

# SISTEM MANAJEMEN BASIS DATA

03. REVIEW DDL

04. Review DML

05. Function

07. View dan User Authorisation

08. Perulangan dan Keputusan

10. Stored Procedure

Doni Abdul Fatah

[github.com/donifaft](https://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) View dan User Authorisation
- 2) Stored Procedure
- 3) Perulangan dan Keputusan
- 4) Trigger
- 5) Kontrak Perkuliahan
- 6) Kebutuhan Software
- 7) Contact
- 8) Referensi

# **View dan User Authorisation**

- 1) View
- 2) Grant
- 3) Revoke
- 4) Privelege
- 5) Adding User
- 6) Limiting User Resource

# SuperUser Vs Privileges

- User '**root**' dalam istilah keamanan komputer sering disebut sebagai '**superuser**'.
- **Superuser** user tertinggi dimana user ini dapat melihat, mengubah, bahkan menghapus seluruh database dan menjalankan perintah apapun yang terdapat dalam SQL (**CRUD**)
- **Privileges User** yang dibatasi hak aksesnya.
- Apakah user tersebut dapat membuat, mengubah dan menghapus sebuah tabel, atau user tersebut kita batasi hanya untuk melihat tabel saja (**SELECT**).

# Hak Akses

- **Hak akses** didalam Database adalah **hak** yang diberikan **kepada User** untuk dapat **mengakses data** / record tertentu.
- **Hak akses** ini jenisnya bermacam-macam bisa saja untuk memberikan **hak akses Tabel**, **hak akses Kolom** untuk dapat diakses oleh User tertentu.
- Setiap **user** dapat **dibatasi** untuk dapat **mengakses** baik itu sebuah **database** tertentu saja, **tabel** tertentu, atau bahkan hanya **kolom** tertentu.
- Kita dapat membuat user baru yang hanya bisa menjalankan perintah **SELECT** saja, dan user tersebut dibatasi untuk tidak dapat menjalankan query **DROP**.

# Users Authorisation

- Untuk proteksi data: USER ,PASSWORD, HAK AKSES
- Hak Akses meliputi:
  - Hak akses terhadap suatu **Kolom** tertentu dari suatu tabel
  - Hak akses terhadap suatu **Tabel**
  - Hak akses terhadap tabel-tabel yang ada pada suatu **Basisdata** tertentu
  - Hak akses suatu **User** terhadap seluruh basisdata yang ada di dalam server

# Bentuk otorisasi dalam basis data

- **Read Authorization** – pengguna diperbolehkan membaca data, tetapi tidak dapat memodifikasi.
- **Insert Authorization** – pengguna diperbolehkan menambah data baru, tetapi tidak dapat memodifikasi data yang sudah ada.
- **Update Authorization** – pengguna diperbolehkan memodifikasi data, tetapi tidak dapat menghapus data.
- **Delete Authorization** – pengguna diperbolehkan menghapus data.

# Bentuk otorisasi untuk modifikasi skema basis data

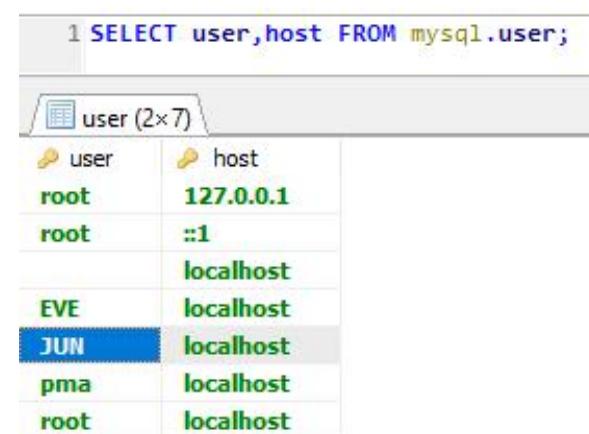
- **Index Authorization** = pengguna diperbolehkan membuat dan menghapus **index** data.
- **Authorization** = pengguna diperbolehkan **membuat relasi-relasi baru**.
- **Alteration Authorization** = pengguna diperbolehkan **menambah/menghapus atribut suatu relasi**.
- **Drop Authorization** = pengguna diperbolehkan **menghapus relasi** yang sudah ada.

# Menambah, Melihat & Menghapus USERS

- **CREATE USER 'user\_name'@'host\_name' IDENTIFIED BY password\_user**
- Contoh 1: Buatlah dua user baru, yaitu **EVE** dengan password **EVE\_PASS**, dan **JUN** dengan password **JUN\_PASS**!
- **CREATE USER**  
`'EVE'@'localhost' IDENTIFIED BY 'EVE_PASS',  
'JUN'@'localhost' IDENTIFIED BY 'JUN_PASS'`

Melihat user :

```
SELECT user,host FROM mysql.user;
```



The screenshot shows the results of a SQL query in MySQL Workbench. The query is:

```
1 SELECT user,host FROM mysql.user;
```

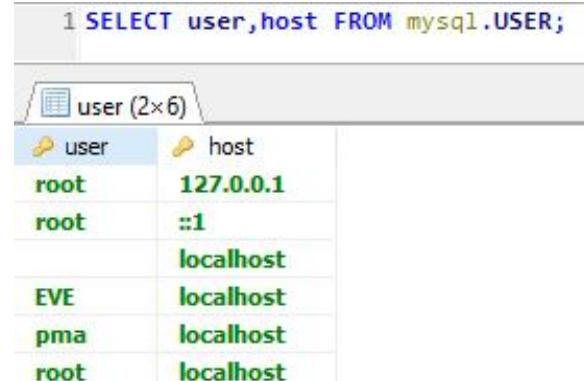
The results are displayed in a table titled "user (2x7)". The table has two columns: "user" and "host". The data is as follows:

user	host
root	127.0.0.1
root	::1
	localhost
EVE	localhost
<b>JUN</b>	<b>localhost</b>
pma	localhost
root	localhost

# Menambah, Melihat & Menghapus USERS

- DROP USER 'user\_name'@'host\_name'
- **Contoh 2:** Hapuslah user JUN!

```
DROP USER 'JUN'@'localhost'
```



The screenshot shows the results of a SQL query in MySQL Workbench. The query is:

```
1 SELECT user,host FROM mysql.USER;
```

The results are displayed in a table titled "user (2x6)".

user	host
root	127.0.0.1
root	::1
	localhost
EVE	localhost
pma	localhost
root	localhost

```
DROP USER  
'JUN'@'localhost', 'EVE'@'localho  
st'
```

# Mengubah Nama & Password USERS

- **RENAME USER** 'user\_name'@'host\_name' TO 'user\_name'@'host\_name'
- Contoh 3: **Ubahlah** nama user **EVE** dan **JUN** menjadi **EVE1** dan **JUN1**

```
RENAME USER
'EVE'@'localhost' TO 'EVE1'@'localhost',
'JUN'@'localhost' TO 'JUN1'@'localhost'
```

- **SET PASSWORD FOR** 'user\_name'@'host\_name' = **PASSWORD** ('new\_password')
- Contoh 4: **Ubahlah** password user **JUN1** menjadi **JUN1\_PASS!**

```
SET PASSWORD FOR 'JUN1'@'localhost' = PASSWORD ('JUN1_PASS')
```

