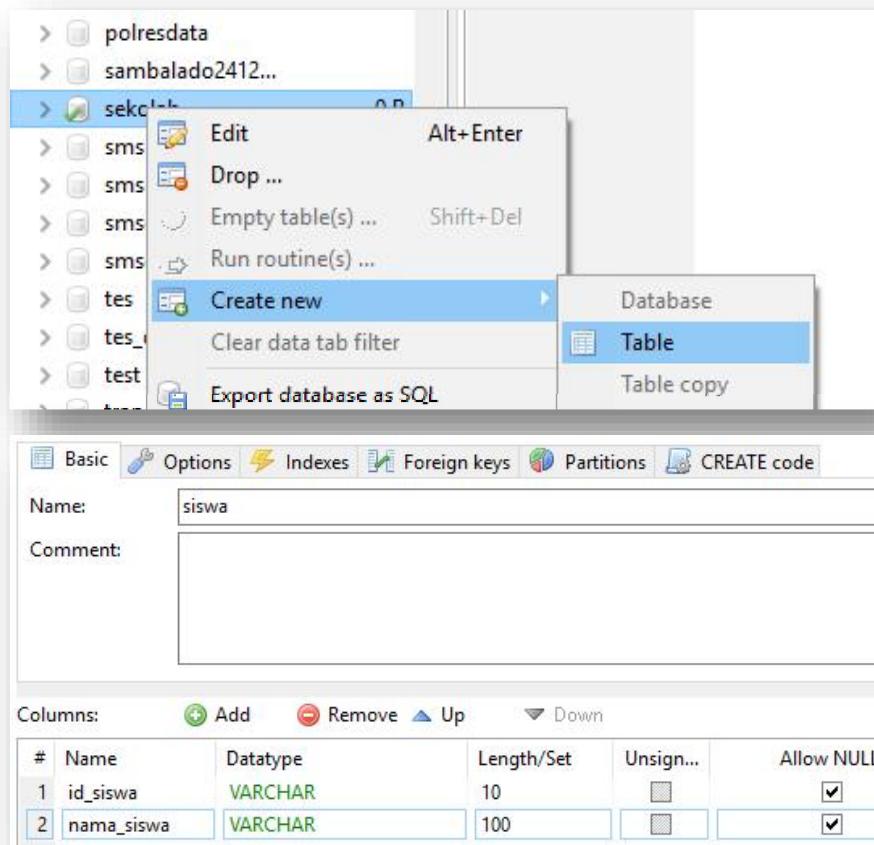
```
USE `sekolah`;
```

```
SHOW TABLES;
```

SQL

b) DDL Script - buat tabel

GUI



CLI

```
CREATE TABLE `siswa` (
  `id_siswa` VARCHAR(10)
NULL,
  `nama_siswa` VARCHAR(100)
NULL
);
```

```
SHOW TABLES;
```

Uji Coba - *CREATE/DROP*basisdata

- **CREATE DATABASE db_name**

Contoh:

- CREATE DATABASE tennis
- USE tennis
- CREATE DATABASE president
- USE president

- **DROP DATABASE db_name**

Contoh:

- DROP DATABASE tennis
- DROP DATABASE president

Uji Coba - *CREATE*tabel

```
CREATE TABLE tbl_name (  
    column_name data_type [DEFAULT  
expr]  
    [column_constraint] , ...  
    [table_constraint] );
```

Uji Coba - *CREATE*tabel (contd-1)

Contoh: Buat tabel Players!

Statement SQL:

```
CREATE TABLE PLAYERS  
  (PLAYERNO SMALLINT NOT NULL,  
   NAME CHAR(15) NOT NULL,  
   INITIALS CHAR(3) NOT NULL,  
   BIRTH_DATE DATE ,  
   SEX CHAR(1) NOT NULL,  
   JOINED SMALLINT NOT NULL,  
   STREET CHAR(15) NOT NULL,  
   HOUSENO CHAR(4) ,  
   POSTCODE CHAR(6) ,  
   TOWN CHAR(10) NOT NULL,  
   PHONENO CHAR(10) ,  
   LEAGUENO CHAR(4) ,  
   PRIMARY KEY (PLAYERNO));
```

Uji Coba - *CREATE*tabel (contd-2)

Contoh: Buat tabel Committee_Members!

Statement SQL:

```
CREATE TABLE COMMITTEE_MEMBERS  
  (PLAYERNO SMALLINT NOT NULL,  
   BEGIN_DATE DATE NOT NULL,  
   END_DATE DATE ,  
   POSITION CHAR(20) ,  
   PRIMARY KEY (PLAYERNO, BEGIN_DATE) );
```

Uji Coba - Skema Tabel (I)

PLAYERS

PLAYERNO (PLAYERNO) NOT NULL	NAME (NAME) NOT NULL	INITIALS (INITIALS) NOT NULL	BIRTH_DATE (DATE)	SEX (SEXCODE) NOT NULL	JOINED (DATE) NOT NULL
------------------------------------	----------------------------	------------------------------------	----------------------	------------------------------	------------------------------

<—PK—>

STREET (STREETNAME) NOT NULL	HOUSENO (HOUSENO)	POSTCODE (POSTCODE)	TOWN (TOWNNAME) NOT NULL	PHONENO (PHONENO)	LEAGUENO (LEAGUENO)
------------------------------------	----------------------	------------------------	--------------------------------	----------------------	------------------------

<—————>

TEAMS

TEAMNO (TEAMNO) NOT NULL	PLAYERNO (PLAYERNO) NOT NULL	DIVISION (DIVISIONNAME) NOT NULL
--------------------------------	------------------------------------	--

<—PK—> <—————>

Uji Coba - Skema Tabel (2)

MATCHES

MATCHENO (MATCHENO) NOT NULL	TEAMNO (TEAMNO) NOT NULL	PLAYERNO (PLAYERNO) NOT NULL	WON (NR_OF_SETS) NOT NULL	LOST (NR_OF_SETS) NOT NULL
------------------------------------	--------------------------------	------------------------------------	---------------------------------	----------------------------------

<—PK—>

PENALTIES

PAYMENTNO (PAYMENTNO) NOT NULL	PLAYERNO (PLAYERNO) NOT NULL	PAYMENT_DATE (DATE) NOT NULL	AMOUNT (AMOUNT) NOT NULL
--------------------------------------	------------------------------------	------------------------------------	--------------------------------

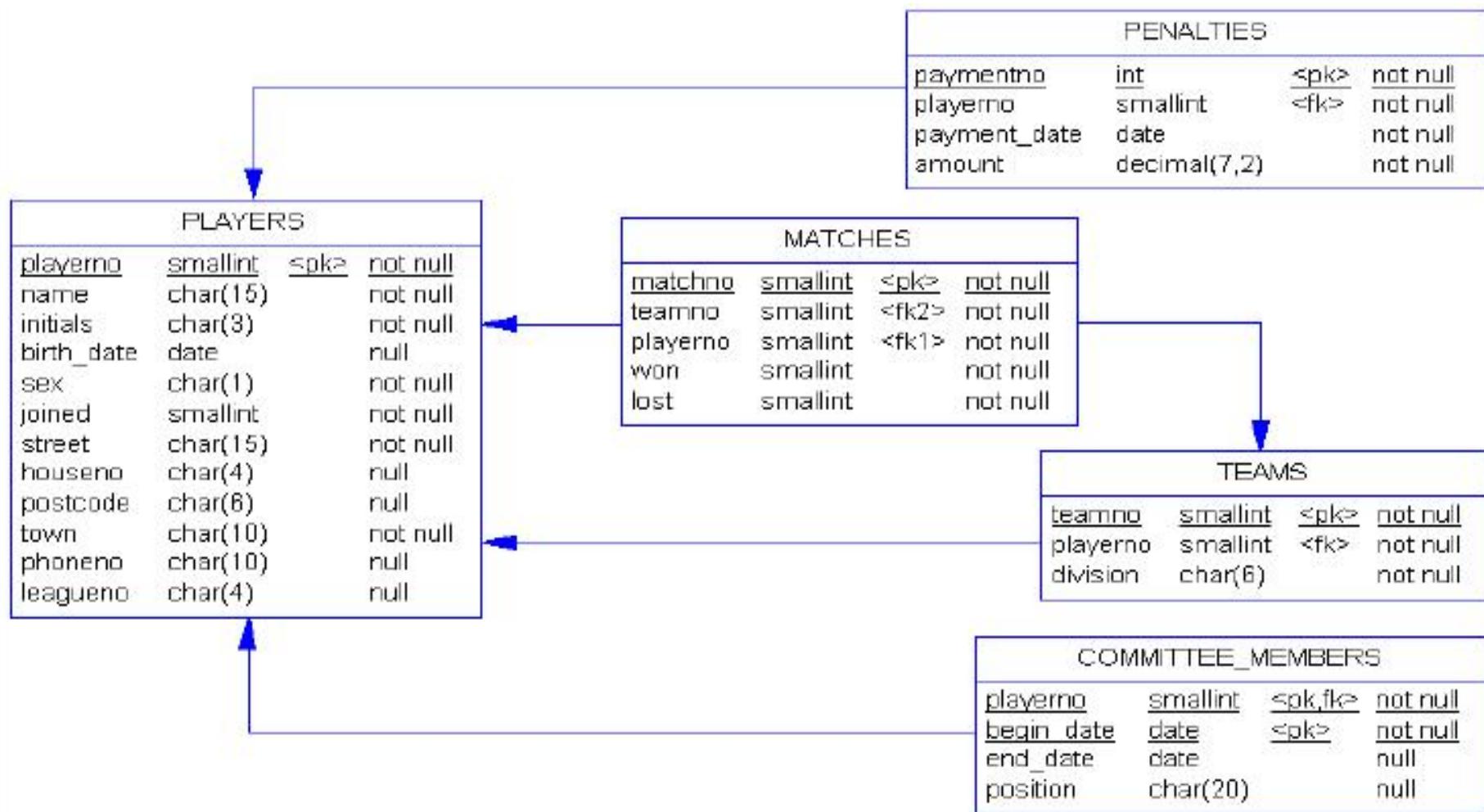
<—PK—>

COMMITTEE_MEMBERS

PLAYERNO (PLAYERNO) NOT NULL	BEGIN_DATE (DATE) NOT NULL	END_DATE (DATE)	POSITION (POSITIONNAME) NOT NULL
------------------------------------	----------------------------------	--------------------	--

<—PK—>

Uji Coba - Skema Basisdata



Uji Coba - Domains & domain constraints

- **PLAYERNO** = NUMERIC (4,0) values
 ≥ 1
- **NAME** = CHARACTER (15)
- **INITIALS** = CHARACTER (3)
- **DATE** = DATE
- **YEARNO** = NUMERIC (4,0)
- **SEXCODE** = CHARACTER (1) set of values = {'M','F'}
- **STREETNAME** = CHARACTER (15)
- **HOUSENO** = CHARACTER (4)
- **POSTCODE** = CHARACTER (6)
- **TOWNNAME** = CHARACTER (10)
- **PHONENO** = CHARACTER (10)
- **LEAGUENO** = CHARACTER (4)
- **TEAMNO** = NUMERIC (2,0) values
 ≥ 1
- **DIVISIONNAME** = CHARACTER (6)
- **MATCHENO** = NUMERIC (4,0) values
 ≥ 1
- **NR_OF_SETS** = NUMERIC (1,0) set of values = {0,1,2,3}
- **PAYMENTNO** = NUMERIC (8,0) values ≥ 1
- **AMOUNT** = NUMERIC (7,2)
- **POSITIONNAME** = CHARACTER (20) set of values = { 'Chairman','Secretary','Treasurer','General member'}

Uji Coba - *Intra table constraints*

Primary keys:

Indicated in the table schema's: <---PK--->

Note: Unique and all columns NOT NULL

Alternative keys:

Indicated in the table schema's:<----->

Note: Unique, not necessary NOT NULL

NOT NULL constraints:

Indicated in the table schema's:NOT NULL

Column value constraints:

PLAYERS (JOINED) values \geq 1970

PENALTIES (PAYMENT_DATE) values \geq '1970-01-01'

PENALTIES (AMOUNT) values $>$ 0.00

COMMITTEE_MEMBERS (BEGIN_DATE) values $>$ '1990-01-01'

Row constraints:

PLAYERS: YEAR (BIRTH_DATE) \leq JOINED

COMMITTEE_MEMBERS: END_DATE \geq BEGIN_DATE

Other intra table constraints:

Note: Not present in this database.

Uji Coba - Inter table constraints

- *Foreign key references:*

TEAMS (PLAYERNO) ---> PLAYERS (PLAYERNO)

MATCHES (TEAMNO) ---> TEAMS (TEAMNO)

MATCHES (PLAYERNO) ---> PLAYERS (PLAYERNO)

PENALTIES (PLAYERNO) ---> PLAYERS (PLAYERNO)

COMMITTEE_MEMBERS (PLAYERNO) ---> PLAYERS
(PLAYERNO)

- *Other inter table constraints:*

For rows of table PLAYERS that are referenced by a foreign key from tables TEAMS, MATCHES and PENALTIES: LEAGUENO IS NOT NULL

For table PENALTIES the value of PAYMENT_DATE must be greater or

equal than the value of JOINED in table PLAYERS for the same player.

Uji Coba - CREATE Tabel (contd-3)

- Contoh: Buat tabel Players!
- Statemen SQL:

```
CREATE TABLE PLAYERS
(PLAYERNO      SMALLINT      NOT NULL,
 NAME          CHAR(15)      NOT NULL,
 INITIALS       CHAR(3)      NOT NULL,
 BIRTH_DATE    DATE         ,
 SEX           CHAR(1)      NOT NULL,
 JOINED        SMALLINT      NOT NULL,
 STREET         CHAR(15)      NOT NULL,
 HOUSENO        CHAR(4)      ,
 POSTCODE       CHAR(6)      ,
 TOWN          CHAR(10)      NOT NULL,
 PHONENO        CHAR(10)      ,
 LEAGUENO        CHAR(4)      ,
 PRIMARY KEY   (PLAYERNO)  ,
 CHECK (PLAYERNO >= 1)    ,
 CHECK (SEX IN ('M', 'F')) ,
 CHECK (JOINED >= 1970)   ,
 CHECK (YEAR (BIRTH_DATE) <= JOINED) );
```

Uji Coba - CREATE Tabel (contd-4)

Contoh: Buat tabel TEAMS!

Statemen SQL:

```
CREATE TABLE TEAMS  
  (TEAMNO SMALLINT NOT NULL,  
   PLAYERNO SMALLINT NOT NULL,  
   DIVISION CHAR(6) NOT NULL,  
   PRIMARY KEY (TEAMNO),  
   UNIQUE (PLAYERNO),  
   FOREIGN KEY (PLAYERNO)  
     REFERENCES PLAYERS (PLAYERNO),  
   CHECK (TEAMNO >= 1);
```

Uji Coba - CREATE Tabel (contd-5)

Contoh: Buat tabel Committee_Members!