# Level Hak Akses - GRANT

- Hak Akses Global (\*.\*)
- Hak Akses Level Database (nama\_database.\*)
- Hak Akses Level Tabel (nama\_database.nama\_tabel)
- Hak Akses Level Kolom (nama\_kolom)

# Hak Akses Global (\*.\*)

- Hak akses yang dapat mengakses seluruh bagian didalam suatu database
- Misalkan Tabel, Kolom, dan Data.
- Hak akses jenis ini dapat melakukan apapun CRUD didalam Database.
- Penulisan querynya biasanya (\*.\*).
- Untuk memberi akses terhadap user dilakukan di User Root.
- **Sql :**  
**GRANT hak\_akses ON \*.\* TO "nama\_user"@"lokasi\_user";**  
**GRANT SELECT ON \*.\* TO 'JUN1'@'localhost', 'EVE1'@'localhost';**

# Hak Akses : Basisdata

- **SELECT** — user berhak untuk **mengakses seluruh tabel dan view** dari suatu basisdata dengan menggunakan pernyataan **SELECT**
- **INSERT** — user berhak untuk **menambah baris pada seluruh tabel** yang ada dalam suatu basisdata dengan menggunakan pernyataan **INSERT**
- **DELETE** — user berhak untuk **menghapus baris dari seluruh tabel** yang ada dalam suatu basisdata dengan menggunakan pernyataan **DELETE**
- **UPDATE** — user berhak untuk **mengubah nilai dari seluruh tabel** yang ada dalam suatu basisdata dengan menggunakan pernyataan **UPDATE**
- **REFERENCES** — user berhak untuk membuat **Foreign Key ( FK )** yang mengacu pada tabel-tabel yang ada dalam suatu basisdata
- **CREATE** — **CREATE** — user berhak untuk **membuat tabel-tabel baru** dalam suatu basisdata dengan menggunakan pernyataan **CREATE TABLE**
- **ALTER** — user berhak untuk **mengubah tabel-tabel** pada suatu basisdata dengan menggunakan pernyataan **ALTER TABLE**

# Hak Akses : Basisdata (contd-2)

- **DROP** —user berhak untuk **menghapus seluruh tabel dan view** yang ada dalam suatu basisdata
- **INDEX** —user berhak untuk **menentukan dan menghapus indeks** dari tabel-tabel yang ada dalam suatu basisdata
- **CREATE TEMPORARY TABLES** —user berhak untuk **membuat tabel-tabel sementara** di dalam suatu basisdata
- **CREATE VIEW** — user berhak untuk **membuat view baru** dalam suatu basisdata dengan menggunakan pernyataan **CREATE VIEW**
- **SHOW VIEW** — user berhak untuk melihat **definisi view** dari **seluruh view** yang ada dalam suatu basisdata dengan menggunakan pernyataan **SHOW VIEW**
- **CREATE ROUTINE** —user berhak untuk **membuat prosedur (stored procedure) dan fungsi (stored function)** baru untuk suatu basisdata
- **ALTER ROUTINE** —user berhak untuk **mengubah dan menghapus prosedur (stored procedure) dan fungsi (stored function)** dari suatu basisdata

# Hak Akses : Basisdata (contd-3)

- ❑ **EXECUTE ROUTINE** — user berhak untuk membatalkan prosedur (**stored procedure**) dan fungsi (**stored function**) yang ada pada suatu basisdata
- ❑ **LOCK TABLES** — user berhak untuk **memblok tabel-tabel** yang ada dalam suatu basisdata
- ❑ **ALL** atau **ALL PRIVILEGES** —merupakan 'singkatan' atas **seluruh hak akses** yang telah disebutkan sebelumnya

# Hak Akses Database - nama\_database.\*

- Hanya dapat mengakses Database tertentu saja serta dapat mengakses seluruh Tabel dan Kolom didalam Database tersebut.
- Sql :

```
GRANT hak_akses ON nama_database.* TO "nama_user"@"lokasi_user";
```

```
GRANT SELECT ON tennis.* TO 'JUN1'@'localhost';
```

# Hak Akses: Basisdata (contd-4)

- **Contoh 8:** Berikanlah hak SELECT untuk JUN1 pada seluruh tabel Yang ada di dalam basisdataTENNIS!

GRANT SELECT ON TENNIS.\* TO JUN1

`GRANT SELECT ON TENNIS.* TO 'JUN1'@'localhost';`

- **Contoh 9:** Berikanlah hak CREATE, UPDATE dan REMOVE tabel-tabel dan view baru untuk JUN1 didalam basisdata TENNIS!

GRANT CREATE, ALTER, DROP, CREATE VIEW ON TENNIS.\* TO JUN1

`GRANT CREATE, ALTER, DROP, CREATE VIEW ON TENNIS.* TO 'JUN1'@'localhost'`

# Hak Akses : Tabel & Kolom

- **Select** : User berhak untuk mengakses tabel tertentu dengan menggunakan pernyataan SELECT
- **INSERT** : user berhak untuk menambah baris pada tabel tertentu dengan menggunakan pernyataan INSERT
- **DELETE** : user berhak untuk menghapus baris dari tabel tertentu dengan menggunakan pernyataan DELETE
- **UPDATE** : user berhak untuk mengubah nilai dari suatu tabel tertentu dengan menggunakan pernyataan UPDATE
- **REFERENCES** : user berhak untuk membuat Foreign Key ( FK ) yang mengacu pada suatu tabel
- **CREATE** : user berhak untuk membuat tabel dengan suatu nama tertentu
- **ALTER** : user berhak untuk mengubah tabel dengan menggunakan pernyataan ALTER TABLE
- **INDEX** : user berhak untuk menentukan indeks pada satu tabel
- **DROP** : user berhak untuk menghapus tabel
- **ALL** atau **ALL PRIVILEGES** : merupakan 'singkatan' atas seluruh hak akses yang telah disebutkan di atas

# Hak Akses :Tabel & Kolom (contd-2)

- **Contoh 5:** Berikanlah hak SELECT untuk JUN1 pada tabel PLAYERS!

**GRANT SELECT ON PLAYERS TO JUN1**

```
GRANT SELECT ON tennis.players TO 'JUN1'@'localhost';
```

- **Contoh 6:** Berikanlah hak SELECT untuk user baru BOB pada tabel PLAYERS!

**GRANT SELECT ON PLAYERS TO 'BOB'@'localhost' IDENTIFIED BY 'BOB\_PASS'**

```
GRANT SELECT ON PLAYERS TO 'BOB'@'localhost' IDENTIFIED BY 'BOB_PASS'
```

- **Contoh 7:** Berikanlah hak INSERT dan UPDATE untuk EVE1 dan JUN1 pada seluruh kolom dari tabel TEAMS!