Statemen SQL:

```
CREATE TABLE COMMITTEE_MEMBERS  
  (PLAYERNO SMALLINT NOT NULL,  
   BEGIN_DATE DATE NOT NULL,  
   END_DATE DATE ,  
   POSITION CHAR(20) ,  
   PRIMARY KEY (PLAYERNO, BEGIN_DATE) ,  
   FOREIGN KEY (PLAYERNO)  
     REFERENCES PLAYERS (PLAYERNO) ,  
   CHECK (POSITION IN ('Chairman', 'Secretary',  
'Treasurer', 'General member')),  
   CHECK (BEGIN_DATE >= '1990-01-01'),  
   CHECK (END_DATE >= BEGIN_DATE) );
```

Uji Coba - CREATE Tabel (contd-6)

Contoh: Buat tabel TEAMS!

Statemen SQL:

```
CREATE TABLE TEAMS  
  (TEAMNO SMALLINT NOT NULL PRIMARY KEY  
    CHECK (TEAMNO >= 1),  
    PLAYERNO SMALLINT NOT NULL UNIQUE,  
    FOREIGN KEY (PLAYERNO)  
    REFERENCES PLAYERS (PLAYERNO),  
    DIVISION CHAR(6) NOT NULL);
```

Uji Coba - Drop Table

DROP TABLE tbl_name [,tbl_name]

Contoh: Drop seluruh tabel dalam basisdata tennis!

Statemen SQL:

```
DROP TABLE COMMITTEE_MEMBERS;  
DROP TABLE PENALTIES;  
DROP TABLE MATCHES;  
DROP TABLE TEAMS;  
DROP TABLE PLAYERS;
```

Uji Coba - RENAME Tabel

- **RENAME TABLE tbl_name TO new_tbl_name**
[**,tbl_name2 TO new_tbl_name2**] ...

Contoh: Ubah nama tabel Teams menjadi Groups!

Statemen SQL:

RENAME TABLE Teams TO Groups

RENAME tabel

Uji Coba - ALTER Tabel (contd-1)

- **ALTER TABLE *tbl_name***
alter_specification [, alter_specification] ...

alter_specification:

- ADD**
- CHANGE & MODIFY**
- DROP**
- RENAME**

ALTER tabel

Uji Coba - ALTER Tabel ADD

**ADD [COLUMN] col_name column_definition [FIRST | AFTER col_name]
ADD [COLUMN] (col_name column_definition,...)**

Contoh: Tambahkan satu kolom bernama TYPE di dalam tabel TEAMS.
Kolom TYPE ini untuk memberikan identifikasi tim Pria dan Wanita.

Statemen SQL:

```
ALTER TABLE TEAMS  
ADD COLUMN TYPE CHAR (1);
```

Contoh: Dengan adanya kolom TYPE, maka (misalkan) tim 2 sebagai tim Pria harus di-update.

Statemen:

```
UPDATE TEAMS  
SET TYPE = 'M'  
WHERE TEAMNO = 2;
```

ALTER tabel : ADD

Uji Coba - ALTER Tabel CHANGE & MODIFY

**CHANGE [COLUMN] old_col_name new_col_name
column_definition [FIRST|AFTER col_name]**

**MODIFY [COLUMN] col_name column_definition
[FIRST | AFTER col_name]**

Contoh: Tambahkan panjang karakter kolom TOWN
dari 10 menjadi 20!

```
ALTER TABLE PLAYERS  
MODIFY COLUMN TOWN CHAR (20);
```

ALTER tabel : CHANGE & MODIFY

Latihan - DROP

DROP [COLUMN] col_name

DROP PRIMARY KEY

DROP FOREIGN KEY fk_symbol

Contoh: Hapuskan kolom TYPE dari tabel TEAMS!

Statemen:

ALTER TABLE TEAMS

DROP COLUMN TYPE

Uji Coba - ALTER Tabel RENAME

RENAME [TO] new_tbl_name

Contoh: Ubah nama tabel Teams menjadi Groups!

Statemen SQL:

ALTER TABLE Teams RENAME Groups

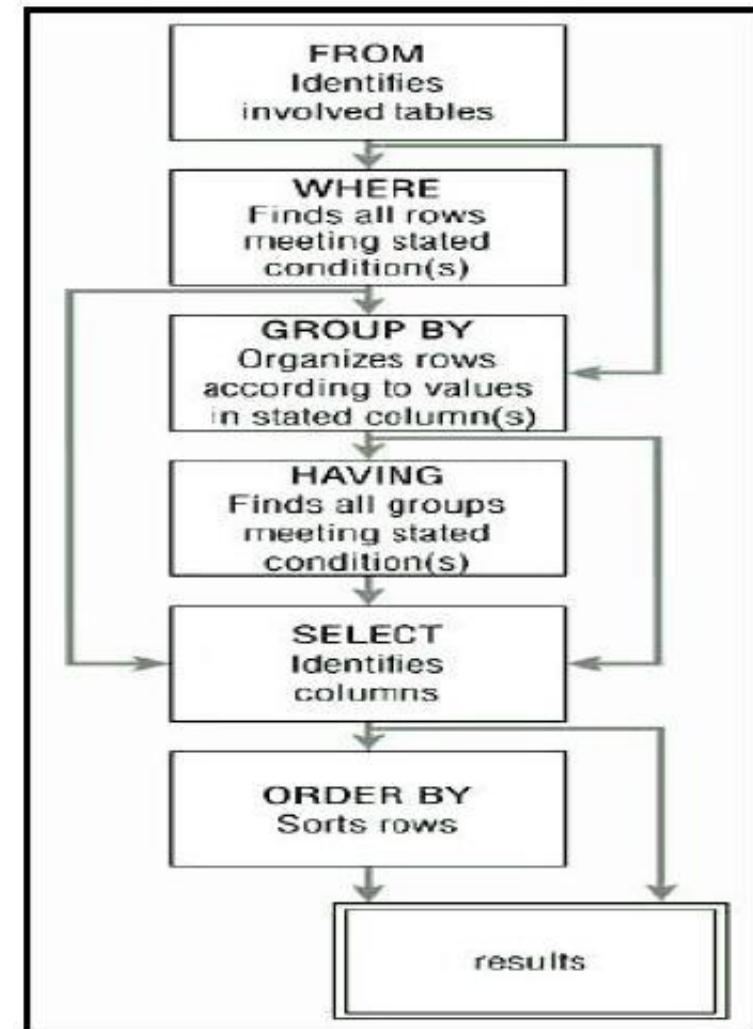
ALTERtabel : RENAME

DML (DATA MANIPULATION LANGUAGE)

- 1) *INSERT* data ke dalam tabel
- 2) *SELECT*(dari 1 tabel)
- 3) *SELECT*(lebih dari 1 tabel)
- 4) *UPDATE* data dalam tabel
- 5) *DELETE* data dalam tabel

Data Manipulation Language (DML)

- *INSERT / UPDATE / DELETE, SELECT*
- Klausula-klausula *SELECT*:
 - **FROM** --> menentukan tabel(-tabel) sumber
 - **WHERE** --> memilih baris(-baris) yang memenuhi kondisi (-kondisi)
 - **GROUP BY** --> menggabungkan baris(-baris) yang kolomnya memiliki nilai yang sama
 - **HAVING** --> memilih grup yang memenuhi kondisi tertentu
 - **SELECT** --> memilih kolom(-kolom)
 - **ORDER BY** --> mengurutkan baris-baris berdasarkan nilai-nilai yang ada dalam kolom(-kolom)



SQL

c) DML

- ❑ Pengelolaan **data** dalam tabel.
- ❑ Bentuk **CRUD**:
 1. **Create**
 2. **Read** (Max, Min, Sum, dll)
 3. **Update**
 4. **Delete**

Start MySQL

```
C:\Users\Smart>
```

```
cd c:\xampp\mysql\bin\
```

```
c:\xampp\mysql\bin>
```

```
mysql -u root -p
```

Enter password:

Welcome to the MariaDB monitor. Commands end with ; or \g.

Microsoft Windows
[Version 6.1.7600]

```
MariaDB [(none)]> show databases;
```

Database
fifin
information_schema
mysql
performance_schema
phpmyadmin
test

6 rows in set (0.11 sec)

```
MariaDB [(none)]> use fifin;
```

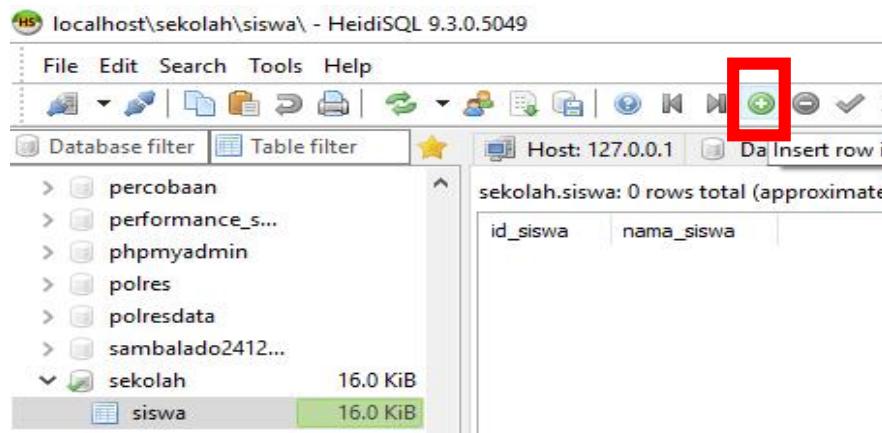
Database changed

```
MariaDB [fifin]>
```

SQL

c) DML Script - Create

GUI



This screenshot shows the same HeidiSQL interface after an insertion. The status bar now says 'Host: 127.0.0.1 Database: Post (Ctrl+Enter)' and 'sekolah.siswa: 1 rows total (approximately)'. The table data shows one row: 'id_siswa' is 2016111234 and 'nama_siswa' is Furi hikmawati. A red box highlights the green checkmark icon in the toolbar, indicating the operation was successful.

id_siswa	nama_siswa
2016111234	Furi hikmawati

CLI

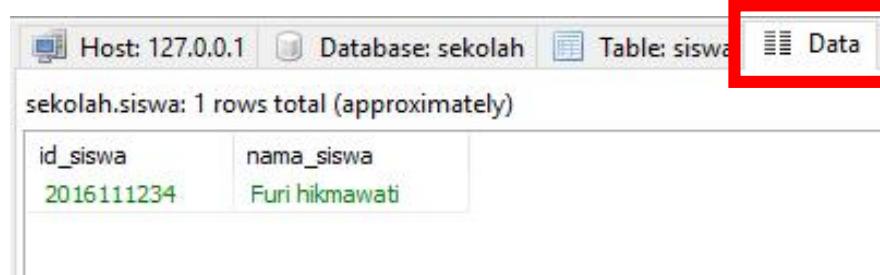
INSERT INTO

```
 `sekolah`.`siswa`  
(`id_siswa`,  
 `nama_siswa`) VALUES  
('2016111234', 'Furi  
Hikmawati');
```

SQL

c) DML Script - Read

GUI



A screenshot of the MySQL Workbench interface. At the top, there are tabs for Host: 127.0.0.1, Database: sekolah, Table: siswa, and Data (which is highlighted with a red box). Below the tabs, a message says "sekolah.siswa: 1 rows total (approximately)". A table below shows one row of data:

id_siswa	nama_siswa
2016111234	Furi hikmawati

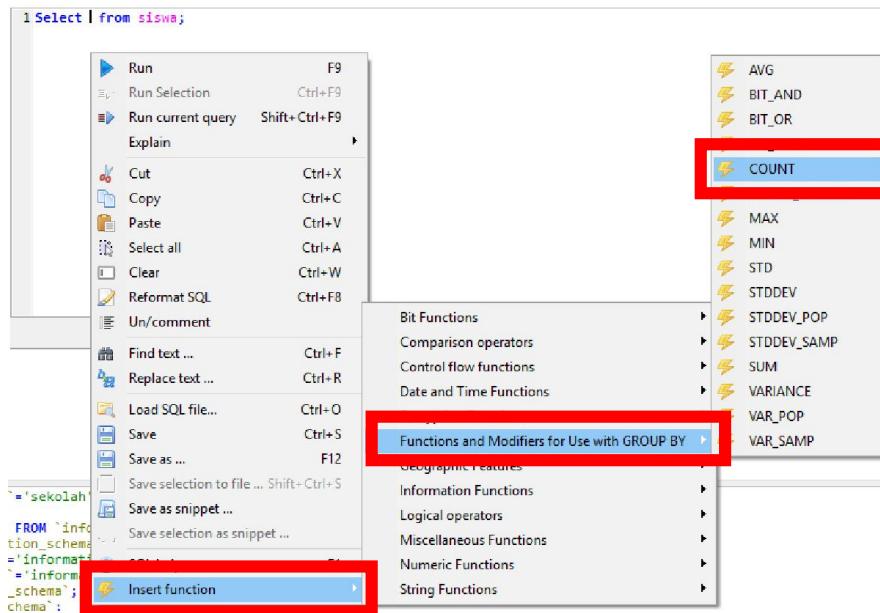
CLI

```
SELECT * FROM  
`sekolah`.`siswa`;
```

SQL

c) DML Script - Read (count)

GUI



CLI

```
Select COUNT(*) from siswa;
```

SQL

c) DML Script – Read (count)

Statement SELECT

Cari nomor pemain yang telah melakukan setidaknya dua kali penalti yang jumlahnya lebih dari \$25! Urutkan hasil berdasarkan nomor pemainnya!

Query:

```
SELECT PLAYERNO FROM PENALTIES WHERE AMOUNT >  
25 GROUP BY PLAYERNO HAVING COUNT (*) > 1 ORDER  
BY PLAYERNO;
```

SQL

c) DML Script – Read (count)

Query:

```
SELECT          PLAYERO NO  
FROM           PENALTIES  
WHERE          AMOUNT > 25  
GROUP BY       PLAYERO NO  
HAVING         COUNT (*) > 1  
ORDER BY       PLAYERO NO;
```

FROM

PAYMENTNO	PLAYERO NO	PAYMENT_DATE	AMOUNT
1	6	1980-12-08	100.00
2	44	1981-05-05	75.00
3	27	1983-09-10	100.00
4	104	1984-12-08	50.00
5	44	1980-12-08	25.00
6	8	1980-12-08	25.00
7	44	1982-12-30	30.00

WHERE

PAYMENTNO	PLAYERO NO	PAYMENT_DATE	AMOUNT
1	6	1980-12-08	100.00
2	44	1981-05-05	75.00
3	27	1983-09-10	100.00
4	104	1984-12-08	50.00
7	44	1982-12-30	30.00
8	27	1984-11-12	75.00

GROUP BY

PAYMENTNO	PLAYERO NO	PAYMENT_DATE	AMOUNT
1	6	1980-12-08	100.00
2	44	1981-05-05	75.00
7	44	1982-12-30	30.00
3	27	1983-09-10	100.00
8	27	1984-11-12	75.00
4	104	1984-12-08	50.00