**GRANT INSERT, UPDATE ON TEAMS TO EVE1, JUN1**

```
GRANT insert,update,ALTER ON tennis.teams TO  
'JUN1'@'localhost';
```

# Hak Akses Tabel - nama\_database.nama\_tabel

- User memiliki hak akses pada sebuah tabel beserta Kolomnya yang berada pada sebuah database.
- Hak akses yang dimiliki user hanya terbatas pada level sebuah tabel saja.
- Sql :

```
GRANT hak_akses ON nama_database . nama_tabel TO  
"nama_user"@"lokasi_user";
```

```
GRANT SELECT ON tennis.players TO 'JUN1'@'localhost';
```

# Hak Akses Level Kolom - nama\_kolom

- ❑ Hak akses ini adalah hak akses paling kecil yang dapat diberikan kepada sebuah user.
- ❑ Dengan hak akses level kolom, user hanya memiliki hak akses untuk beberapa kolom pada sebuah tabel.
- ❑ Level paling akhir ini kita membatasi hak akses user hanya untuk kolom tertentu saja.
- ❑ Penulisan kolom yang diperbolehkan diletakkan di dalam tanda kurung.
- ❑ Sql :

```
GRANT hak_akses (nama_kolom) ON nama_database.nama_tabel  
TO "nama_user"@"lokasi_user";
```

```
GRANT SELECT (nama,umur) ON tennis.players TO 'JUN1'@'localhost';
```

# Pemberian Akses DBA ke User account terhadap database:

- Membatasi User akses data **table** baik untuk **melihat struktur table, melihat data** maupun melakukan **operasi manipulasi data** seperti **Insert, Update, Delete** data
- Membatasi user akses **view** database baik melihat, maupun merubah struktur **view**
- Membatasi user akses **strored procedure** baik **execute** dan merubah struktur **SQL** didalam **stored procedure** tersebut.
- Membatasi Host akses yang digunakan user baik local host(**127.0.0.1**), user akses dalam **jaringan LAN** dan Remot IP Public.
- Memberikan **timer user akses database** pada saat **jam kerja** saja misalnya diluar jam kerja user tidak dapat melakukan akses database.
- Memberikan **otoritas user** untuk **create tabel, view function, stored procedure , trigger**

# Contoh Perintah SQL

- ❑ **GRANT** : memberikan wewenang kepada pemakai
- ❑ Syntax : GRANT ON TO
- ❑ Contoh :
  - ❑ GRANT SELECT ON S TO BUDI
  - ❑ GRANT SELECT,UPDATE (STATUS,KOTA) ON S TO SULIS,WENDI
- ❑ **REVOKE** : mencabut wewenang yang dimiliki oleh pemakai
- ❑ Syntax : REVOKE ON FROM
- ❑ Contoh :
  - ❑ REVOKE SELECT ON S TO BUDI
  - ❑ REVOKE SELECT,UPDATE (STATUS,KOTA) ON S TO SULIS,WENDI
- ❑ **Priviledge list** : READ, INSERT, DROP, DELETE, INEX, ALTERATION, RESOURCE

# Hak Akses:User

- Contoh 10: Berikanlah hak CREATE,ALTER,dan DROP untuk EVE1 pada seluruh tabel dari seluruh basisdata!

GRANT CREATE, ALTER, DROP ON \*.\* TO EVE1

**GRANT CREATE, ALTER, DROP ON \*.\* TO EVE1 @'localhost'**

- Contoh 11: Berikanlah hak kepada JUN1 untuk membuat user baru!

GRANT CREATE USER ON \*.\* TO JUN1

**GRANT CREATE USER ON \*.\* TO JUN1@localhost**

# Meneruskan Hak Akses: WITH GRANTOPTION

- Contoh 12: Berikanlah hak REFERENCES untuk JUN1 pada tabel TEAMS dan berikanlah ijin padanya untuk meneruskan hak tersebut kepada user lain!

GRANT REFERENCES ON TEAMS TO JUN1 WITH GRANT OPTION

**GRANT REFERENCES ON TEAMS TO JUN1@localhost WITH GRANT OPTION**

- Dengan adanya klausa WITH GRANT OPTION, maka JUN1 dapat meneruskan hak akses REFERENCES tersebut kepada EVE1

GRANT REFERENCES ON TEAMS TO EVE1

**GRANT REFERENCES ON TEAMS TO EVE1@localhost**

# Membatalkan Hak Akses

- Contoh 13: Batalkan hak SELECT untuk JUN1 pada tabel PLAYERS!

REVOKE SELECT ON PLAYERS FROM JUN1

`REVOKE SELECT ON PLAYERS FROM JUN1@localhost`

- Contoh 14: Batalkan hak INSERT dan SELECT untuk EVE1 pada tabel TEAMS!

REVOKE INSERT, SELECT ON TEAMS FROM EVE1

`REVOKE INSERT, SELECT ON teams FROM EVE1@localhost`

# Otorisasi dan View

- **View** adalah **objek basis data** yang berisi perintah **query** ke basis data
- Setiap kali sebuah **view diaktifkan**, pemakai akan selalu **melihat hasil querynya**.
- Berbeda dengan tabel, **data** yang ditampilkan **didalam view tidak bisa di ubah**.
- **View** menyediakan mekanisme **pengamanan yang fleksibel** namun **kuat** dengan cara **menyembunyikan sebagian basis data** dari user lain
- **User** dapat **diberikan otorisasi pada View**, tanpa harus diberikan otorisasi terhadap relasi yang digunakan di dalam **definisi view**.
- **View** dapat **meningkatkan keamanan data** dengan **mengizinkan user** untuk **hanya dapat mengakses data** sesuai dengan **pekerjaannya masing-masing**.
- **Kombinasi** antara **level keamanan** ditingkat **relasional** dengan **level keamanan** **dingkat view** dapat digunakan untuk **membatasi hak akses user**, sehingga mereka **hanya mengakses data** sesuai dengan **kebutuhan** saja.

# VIEWS

```
CREATE VIEW view_name (column_name) AS  
[SELECT BLOCK]  
[WITH CHECK OPTION]
```

- Merupakan '*derivedtables*' ⇒ harus didefinisikan dalam bentuk query pada tabel atau view yang lain
- Merupakan '*virtual tables*' ⇒ tidak akan dievaluasi, kecuali jika mereka digunakan dalam query lain
- reusable
- Dapat digunakan untuk membuat query yang sulit (atau bahkan tidak mungkin) untuk dilakukan dalam suatu kondisi tertentu perintah INSERT,
- UPDATE, atau DELETE dapat dilakukan terhadap data yang ada di dalam tabel basis melalui view tabel basis tersebut

# VIEWS

```
CREATE VIEW view_name (column_name) AS  
[SELECT BLOCK]  
[WITH CHECK OPTION]
```

- ❑ Tabel View bisa berasal dari tabel lain, atau gabungan dari beberapa tabel.
- ❑ Tujuan :
  - ❑ kenyamanan (mempermudah penulisan query),
  - ❑ Keamanan (menyembunyikan beberapa kolom yang bersifat rahasia),
  - ❑ beberapa kasus bisa digunakan untuk mempercepat proses menampilkan data (terutama jika kita akan menjalankan query tersebut secara berulang)

# Create VIEWS

- **Contoh 1:** Buatlah **view** untuk membuat daftar seluruh nama kota yang ada dalam tabel **PLAYERS**!
- **CREATE VIEW TOWNS (TOWN)  
AS SELECT DISTINCT TOWN  
FROM PLAYERS**
- Sintaks Alternatif:
- **CREATE VIEW TOWNS AS  
SELECT DISTINCT TOWN FROM  
PLAYERS**
- Query SELECT:
- **SELECT \* FROM TOWNS**

1	CREATE VIEW TOWNS AS SELECT DISTINCT TOWN FROM PLAYERS
/	players (1x6)
	TOWN
	Stratford
	Inglewood
	Eltham
	Midhurst
	Douglas
	Plymouth