HAVING

PAYMENTNO	PLAYERO NO	PAYMENT_DATE	AMOUNT
2	44	1981-05-05	75.00
7	44	1982-12-30	30.00
3	27	1983-09-10	100.00
8	27	1984-11-12	75.00

SELECT

PLAYERO NO
44
27

ORDER BY

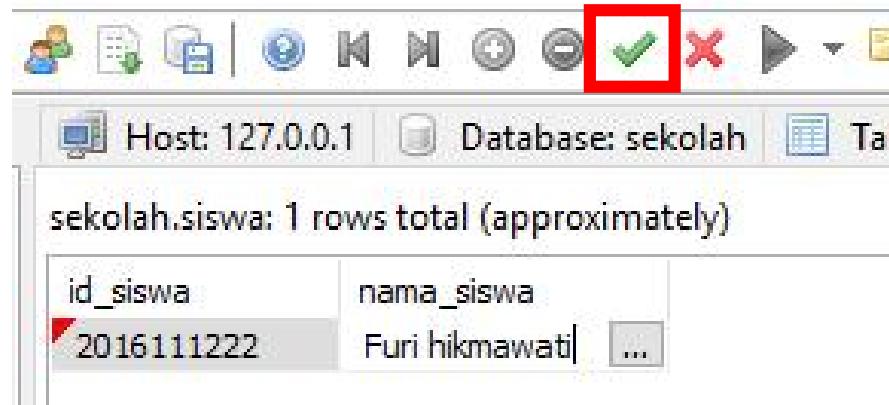
PLAYERO NO
27
44

A
G

SQL

c) DML Script - Update

GUI



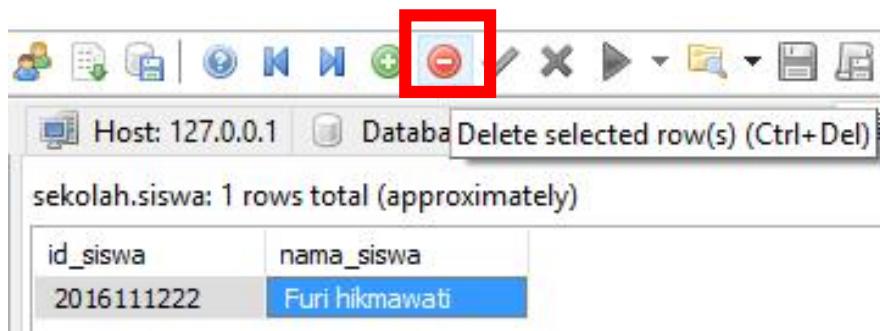
CLI

```
UPDATE `sekolah`.`siswa`
SET
`id_siswa`='2016111222'
WHERE
`id_siswa`='2016111234'
AND `nama_siswa`='Furi
hikmawati' LIMIT 1;
```

SQL

c) DML Script - Delete

GUI



CLI

DELETE FROM

```
`sekolah`.`siswa` WHERE
`id_siswa`='2016111222'
AND `nama_siswa`='Furi
hikmawati' LIMIT 1;
```

Latihan DML

DML: Insert data ke dalam Tabel

1. INSERT satu baris

INSERT INTO <table specification>

[<column list>]

VALUES (<expression> [{,<expression>}...])

2. INSERT dari tabel lain (Mengisi suatu tabel dari isi tabel lain)

INSERT INTO<table specification>

[<column list>]

<select clause>

<from clause>

[<where clause>]

[<group by clause>]

[<having clause>]

DML: Insert data ke dalam Tabel (contd -2)

Contoh 1: Sebuah tim baru dengan pemain nomor 100 sebagai kaptennya telah bergabung di dalam liga. Tim ketiga ini akan bermain di dalam divisi ketiga.

Query:

```
INSERT INTO TEAMS (TEAMNO, PLAYERNO, DIVISION)  
VALUES (3,100,'third');
```

atau:

```
INSERT INTO TEAMS VALUES (3,100,'third');
```

DML: Insert data dari Tabel lain (contd -3)

- **Contoh 1.1:** Buatlah suatu tabel baru yang mencatat nomor pemain, nama, kota dan nomor telepon dari masing-masing pemain yang tidak memiliki nomor liga (tidak pernah bermain di dalam liga)!

Query:

- Membuat tabel baru:

```
CREATE TABLE RECR_PLAYERS  
(PLAYERNO SMALLINT NOT NULL,  
NAME CHAR(15) NOT NULL,  
TOWN CHAR(10) NOT NULL,  
PHONENO CHAR(10) ,  
PRIMARY KEY (PLAYERNO) )
```

- Insert data ke tabel RECR_PLAYERS dari tabel PLAYERS:

```
INSERT INTO RECR_PLAYERS  
SELECT PLAYERNO, NAME, TOWN, PHONENO  
FROM PLAYERS  
WHERE LEAGUENO IS NULL
```

KLAUSA : SELECT (dari 1 tabel)

Contoh 2: Tampilkan seluruh data penalti.

Query:

```
SELECT * FROM Penalties;
```

Contoh 3: Tampilkan nomor pembayaran, nomor pemain dan jumlah dari masing-masing penalti

Query:

```
SELECT paymentno, playerno, amount  
FROM Penalties;
```

Contoh 3.1 : Tampilkan nomor pemain, nama dan umur ketika bergabung di klub tenis dari masing-masing pemain.

Query:

```
SELECT playerno, name,  
       joined - Year(birth_date) AS join_age  
FROM Players
```

KLAUSA : SELECT - WHERE (dari 1 tabel)

Contoh 4: Dapatkan nomor pemain dan nomor liga dari masing-masing pemain yang memang benar-benar memiliki nomor liga.

Query:

```
SELECT playerno, leagueno  
FROM Players  
WHERE leagueno IS NOT NULL;
```

KLAUSA : SELECT - BETWEEN (dari 1 tabel)

Contoh 5: Cari nomor dan tanggal lahir dari masing-masing pemain yang lahir antara tahun 1962 sampai 1964.

Query:

```
SELECT playerno, birth_date  
FROM Players  
WHERE Year(birth_date) BETWEEN 1962 AND  
1964;
```

KLAUSA : SELECT - IN (dari 1 tabel)

Contoh 6: Cari nomor, nama dan kota dari masing-masing pemain yang tinggal di *Inglewood*, *Plymouth*, *Midhurst* atau *Douglas*.

Query dengan OR:

```
SELECT playerno, name, town  
FROM Players  
WHERE town = 'Inglewood' OR town = 'Plymouth' OR  
town = 'Midhurst' OR town = 'Douglas';
```

Query dengan IN:

```
SELECT playerno, name, town  
FROM Players  
WHERE town IN ('Inglewood', 'Plymouth', 'Midhurst', 'Douglas');
```

KLAUSA : SELECT – LIKE, ORDER BY (dari 1 tabel)

Contoh 7: Dapatkan nama dan nomor dari masing-masing pemain yang memiliki huruf “e” pada posisi huruf sebelum huruf terakhir dari namanya.

Query:

```
SELECT name, playerno  
FROM Players  
WHERE name LIKE '%e_';
```

Contoh 8: Buat daftar nomor pembayaran dan nomor pemain untuk masing-masing penalti; urutkan hasil berdasarkan nomor pemain dan nomor pembayaran untuk masing-masing pemain.