# Create VIEWS (contd 2)

- Contoh 2: Buatlah view untuk membuat daftar nomor pemain dan nomor liga dari seluruh pemain yang memiliki nomor liga!
- **CREATE VIEW CPLAYERS AS SELECT PLAYERN0, LEAGUENO FROM PLAYERS WHERE LEAGUENO IS NOT NULL**
- Query SELECT:
- **SELECT \* FROM CPLAYERS**
- Contoh 3: Dapatkan seluruh informasi mengenai pemain yang memiliki nomor liga yang nomor pemainnya adalah antara 6 dan 44!
- **SELECT \* FROM CPLAYERS WHERE PLAYERN0 BETWEEN 6 AND 44**
- Atau sama dengan :  
**SELECT \* FROM CPLAYERS WHERE PLAYERN0 >= 6 AND PLAYERN0 <=44;**

```
1 CREATE VIEW CPLAYERS AS SELECT PLAYERN0, LEAGUENO FROM PLAYERS WHERE LEAGUENO IS NOT NULL  
2 SELECT * FROM CPLAYERS  
3
```

CPLAYERS (2x10)	
PLAYERN0	LEAGUENO
2	2411
6	8467
8	2983
27	2513
44	1124
57	6409
83	1608
100	6524
104	7060
112	1319

```
1 SELECT * FROM CPLAYERS WHERE PLAYERN0 BETWEEN 6 AND 44
```

CPLAYERS (2x4)	
PLAYERN0	LEAGUENO
6	8467
8	2983
27	2513
44	1124

# Create VIEWS (contd 3)

- Contoh 4: Buatlah **view** untuk membuat **daftar seluruh pemain** yang **memiliki nomor liga** dengan **nomor pemain antara 6 dan 27!**
- **CREATE VIEW SEVERAL AS  
SELECT \* FROM CPLAYERS  
WHERE PLAYERNO BETWEEN  
6 AND 27**
- Untuk kebalikannya :
- **CREATE VIEW SEVERAL2 AS  
SELECT \* FROM CPLAYERS  
WHERE PLAYERNO not  
BETWEEN 6 AND 27**

1 SELECT \* FROM CPLAYERS

CPLAYERS (2x10)	
PLAYERNO	LEAGUENO
2	2411
6	8467
8	2983
27	2513
44	1124
57	6409
83	1608
100	6524
104	7060
112	1319

1 SELECT \* FROM SEVERAL

SEVERAL (2x3)	
PLAYERNO	LEAGUENO
6	8467
8	2983
27	2513

# Create VIEWS (contd 4)

- Contoh 5: Tentukan **(nilai) rata-rata** dari **keseluruhan penalti** yang pernah dilakukan oleh **pemain!**
- Solusi berikut **tidak mungkin** dapat dilakukan:
- **SELECT PLAYERNO, AVG(SUM(AMOUNT)) FROM PENALTIES GROUP BY PLAYERNO;**
- Solusiyang dapat dilakukan dengan menggunakan VIEW:
- **CREATE VIEW SUM\_PENALTIES (PLAYERNO, SUM\_AMOUNT) AS SELECT PLAYERNO, SUM(AMOUNT) FROM PENALTIES GROUP BY PLAYERNO;**
- **SELECT AVG(SUM\_AMOUNT) FROM SUM\_PENALTIES;**

```
1 CREATE VIEW SUM_PENALTIES (PLAYERNO, SUM_AMOUNT) AS  
2 SELECT PLAYERNO, SUM(AMOUNT) FROM PENALTIES GROUP BY PLAYERNO;
```

# Create VIEWS (contd 5)

- Sintaks Alternatif untuk membuat VIEW:
- **CREATE VIEW**  
**SUM\_PENALTIES AS**  
**SELECT PLAYERNO,**  
**SUM(AMOUNT) AS**  
**SUM\_AMOUNT FROM**  
**PENALTIES GROUP BY**  
**PLAYERNO;**

```
1 SELECT AVG(SUM_AMOUNT)
2 FROM SUM_PENALTIES;|
```

Hasil #1 (1x1)

Avg(SUM_AMOUNT)
96,000000

# Create VIEWS (contd 5)

- Contoh 6: Buatlah **view** untuk **membuat daftar** urutan digit dari 0 sampai 9!

- **CREATE VIEW DIGITS AS**

```
SELECT 0 DIGIT UNION
```

```
SELECT 1 UNION
```

```
SELECT 2 UNION
```

```
SELECT 3 UNION
```

```
SELECT 4 UNION
```

```
SELECT 5 UNION
```

```
SELECT 6 UNION
```

```
SELECT 7 UNION
```

```
SELECT 8 UNION
```

```
SELECT 9
```

- **SELECT \* FROM DIGITS**

```
1 SELECT * FROM DIGITS
```

Hasil #1 (1×10)

DIGIT
0
1
2
3
4
5
6
7
8
9

# Create VIEWS (contd 6)

- Contoh 7 : Misalkan kita ingin menampilkan nama pemain yang berjenis kelamin female.

```
SELECT PLAYERO NO, NAME, INITIALS , sex, TOWN FROM  
players WHERE sex='F'
```

```
1 SELECT PLAYERO NO, NAME, INITIALS , sex, TOWN FROM players WHERE sex='F'  
2 |
```

players (5x5)				
PLAYERO NO	NAME	INITIALS	sex	TOWN
8	Newcastle	B	F	Inglewood
27	Collins	DD	F	Eltham
28	Collins	C	F	Midhurst
104	Moorman	D	F	Eltham
112	Bailey	IP	F	Plymouth

Bagaimana jika query tersebut akan dijalankan setiap beberapa detik (diakses dari website yang sibuk)???

**Solusinya? view**

# Create VIEWS (contd 7)

- Contoh 8 : Solusinya dengan membuat View untuk menampilkan nama pemain yang berjenis kelamin female. Sehingga beban server berkurang dan VIEW dapat menyembunyikan beberapa kolom dari tabel players.

```
CREATE VIEW sex_female AS SELECT  
PLAYERNO, NAME, INITIALS, sex,  
TOWN FROM players WHERE sex='F'
```

```
SELECT * FROM sex_female
```

```
SELECT name FROM sex_female  
WHERE playerno='8'
```

SELECT \* from sex\_female

sex_female (5x5)				
PLAYERNO	NAME	INITIALS	sex	TOWN
8	Newcastle	B	F	Inglewood
27	Collins	DD	F	Eltham
28	Collins	C	F	Midhurst
104	Moorman	D	F	Eltham
112	Bailey	IP	F	Plymouth

```
1 SELECT name FROM sex_female WHERE playerno='8'
```

sex\_female (1x1)

NAME
Newcastle

# Create VIEWS (contd 8)

- Bagaimana jika tabel utama di update?

```
1 SELECT * from players
```

players (12x14)											
PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1948-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411
6	Parmenter	R	1964-06-25	M	1.977	Haseltine Lane	80	1234KK	Stratford	070-476537	8467
7	Wise	GWS	1963-05-11	M	1.981	Edgecombe Way	39	9758VB	Stratford	070-347689	(NULL)
8	Newcastle	B	1962-07-08	F	1.980	Station Road	4	6584WO	Inglewood	070-458458	2983
27	Collins	DD	1964-12-28	F	1.983	Long Drive	804	8457DK	Eltham	079-234857	2513
28	Collins	C	1963-06-22	F	1.983	Old Main Road	10	1294QK	Midhurst	010-659599	(NULL)
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
44	Baker	E	1963-01-09	M	1.980	Lewis Street	23	4444LJ	Inglewood	070-368753	1124
57	Brown	M	1971-08-17	M	1.985	Edgecombe Way	16	4377CB	Stratford	070-473458	6409
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608
95	Miller	P	1963-05-14	M	1.972	High Street	33A	5746OP	Douglas	070-867564	(NULL)
100	Parmenter	P	1963-02-28	M	1.979	Haseltine Lane	80	6494SG	Stratford	070-494593	6524
104	Moorman	D	1970-05-10	F	1.984	Stout Street	65	9437AO	Eltham	079-987571	7060
112	Bailey	IP	1963-10-01	F	1.984	Vixen Road	8	6392LK	Plymouth	010-548745	1319