Query:

```
SELECT paymentno, playerno  
FROM Penalties  
ORDER BY playerno, paymentno;
```

Seluruh Klausa dalam 1 Statement

- **Contoh 8.1:** Cari nomor dari masing-masing pemain yang telah melakukan lebih dari 1 penalti yang besarnya lebih dari 25.
- Query:

```
SELECT playerno  
FROM Penalties  
WHERE amount > 25  
GROUP BY playerno  
HAVING COUNT(*) > 1  
ORDER BY playerno
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Count

Contoh 9: Hitung jumlah pemain yang tercatat di dalam tabel PLAYERS!

SQL:

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

Contoh 10: Ada berapa nama kota yang tercatat di dalam kolom TOWN dalam tabel PLAYERS?

SQL:

```
SELECT COUNT(DISTINCT(TOWN)) FROM PLAYERS;
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Max & Min

Contoh 11: Berapakah jumlah penalti yang tertinggi?

SQL:

```
SELECT MAX(AMOUNT) FROM PENALTIES;
```

Contoh 12: Berapakah jumlah penalti yang terendah?

SQL:

```
SELECT MIN(AMOUNT) FROM PENALTIES;
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Sum

Contoh 13: Berapa banyak total SET yang telah dimenangkan, total SET yang telah kalah, dan berapa perbedaan di antara keduanya?

SQL:

```
SELECT SUM(WON),SUM(LOST),SUM(WON)-  
SUM(LOST) AS Difference
```

```
FROM MATCHES;
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Avg

Contoh 14: Berapakah jumlah rata-rata penalti yang dilakukan oleh pemain nomor 44?

SQL:

```
SELECT AVG(AMOUNT)  
FROM PENALTIES  
WHERE PLAYERNO = 44;
```

KLAUSA : SELECT (lebih dari 1 tabel)

Join: Inner

Contoh 15: Tampilkan nomor tim dan nama kapten dari masing-masing tim

Solusi menggunakan “Equi JOIN”:

Query:

```
SELECT TEAMNO, NAME  
FROM TEAMS, PLAYERS  
WHERE TEAMS.PLAYERNO = PLAYERS.PLAYERNO;
```

Solusi menggunakan INNER JOIN:

Query:

```
SELECT TEAMNO, NAME  
FROM PLAYERS P INNER JOIN TEAMS T  
ON (P.PLAYERNO = T.PLAYERNO);
```

KLAUSA : SELECT (lebih dari 1 tabel)

Join: Outer (Left/Right)

Contoh 16: Untuk masing-masing pemain, buatlah daftar nama dan nomor teleponnya (jika memang terdaftar)! Khusus untuk pemain yang menjadi kapten suatu tim, cantumkan juga nomor timnya dan divisinya.

Solusi menggunakan **LEFT JOIN**:

```
SELECT NAME, PHONENO, TEAMNO, DIVISION  
FROM PLAYERS AS P LEFT JOIN TEAMS AS T  
ON P.PLAYERNO = T.PLAYERNO;
```

Solusi menggunakan **RIGHT JOIN**:

```
SELECT NAME, PHONENO, TEAMNO, DIVISION  
FROM TEAMS AS T RIGHT JOIN PLAYERS AS P  
ON T.PLAYERNO = P.PLAYERNO;
```

KLAUSA : SELECT (lebih dari 1 tabel)

Contoh 17: Cari nama dan inisial pemain yang telah bermain di dalam pertandingan minimal sebanyak 1 kali!

Query:

```
SELECT DISTINCT P.NAME, P.INITIALS  
FROM PLAYERS AS P, MATCHES AS M  
WHERE P.PLAYERNO = M.PLAYERNO;
```

KLAUSA : SELECT (lebih dari 1 tabel)

Contoh 18: Cari nomor dan nama pemain yang tinggal di kota yang sama dengan pemain nomor 27!

Query:

SELECT P.PLAYERNO, P.NAME

FROM PLAYERS AS P, PLAYERS AS P27

WHERE P.TOWN = P27.TOWN

AND P27.PLAYERNO = 27

AND P.PLAYERNO <> 27

UPDATE data dalam Tabel

```
UPDATE <table specification>
SET <column name = expression>
[ WHERE <condition> ]
```

Contoh 19: Keluarga Parmenter telah pindah rumah. Sekarang mereka tinggal di Palmer Street nomor 83, kota Inglewood, kode pos 1234UU. Nomor telepon mereka yang baru belum diketahui.

Query:

```
UPDATE PLAYERS
SET STREET = 'Palmer Street', HOUSENO = '83',
    TOWN = 'Inglewood', POSTCODE = '1234UU',
    PHONENO = NULL
WHERE NAME = 'Parmenter';
```

UPDATE data dalam Tabel

Contoh 20: Naikkan jumlah penalti sebanyak 5%!

Query:

```
UPDATE PENALTIES  
SET AMOUNT = AMOUNT * 1.05;
```

DELETE data dalam Tabel

```
DELETE  
FROM <table specification>  
[ WHERE <condition> ]
```

Contoh 21: Hapus seluruh data penalti yang dilakukan oleh pemain nomor 44 pada tahun 1980!

Query:

```
DELETE  
FROM PENALTIES  
WHERE PLAYERNO = 44  
AND YEAR(PAYMENT_DATE) = 1980;
```

Function

Operator IS NULL

Operator IN dalam subquery

Operator EXISTS

Operator ALL & ANY

DISTINCT

Fungsi COUNT

Fungsi MAX dan MIN

Fungsi SUM

Fungsi AVG

Ekspresi CASE

Operator IS NULL

- ❑ Digunakan untuk memilih baris (record) dalam kolom tertentu yang tidak memiliki nilai
- ❑ Contoh 1: Buat daftar seluruh pemain yang memiliki nomor liga!

Query:

```
SELECT PLAYERNO, LEAGUENO  
FROM PLAYERS  
WHERE LEAGUENO IS NOT NULL
```

Contoh 2: Cari nomor pemain yang tidak memiliki nomor liga!

Query:

```
SELECT PLAYERNO  
FROM PLAYERS  
WHERE LEAGUENO IS NULL
```

Operator IN dalam subquery (contd-1)

- Contoh 3: Cari nomor, nama dan inisial dari masing-masing pemain yang telah bermain minimal satu kali pertandingan

Query:

```
SELECT PLAYERNO  
FROM MATCHES
```

Query:

```
SELECT PLAYERNO, NAME, INITIALS  
FROM PLAYERS  
WHERE PLAYERNO IN  
(2,6,8,27,44,57,83,104,112)
```

Operator IN dalam subquery (contd-2)

- Contoh 3: Cari nomor, nama dan inisial dari masing-masing pemain yang telah bermain minimal satu kali pertandingan
- Query:

```
SELECT PLAYERNO, NAME, INITIALS  
FROM PLAYERS  
WHERE PLAYERNO IN  
(SELECT PLAYERNO  
FROM MATCHES)
```

Intermediate Result:
(2,6,8,27,44,57,83,104, 112)

Operator IN dalam subquery (contd-3)

- Contoh 4: Cari nomor dan nama dari masing-masing pemain dari tim 1 yang telah bermain setidaknya satu kali pertandingan!

Query:

```
SELECT PLAYERNO, NAME,  
FROM PLAYERS  
WHERE PLAYERNO IN  
(SELECT PLAYERNO  
FROM MATCHES)  
WHERE TEAMNO = 1)
```



Intermediate Result:
(2,6,8,44,57,83)

Operator IN dalam subquery (contd-4)

- Contoh 5: Cari nomor dan nama dari masing-masing pemain yang telah bermain setidaknya satu kali pertandingan dari suatu tim yang kaptennya bukan pemain no 6!

Query:

```
SELECT PLAYERNO, NAME,  
      FROM PLAYERS  
      WHERE PLAYERNO IN  
            (SELECT PLAYERNO  
              FROM MATCHES  
              WHERE TEAMNO NOT IN  
                    (SELECT TEAMNO  
                      FROM TEAMS  
                      WHERE PLAYERNO = 6))
```

Intermediate Result:
(8,27,104,112)

Intermediate Result:
(1)

Operator EXISTS

- Contoh 6: Cari nama dan inisial dari masing-masing pemain yang telah melakukan minimal satu kali penalti!

Query 1:

```
SELECT NAME, INITIALS  
FROM PLAYERS  
WHERE PLAYERNO IN  
(SELECT PLAYERNO  
FROM PENALTIES)
```

Query 2:

```
SELECT NAME, INITIALS  
FROM PLAYERS  
WHERE EXISTS  
(SELECT *  
FROM PENALTIES  
WHERE PLAYERNO = PLAYERS.PLAYERNO)
```

Operator EXISTS (contd-2)

- Contoh 7: Cari nama dan inisial dari para pemain yang tidak menjadi kapten dalam suatu tim!
- Query:

```
SELECT NAME, INITIALS  
FROM PLAYERS  
WHERE NOT EXISTS
```

```
(SELECT * FROM TEAMS  
WHERE PLAYERNO = PLAYERS.PLAYERNO)
```

Operator ALL & ANY

- ❑ Fungsinya seperti Operator IN dalam subquery
- ❑ Contoh 8: Cari nomor, nama dan tanggal lahir dari pemain yang usianya paling tua! Pemain yang paling tua adalah pemain yang tanggal lahirnya adalah lebih kecil atau sama dengan pemain-pemain yang lain.

- ❑ Query:

```
SELECT PLAYERNO, NAME, BIRTH_DATE  
FROM PLAYERS  
WHERE BIRTH_DATE <= ALL  
      (SELECT BIRTH_DATE  
       FROM PLAYERS)
```

Operator ALL & ANY (contd 2)

- ❑ Contoh 9: Cari nomor, nama dan tanggal lahir dari masingmasing pemain terkecuali pemain yang usianya paling tua!
- ❑ Query:

```
SELECT PLAYERNO, NAME, BIRTH_DATE  
FROM PLAYERS  
WHERE BIRTH_DATE > ANY  
      (SELECT BIRTH_DATE  
       FROM PLAYERS)
```

DISTINCT

- Menghilangkan baris yang isinya sama
- Contoh 10: Cari nama-nama kota yang berbeda dalam tabel PLAYERS!
- Query:
SELECT DISTINCT TOWN FROM PLAYERS

Function dalam SELECT

- COUNT ([DISTINCT | ALL] { * | <expression> }) |
- MIN ([DISTINCT | ALL] <expression>) |
- MAX ([DISTINCT | ALL] <expression>) |
- SUM ([DISTINCT | ALL] <expression>) |
- AVG ([DISTINCT | ALL] <expression>)

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Count

Contoh 9: Hitung jumlah pemain yang tercatat di dalam tabel PLAYERS!

SQL:

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

Contoh 10: Ada berapa nama kota yang tercatat di dalam kolom TOWN dalam tabel PLAYERS?

SQL:

```
SELECT COUNT(DISTINCT(TOWN)) FROM PLAYERS;
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Max

Contoh 11: Berapakah jumlah penalti yang tertinggi?

SQL:

```
SELECT MAX(AMOUNT) FROM PENALTIES;
```

Contoh 11.1: Buatlah daftar nomor dan tanggal lahir para pemain yang lahir di tahun yang sama dengan pemain termuda yang bermain untuk Tim 1!

Query:

```
SELECT PLAYERNO, BIRTH_DATE  
FROM PLAYERS  
WHERE YEAR(BIRTH_DATE) =  
      (SELECT MAX(YEAR(BIRTH_DATE))  
       FROM PLAYERS  
       WHERE PLAYERNO IN  
             (SELECT PLAYERNO  
              FROM MATCHES  
              WHERE TEAMNO = 1))
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Min

Contoh 12: Berapakah jumlah penalti yang terendah?

SQL:

```
SELECT MIN(AMOUNT) FROM PENALTIES;
```

Contoh 12.1 : Berapakah nilai/jumlah penalti terendah yang dilakukan oleh pemain yang bertempat tinggal di Stratford?

Query:

```
SELECT MIN(AMOUNT)  
FROM PENALTIES  
WHERE PLAYERNO IN  
(SELECT PLAYERNO  
FROM PLAYERS  
WHERE TOWN = 'Stratford')
```

Contoh 12.2 : Berapa banyak penalti yang jumlahnya adalah sama dengan nilai/jumlah penalti yang terendah?

Query:

```
SELECT COUNT(*)  
FROM PENALTIES  
WHERE AMOUNT =  
(SELECT MIN(AMOUNT)  
FROM PENALTIES)
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Sum

Contoh 13: Berapa banyak total SET yang telah dimenangkan, total SET yang telah kalah, dan berapa perbedaan di antara keduanya?

SQL:

```
SELECT SUM(WON),SUM(LOST),SUM(WON)-SUM(LOST) AS Difference
FROM MATCHES;
```

Contoh 13.1 : Berapakah jumlah total penalti yang dilakukan oleh pemain yang berasal dari Inglewood?

Query:

```
SELECT SUM(AMOUNT)
FROM PENALTIES
WHERE PLAYERO NO IN
  (SELECT PLAYERO NO
    FROM PLAYERS
    WHERE TOWN = 'Inglewood')
```

KLAUSA : SELECT (dari 1 tabel)

Fungsi dalam Select: Avg

Contoh 14: Berapakah jumlah rata-rata penalti yang dilakukan oleh pemain nomor 44?

SQL:

```
SELECT AVG(AMOUNT)
FROM PENALTIES
WHERE PLAYERNO = 44;
```

Contoh 14.1 : Berapakah rata-rata penalti yang dilakukan oleh para pemain yang bermain untuk Tim 1?

Query:

```
SELECT AVG(AMOUNT)
FROM PENALTIES
WHERE PLAYERNO IN
  (SELECT PLAYERNO
   FROM MATCHES
   WHERE TEAMNO = 1)
```

Ekspresi CASE

- Merupakan suatu bentuk ekspresi CASE di dalam SQL (serupa dengan pernyataan IF-THEN-ELSE)

<case expression>

CASE <expression>

WHEN <expression> THEN <expression>

ELSE <expression>

END

Ekspresi CASE (contd-2)

- Contoh 15 : Cari nomor, jenis kelamin dan nama dari masingmasing pemain yang telah bergabung di dalam klub setelah tahun 1980! Jenis kelamin harus tercetak sebagai 'Female' atau 'Male' dan bukannya 'F' atau 'M' saja.
- Query:

```
SELECT PLAYERO,
      CASE SEX
          WHEN 'F' THEN 'Female'
          ELSE 'Male' END AS GENDER,
      NAME
FROM PLAYERS
WHERE JOINED > 1980
```

5) Kontrak Perkuliahan

- a). Tujuan Perkuliahan
- b). Metode Pengajaran
- c). Metode Penilaian
- d). Tugas dan Projek

Learning Outcomes

Diharapkan mahasiswa mampu:

- Merancang Dan Memodelkan Basis Data
- Melakukan Desain Database Dengan Benar
- Menggunakan Bahasa Query Dan Menjelaskan Konsep Pemrosesan Query
- Menyusun Stored Procedure Dan Trigger Yang Optimal
- Menerapkan Atau Mengimplementasi SMBD Pada Aplikasi Yang Sesuai.

Metode Pengajaran

□ Tatap muda di kelas & Praktikum

- Memberikan framework atau roadmap untuk mengorganisasi informasi mengenai perkuliahan
- Menjelaskan subjek dan perkuat gagasan besar yang penting
- Mengimplementasikan hasil perkuliahan pada praktikum di laboratorium.

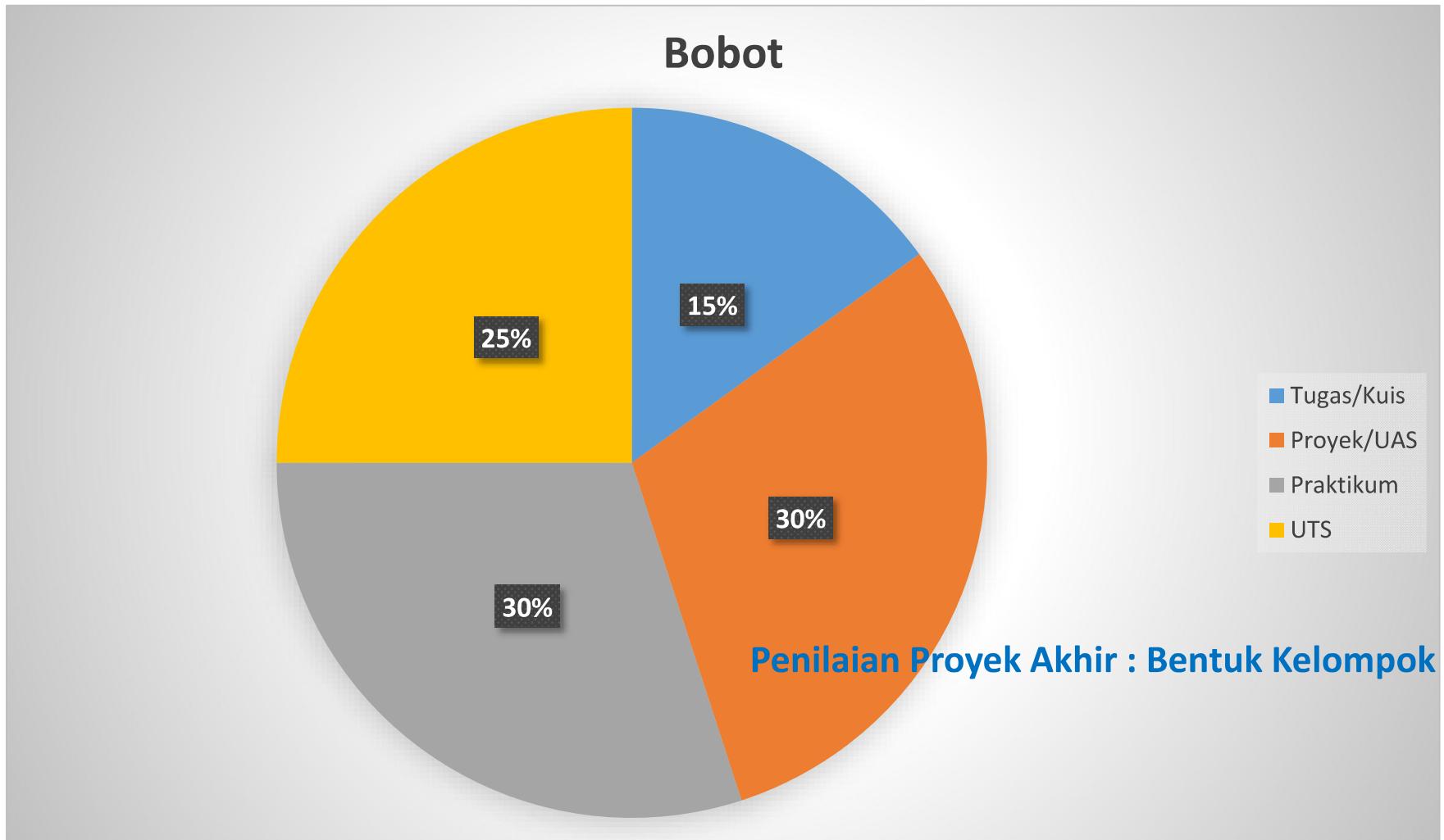
□ Bimbingan dan Arahan

- Meminta mahasiswa mengungkapkan apa yang belum dimengerti, sehingga Dosen dapat membantunya
- Mempersilakan mahasiswa mempraktikkan keterampilan yang diperlukan untuk menguasai penerapannya

Tata Tertib Perkuliahan

- Masuk sesuai jadwal 7.15 WIB, Toleransi keterlambatan adalah 15 menit.
- PAKAIAN** bebas **RAPI, BERKERAH, berSEPATU.**
- Setiap mahasiswa **TIDAK DIPERKENANKAN MENCONTEK, PLAGIAT**, dalam pengerojaan tugas dan ujian, jika terjadi maka pengerojaan tugas dan ujian akan dikurangi 20% atau Gugur.
- Setiap mahasiswa **WAJIB MENGIKUTI UJIAN** dan **TUGAS** baik tugas **MANDIRI, berKELOMPOK** atau **PRAKTIKUM.**
- Wajib untuk **BERTUTUR KATA** yang **SOPAN** dan **SANTUN di DALAM KELAS.**

Metode Penilaian



Tugas

- ❑ Tugas personal akan diberikan pada waktu perkuliahan
- ❑ Untuk pelaksanaan praktikum dilaksanakan berbarengan dengan waktu perkuliahan sesuai dengan jadwal pada lab yang digunakan.

Proyek Akhir

- Membuat aplikasi sederhana dengan fokus **Penerapan Database** ke Aplikasi untuk menyimpan transaksi
- **Tahapannya :**
 - Penentuan Studi Kasus
 - Perancangan Database beserta Relasi Tabelnya
 - Pada database terdapat beberapa SQL Langguage yang dilakukan diantaranya : CRUD, Transactions, Function, Stored Procedure & Trigger, System Catalog hingga hak akses.
 - Untuk Aplikasi boleh Web atau Desktop, fokus pada penerapan Database.
 - Pembuatan Laporan atau Dokumentasi.
- **Poin penilaian:** Aplikasi (Penerapan Database), Dokumentasi, Presentasi.

6) Kebutuhan Software

Kebutuhan Software

Browser

- Adobe flash
- Chrome
- Firefox

Localserver

- Xampp
- Laragon

Desain Tools

- Power Designer
- Sparx Enterprise Architect

Editor

- Notepad++
- Sublime Text

Database GUI

- PostgreSQL
- HeidiSQL
- SQLYog
- FlySpeed SQL

Database

- Mysql
- Oracle

7) Contact

Contact

- ❑ Bahan Kuliah : github.com/doniaft
- ❑ Email : doniaft@gmail.com
- ❑ WA/Telegram :
- ❑ Komting SMBD SI4A Romi : [0857 0681 7980](tel:085706817980)

8) Referensi

Referensi (I)

- Raghu Ramakhrisnan, Johannes Gehrke , “Database Management System” 3rd Edition, Mc Graw Hill,2003.
- Rick van der Lans, Introduction to SQL, Mastering Relational Database Language 2nd Edition, Addison-Wesley, 2000.
- Chris Bates, Web Programming: Building Internet Applications, Third Edition, John Wiley & Sons Ltd, England, 2006.
- Sebesta, R.W., Programming the World Wide Web, Addison Wesley, 2002.
- Elliot White III, Jonathan Eisenhamer, PHP 5 in Practice, Sams, 2006.
- SQL For MySQL Developers, Rick F. van der Lans, Addison Wesley, 2007
- MySQL Reference Manual, MySQL 2003
- Database Systems - A Practical Approach to Design, Implementation, and Management, Thomas Connoly and Carolyn Begg, Addison Wesley 1999