# Create VIEWS (contd 9)

**INSERT INTO PLAYERS VALUES**

```
(3, 'Dian', 'Di', '1999-05-20', 'F', 2000, 'Surabaya', '11', '60208', 'Surabaya', '0856-4868-8777', '12'),
(4, 'Diana', 'Da', '1999-06-21', 'F', 2000,
'Bojonegoro', '12', '62115', 'Bojonegoro',
'0800-0000-1111', '13'),
(5, 'Dinda', 'Dn', '1999-07-22', 'F', 2000,
'Semarang', '13', '63115',
'Semarang', '0811-1111-0000', '14');
```

1 SELECT * FROM players												
players (12x17)												
PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO	
2	Everett	R	1948-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411	
3	Dian	Di	1999-05-20	F	2.000	Surabaya	11	60208	Surabaya	0856-4868-	12	
4	Diana	Da	1999-06-21	F	2.000	Bojonegoro	12	62115	Bojonegoro	0800-0000-	13	
5	Dinda	Dn	1999-07-22	F	2.000	Semarang	13	63115	Semarang	0811-1111-	14	
6	Parmenter	R	1964-06-25	M	1.977	Haselton Lane	80	1234K	Stratford	070-476537	8467	
7	Wise	GWS	1963-05-11	M	1.981	Edgecombe Way	39	9758VB	Stratford	070-347689	(NULL)	
8	Newcastle	B	1962-07-08	F	1.980	Station Road	4	6584WQ	Inglewood	070-458458	2983	
27	Collins	DD	1964-12-28	F	1.983	Long Drive	804	8457OK	Eltham	079-234857	2513	
28	Collins	C	1963-06-22	F	1.983	Old Main Road	10	1294K	Midhurst	010-658599	(NULL)	
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	76	9629CD	Stratford	070-393495	(NULL)	
44	Baker	E	1963-01-09	M	1.980	Lewis Street	23	4441J	Inglewood	070-368753	1124	
57	Brown	M	1971-08-17	M	1.985	Edgecombe Way	16	4377GB	Stratford	070-473458	6409	
83	Hope	PK	1956-11-11	M	1.982	Magnolene Road	16A	1812JP	Stratford	070-353548	1608	
95	Miller	P	1963-05-14	M	1.972	High Street	33A	5746GP	Douglas	070-867564	(NULL)	
100	Parmenter	P	1963-02-28	M	1.979	Haselton Lane	80	6494SG	Stratford	070-494593	6524	
104	Moorman	D	1970-05-10	F	1.984	Stout Street	65	9437AQ	Eltham	079-987571	7060	
112	Bailey	IP	1963-10-01	F	1.984	Viven Road	8	6392LK	Plymouth	010-548745	1319	

1 SELECT * FROM sex_female				
sex_female (5x8)				
PLAYERNO	NAME	INITIALS	sex	TOWN
3	Dian	Di	F	Surabaya
4	Diana	Da	F	Bojonegoro
5	Dinda	Dn	F	Semarang
8	Newcastle	B	F	Inglewood
27	Collins	DD	F	Eltham
28	Collins	C	F	Midhurst
104	Moorman	D	F	Eltham
112	Bailey	IP	F	Plymouth

Terlihat bahwa VIEW juga otomatis diupdate.

# Create VIEWS (contd 10)

- Apakah kita bisa menambahkan data ke dalam VIEW? Nama Viewnya : sex\_female

```
INSERT INTO sex_female VALUES  
(200, 'Sisi',  
'Ss', 'F', 'Madura');
```

Ternyata ketika kita update pada VIEW. Pada tabel asal juga mengalami Update meskipun akan terdapat nilai **NULL** di dalam tabel asli

```
1 SELECT * FROM sex_female
```

sex\_female (5x9)

PLAYERNO	NAME	INITIALS	sex	TOWN
3	Dian	Di	F	Surabaya
4	Diana	Da	F	Bojonegoro
5	Dinda	Dn	F	Semarang
8	Newcastle	B	F	Inglewood
27	Collins	DD	F	Eltham
28	Collins	C	F	Midhurst
104	Moorman	D	F	Eltham
112	Bailey	IP	F	Plymouth
200	Sisi	Ss	F	Madura

```
1 SELECT * FROM players
```

players (12x18)

PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1948-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411
3	Dian	Di	1999-05-20	F	2.000	Surabaya	11	60208	Surabaya	0856-4868-	12
4	Diana	Da	1999-06-21	F	2.000	Bojonegoro	12	62115	Bojonegoro	0800-0000-	13
5	Dinda	Dn	1999-07-22	F	2.000	Semarang	13	63115	Semarang	0811-1111-	14
6	Parmenter	R	1964-06-25	M	1.977	Haseline Lane	80	1234KK	Stratford	070-476537	8467
7	Wise	GWS	1963-05-11	M	1.981	Edgecombe Way	39	9758VB	Stratford	070-347689	(NULL)
8	Newcastle	B	1962-07-08	F	1.980	Station Road	4	6584WO	Inglewood	070-458458	2983
27	Collins	DD	1964-12-28	F	1.983	Long Drive	804	8457DK	Eltham	079-234857	2513
28	Collins	C	1963-06-22	F	1.983	Old Main Road	10	1294QK	Midhurst	010-659599	(NULL)
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
44	Baker	E	1963-01-09	M	1.980	Lewis Street	23	4444LJ	Inglewood	070-368753	1124
57	Brown	M	1971-08-17	M	1.985	Edgecombe Way	15	4377CB	Stratford	070-473458	6409
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608
95	Miller	P	1963-05-14	M	1.972	High Street	33A	5746OP	Douglas	070-867564	(NULL)
100	Parmenter	P	1963-02-28	M	1.979	Haseline Lane	80	6494SG	Stratford	070-494593	6524
104	Moorman	D	1970-05-10	F	1.984	Stout Street	65	9437AO	Eltham	079-987571	7060
112	Bailey	IP	1963-10-01	F	1.984	Vixen Road	8	6392LK	Plymouth	010-548745	1319
200	Sisi	Ss	(NULL)	F	0	(NULL)	(NULL)	(NULL)	Madura	(NULL)	(NULL)

# Update VIEWS -- WITH CHECK OPTION

- Contoh 9 : Buatlah view untuk membuat daftar pemain yang lahir sebelum tahun 1960!
- **CREATE VIEW VETERANS AS SELECT \* FROM PLAYERS WHERE BIRTH\_DATE < '1960-01-01';**
- **SELECT \* FROM VETERANS;**

The screenshot shows a MySQL Workbench environment. On the left, a query editor window displays the SQL command: `1 SELECT * FROM VETERANS;`. To the right of the editor is a results grid titled "VETERANS (12x3)". The grid contains 12 rows of player data. The columns are labeled: PLAYENO, NAME, INITIALS, BIRTH\_DATE, SEX, JOINED, STREET, HOUSENO, POSTCODE, TOWN, PHONENO, and LEAGUENO. The data includes players like Everett (BIRTH\_DATE: 1948-09-01), Bishop (BIRTH\_DATE: 1956-10-29), and Hope (BIRTH\_DATE: 1956-11-11). A legend on the right side of the results grid provides icons for various database operations: DELAY, DELETE, DESC, DETEF, and DIREC.

PLAYENO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1948-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608

# Update VIEWS -- WITH CHECK OPTION (Contd-2)

- **SELECT \* FROM VETERANS;**

1 `SELECT * FROM VETERANS;`

PLAYERO NO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1948-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608

- Kemudian lakukan perubahan tanggal lahir dari pemain nomor 2 (dari 1 September 1948 menjadi 1 September 1970)

`UPDATE VETERANS`

`SET BIRTH_DATE = '1970-09-01'`

`WHERE PLAYERO NO = 2;`

`SELECT * FROM VETERANS;`

PLAYERO NO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608

# Update VIEWS -- WITH CHECK OPTION (contd-3)

- Penggunaan WITH CHECK OPTION pada VIEW adalah melakukan pengecekan validitas perintah UPDATE, INSERT dan DELETE:
  - Perintah UPDATE adalah benar jika seluruh baris yang di-UPDATE tetap menjadi isi ('virtual') tabel VIEW
  - Perintah INSERT adalah benar jika baris baru yang di INSERT merupakan isi ('virtual') tabel VIEW
  - Perintah DELETE adalah benar jika baris baru yang di DELETE merupakan isi ('virtual') tabel VIEW
- Maka, view pada **Contoh 9** diubah menjadi seperti berikut:

```
CREATE OR REPLACE VIEW VETERANS AS
SELECT * FROM PLAYERS
WHERE BIRTH_DATE < '1960-01-01'
WITH CHECK OPTION
```

# Delete VIEWS

- Contoh 9: Hapuslah view CPLAYERS!

**DROP VIEW CPLAYERS;**

- Contoh 10: Hapuslahview VETERANS!

**DROP VIEW VETERANS**

```
1 SELECT * from CPLAYERS;
2 SELECT * from VETERANS
3
4 DROP VIEW VETERANS
5 DROP VIEW CPLAYERS;
```

CPLAYERS (2×13)		VETERANS (12×2)	
PLAYERNO	LEAGUENO	PLAYERNO	LEAGUENO
2	2411		
3	12		
4	13		
5	14		
6	8467		
8	2983		
27	2513		
44	1124		
57	6409		
83	1608		
100	6524		
104	7060		
112	1319		

# **Stored Procedure**

- 1) Function Lanjutan
- 2) Stored Procedure
- 3) Trigger

# Function Lanjutan

- ❑ Stored function merupakan sekumpulan perintah SQL yang disusun dalam sebuah fungsi yang memiliki nama, kegunaan, dan pembalian nilai (return value)
- ❑ Keuntungan dengan stored function yakni kita dapat membuat fungsi yang tidak tersedia.
- ❑ Aturan pembuatan stored function mirip dengan stored procedure. Perbedaanya adalah dalam function adanya nilai RETURN.
- ❑ Sama seperti halnya prosedur, dalam tubuh fungsi dapat berisi statement seperti deklarasi variabel, perintah pemilihan, dan perulangan

# Function Lanjutan

- Stored Function hampir sama seperti stored procedure, yaitu mempunyai sekumpulan statemen sql dan statemen procedural yang tersimpan pada database MySQL dan dapat dipanggil dari aplikasi maupun database MySQL.
- Poin :
  - Terdapat statemen sql dan procedural language
  - Tersimpan pada database server (MySQL)
  - Dapat dipanggil dari program aplikasi dan Database server (MySQL)

Sintaks :

```
CREATE FUNCTION <namafungsi>(IN/OUT/INOUT
parameter TYPE)
RETURNS <tipedata>
BEGIN
    <statement>
END
```

# Function Lanjutan - Sintak

```
CREATE FUNCTION sp_name ([func_parameter[, ...]])  
    RETURNS type  
    [characteristic ...] routine_body  
proc_parameter:  
    [ IN | OUT | INOUT ] param_name type  
func_parameter:  
    param_name type  
type:  
    Any valid MySQL data type  
characteristic:  
    LANGUAGE SQL  
    | [NOT] DETERMINISTIC  
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIESTEXT DATA }  
    | SQL SECURITY { DEFINER | INVOKER }  
    | COMMENT 'string'  
routine_body:  
    Valid SQL procedure statement or statements
```

# Keterangan Sintak Function

- ❑ **sp\_name:** Nama *routine* yang akan dibuat
- ❑ **proc\_parameter:** Spesifikasi parameter sebagai IN, OUT, atau INOUT valid hanya untuk PROCEDURE. (parameter FUNCTION selalu sebagai parameter IN)
- ❑ **returns:** Klausa RETURNS dispesifikan hanya untuk suatu FUNCTION. Klausa ini digunakan untuk mengembalikan tipe fungsi, dan *routine\_body* harus berisi suatu statemen nilai RETURN.
- ❑ **comment:** Klausa COMMENT adalah suatu ekstensi MySQL, dan mungkin digunakan untuk mendeskripsikan *stored procedure*. Informasi ini ditampilkan dengan statemen SHOW CREATE PROCEDURE dan SHOW CREATE FUNCTION.

# Keterangan Sintak Function MYSQL

- **DELIMITER** adalah untuk memberi tahu kepada mysql soal delimiter yang digunakan, secara default menggunakan ; jadi bila ada tanda ; mysql akan mengartikan akhir dari statement, pada contoh di atas delimiter yang digunakan // jadi akhir statementnya adalah //
- **CREATE FUNCTION** adalah header untuk membuat sebuah function.
- **RETURNS** adalah untuk menentukan tipe data yang di return-kan oleh sebuah function.

# Keterangan Sintak Function MYSQL

- **DETERMINISTIC/ NOT DETERMINISTIC** adalah untuk menentukan yang bisa menggunakan function ini adalah user pembuatnya saja (deterministic) atau user siapa saja (not deterministic).
- Untuk penulisan **DETERMINISTIC** bisa ditulis secara implisit dengan memberikan setting global pada mysql dan secara default benilai **NOT DETERMINISTIC**.
- **BEGIN END** adalah body dari function jadi semua SQL nya di tulis disini.

# Function Lanjutan - Contoh

**Contoh 1 :** Buatlah Function dengan Case untuk menentukan jumlah diskon jika jumlah nilainya lebih dari 100 maka diskonnya 10, jika jumlah nilainya dibawah 100 s.d 50 maka diskonya 5, dan jika jumlah nilanya dibawah 50 hingga 20 maka diskonnya 3 dan jika tidak maka tidak ada diskon.

```
DELIMITER //
CREATE FUNCTION getDiskon(jumlah INT)
RETURNS int(11)
BEGIN
    DECLARE diskon INT; CASE
        WHEN (jumlah >= 100) THEN SET diskon = 10;
        WHEN (jumlah >= 50 AND jumlah < 100) THEN SET diskon = 5;
        WHEN (jumlah >= 20 AND jumlah < 50) THEN SET diskon = 3;
        ELSE SET diskon = 0; END CASE;
    RETURN diskon;
END//
```

Perintah Untuk memanggilnya :

```
SELECT getdiskon('20');
```



# Function Lanjutan - Contoh

Contoh 2 : Buatlah Function untuk menentukan volume segitiga

Delimiter //

```
create function volume (panjang int, lebar int,  
tinggi int) returns int  
deterministic  
begin  
declare volum INT;  
set volum = panjang * lebar * tinggi;  
return volum;  
END//
```

Contoh Perintah Untuk memanggilnya :

```
Select volume (5,3,7);
```

1 Select volume (5,3,7);
Hasil #1 (1x1)
volume (5,3,7)
105

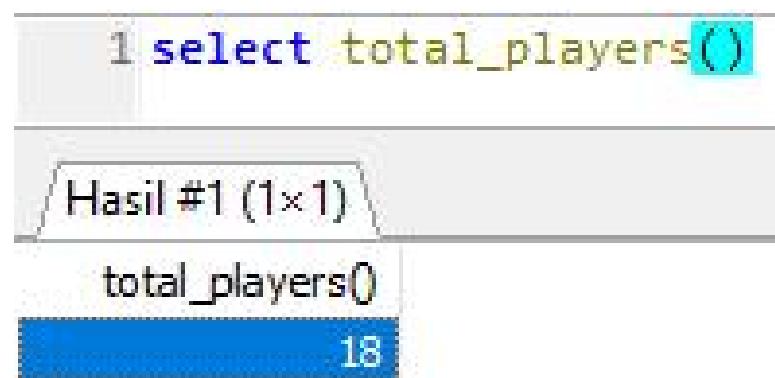
# Function Lanjutan - Contoh

**Contoh 3:** Buatlah Function untuk menghitung jumlah pemain yang tercatat di dalam tabel PLAYERS!

```
Delimiter //
CREATE FUNCTION total_players()
RETURNS Int
BEGIN
RETURN (SELECT COUNT(*) FROM PLAYERS);
end
```

Perintah Untuk memanggilnya :

```
select total_players()
```



The screenshot shows a MySQL command-line interface. A query is being run: "1 select total\_players()". The result set is labeled "Hasil #1 (1x1)" and contains a single row with the value "total\_players0". The number "18" is visible at the bottom right of the interface.

total_players()
total_players0

18

# Function Lanjutan - Contoh

- **Contoh 4:** Buatlah Function untuk menghitung Ada berapa nama kota yang tercatat di dalam kolom TOWN dalam tabel PLAYERS, dengan menghilangkan data yang duplikat?

Delimiter //

```
CREATE FUNCTION number_players()
RETURNS Int
BEGIN
RETURN (SELECT COUNT(DISTINCT (TOWN)) FROM PLAYERS);
end
```

Perintah Untuk memanggilnya :

```
select number_players()
```

number_players()
10

# Stored Procedures

- Merupakan sekumpulan sintaks SQL yang tersimpan pada server
- Memiliki beberapa keunggulan
  - Karena sintaks sql pada stored procedure tersimpan pada server maka pemanggilan lebih cepat.
  - Reuseable artinya cukup ditulis sekali dapat digunakan berkali-kali
  - Meningkatkan keamanan

# Stored Procedure vs Trigger

## □ Persamaan:

- Obyeknya dapat berupa TABLE, VIEW, dll
- Tersimpan di dalam System Catalog basisdata
- Terdiri dari pernyataan SQL yang deklaratif (seperti CREATE, UPDATE, dan SELECT) dan prosedural (seperti IF-THEN-ELSE dan WHILE-DO)

## □ Perbedaan:

- Stored procedure diaktifkan sebagai suatu pernyataan oleh SQL editor, program, atau oleh stored procedure atau trigger lain
- Trigger diaktifkan hanya oleh RDBMS dalam suatu kondisi tertentu (ketika pernyataan INSERT, UPDATE, DELETE dieksekusi)

# Komponen Stored Procedure

- Nama Stored Procedure
- Parameter
- Body
- Sintaknya :

```
Create procedure nama_prosedur(param1,  
                                param2,...,paramn)  
Begin  
    <sintaks SQL>  
end
```

## Contd. Stored Procedure

```
CREATE PROCEDURE DELETE_MHS  
    (IN P_MHSID INTEGER)  
BEGIN  
    DELETE FROM MAHASISWA  
    WHERE ID_MHS = P_MHSID;  
END
```

Nama Procedure

Parameter

Body

# Stored Procedure - Body

- Berisi semua statement yang akan dieksekusi.
- Diawali keyword BEGIN dan diakhiri END
- Statement SQL dapat berupa : DDL, DML, DCL
- Procedural Language : IF-THEN-ELSE, WHILE-DO
- Dapat mendeklarasikan variabel (local variabel)

# Stored Procedure

## □ Create Stored Procedure:

```
<create procedure statement> ::=  
CREATE PROCEDURE <procedure name> ( [ <parameter list> ] )  
<routine body>  
<parameter list> ::=  
<parameter specification> [ , <parameter specification> ]...  
<parameter specification> ::=  
[ IN | OUT | INOUT ] <parameter> <data type>  
<routine body> ::= <begin-end block>  
<begin-end block> ::=  
[ <label> : ] BEGIN <statement list> END [ <label> ]  
<statement list> ::= { <body statement> ; }...  
<statement in body> ::=  
<declarative statement> | <procedural statement>
```

# Stored Procedure

## □ Aktivasi /pemanggilan Stored Procedure :

```
<call statement> ::=  
CALL [ <database name> . ] <stored procedure  
name>  
( [ <scalar expression> [ , <scalar expression>  
]... ] )
```

## □ Menghapus Stored Procedure:

```
<drop procedure statement> ::=  
DROP PROCEDURE [ IF EXISTS ]  
[ <database name> . ] <procedure NAME>
```

# Parameter Stored Procedure

## □ IN

- Parameter yang merupakan mode default ini mengindikasikan bahwa sebuah parameter dapat di-pass ke dalam stored procedure tetapi nilainya tidak dapat diubah (dari dalam stored procedure)/
- untuk inputan parameter yang akan disimpan dalam stored procedure

## □ OUT

- stored procedure dapat mengubah parameter dan mengirimkan kembali ke program pemanggil./
- sebagai output parameter yang diperoleh dalam stored procedure

## □ INOUT

- Kombinasi dari mode IN dan OUT.
- kita bisa mengirimkan parameter kedalam stored procedure dan mendapatkan nilai kembalian yang baru dari stored procedure yang didefinisikan./
- sebagai parameter yang dapat digunakan sebagai input maupun output

# Variabel Stored Procedure

- ❑ Dideklarasikan dengan keyword “DECLARE” kemudian diikuti dengan nama variabel dan tipe data.
- ❑ Sintaks:

**DECLARE** namavariabel **TYPE** **DEFAULT** nilai

# Pemilihan Stored Procedure

- ❑ Perintah pemilihan ini berupa statement-statement yang akan mengerjakan instruksi jika kondisi benar/terpenuhi
- ❑ Sintaks :

```
IF [val] THEN  
[result1]  
END IF ;
```

```
IF [val] THEN  
[result1]  
ELSE  
[result2]  
END IF ;
```

# Perulangan Stored Procedure

- Perintah perulangan dengan menggunakan statement LOOP, WHILE, dan REPEAT.
- Penggunaan statement LOOP diawali dengan menentukan nama perulangan : LOOP dan diakhiri dengan END LOOP.

**Sintak Loop :**

```
Loop_name : LOOP  
[statement1]  
[statement2]  
END LOOP loop_name
```

**Sintak While :**

```
Loop_Name : WHILE  
[condition] DO  
[statement1]  
[statement2]  
END WHILE  
Loop_Name;
```

**Sintak Repeat :**

```
[begin_label:] REPEAT  
statement_list  
UNTIL search_condition  
END REPEAT [end_label]
```

# Contoh Stored Procedure

```
select * from players;
```

players (12x18)											
PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1970-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411
3	Dian	Di	1999-05-20	F	2.000	Surabaya	11	60208	Surabaya	0856-4868-	12
4	Diana	Da	1999-06-21	F	2.000	Bojonegoro	12	62115	Bojonegoro	0800-0000-	13
5		(NULL)			0		(NULL)	(NULL)		(NULL)	98
6	Parmenter	R	1964-06-25	M	1.977	Haseltine Lane	80	1234KK	Stratford	070-476537	8467
7	Wise	GWS	1963-05-11	M	1.981	Edgecombe Way	39	9758VB	Stratford	070-347689	(NULL)
8	Newcastle	B	1962-07-08	F	1.980	Station Road	4	6584WO	Inglewood	070-458458	2983
27	Collins	DD	1964-12-28	F	1.983	Long Drive	804	8457DK	Eltham	079-234857	2513
28	Collins	C	1963-06-22	F	1.983	Old Main Road	10	1294QK	Midhurst	010-659599	(NULL)
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
44	Baker	E	1963-01-09	M	1.980	Lewis Street	23	4444LJ	Inglewood	070-368753	1124
57	Brown	M	1971-08-17	M	1.985	Edgecombe Way	16	4377CB	Stratford	070-473458	6409
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608
95	Miller	P	1963-05-14	M	1.972	High Street	33A	5746OP	Douglas	070-867564	(NULL)
100	Parmenter	P	1963-02-28	M	1.979	Haseltine Lane	80	6494SG	Stratford	070-494593	6524
104	Moorman	D	1970-05-10	F	1.984	Stout Street	65	9437AO	Eltham	079-987571	7060
112	Bailey	IP	1963-10-01	F	1.984	Vixen Road	8	6392LK	Plymouth	010-548745	1319
200	Sisi	Ss	(NULL)	F	0		(NULL)	(NULL)	Madura	(NULL)	(NULL)

# Contoh Stored Procedure

**Contoh 1:** Buatlah stored procedure untuk memanggil isi pada tabel players

```
delimiter //
CREATE procedure getPlayers()
BEGIN
SELECT * from players;
end //
delimiter;
```

Perintah Untuk memanggilnya :

```
CALL getPlayers();
```

1 CALL getPlayers();													
/players (12x18)/													
PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO		
2	Everett	R	1970-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411		
3	Dian	Di	1999-05-20	F	2.000	Surabaya	11	60208	Surabaya	0856-4868-	12		
4	Diana	Da	1999-05-21	F	2.000	Bojonegoro	12	62115	Bojonegoro	0800-0000-	13		
5		(NULL)			0		(NULL)	(NULL)		(NULL)	98		
6	Parmenter	R	1964-06-25	M	1.977	Haseltine Lane	80	1234KK	Stratford	070-476537	8467		
7	Wise	GWS	1963-05-11	M	1.981	Edgecombe Way	39	9758VB	Stratford	070-347689	(NULL)		
8	Newcastle	B	1962-07-08	F	1.980	Station Road	4	6584WO	Inglewood	070-458458	2983		
27	Collins	DD	1964-12-28	F	1.983	Long Drive	804	8457DK	Eltham	079-234857	2513		
28	Collins	C	1963-06-22	F	1.983	Old Main Road	10	1294QK	Midhurst	010-659599	(NULL)		
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)		
44	Baker	E	1963-01-09	M	1.980	Lewis Street	23	4444LJ	Inglewood	070-368753	1124		
57	Brown	M	1971-08-17	M	1.985	Edgecombe Way	16	4377CB	Stratford	070-473458	6409		
83	Hope	PK	1956-11-11	M	1.982	Magdalene Road	16A	1812UP	Stratford	070-353548	1608		
95	Miller	P	1963-05-14	M	1.972	High Street	33A	5746OP	Douglas	070-867564	(NULL)		
100	Parmenter	P	1963-02-28	M	1.979	Haseltine Lane	80	6494SG	Stratford	070-494593	6524		
104	Moorman	D	1970-05-10	F	1.984	Stout Street	65	9437AO	Eltham	079-987571	7060		
112	Bailey	IP	1963-10-01	F	1.984	Vixen Road	8	6392LK	Plymouth	010-548745	1319		
200	Sisi	Ss	(NULL)	F	0		(NULL)	(NULL)	Madura	(NULL)	(NULL)		

# Contoh Stored Procedure

- **Contoh 2:** Buatlah stored procedure untuk menghapus seluruh pertandingan yang pernah dimainkan oleh suatu pemain tertentu!

```
DELIMITER $$  
CREATE PROCEDURE DELETE_MATCHES (IN P_PLAYERNO INTEGER)  
BEGIN  
DELETE FROM MATCHES  
WHERE PLAYERNO = P_PLAYERNO;  
END$$  
DELIMITER ;
```

- **Contoh 2:** Hapuslah seluruh pertandingan yang pernah dimainkan oleh pemain nomor 8 dengan penggunaan prosedur DELETE\_MATCHES!

```
CALL DELETE_MATCHES (8);
```

# Contoh Stored Procedure (Contd-2)

- **Contoh 3:** Hapuslah pemain nomor 83 dari basisdata TENNIS. Akan tetapi jika pemain tersebut adalah kapten tim, maka pemain tidak boleh dihapus dari basisdata!

Alur 1 : Mengecek apakah pemain nomor 83 adalah kapten tim:

```
SELECT COUNT(*) AS IS_CAPTAIN  
FROM TEAMS WHERE PLAYERO NO = 83
```

IS\_CAPTAIN

0

Karena pemain nomor 83 bukanlah kapten suatu tim,Sehingga dapat diketahui,  
Kita dapat menghapus semua data mengenai pemain tersebut:

```
DELETE FROM MATCHES  
WHERE PLAYERO NO = 83;  
DELETE FROM COMMITTEE_MEMBERS  
WHERE PLAYERO NO = 83;  
DELETE FROM PENALTIES  
WHERE PLAYERO NO = 83;  
DELETE FROM PLAYERS  
WHERE PLAYERO NO = 83;
```

# Contoh Stored Procedure (Contd-3)

## Percabangan

```
delimiter //  
CREATE PROCEDURE DELETE_PLAYER_83()  
BEGIN  
    DECLARE NUMBER_OF_TEAMS INTEGER;  
  
    SELECT COUNT(*)  
    INTO NUMBER_OF_TEAMS  
    FROM TEAMS  
    WHERE PLAYERO NO = 83;  
  
    IF NUMBER_OF_TEAMS = 0 THEN  
        DELETE FROM MATCHES  
        WHERE PLAYERO NO = 83;  
        DELETE FROM COMMITTEE_MEMBERS  
        WHERE PLAYERO NO = 83;  
        DELETE FROM PENALTIES  
        WHERE PLAYERO NO = 83;  
        DELETE FROM PLAYERS  
        WHERE PLAYERO NO = 83;  
    END IF;  
END//  
delimiter
```

Aktivasi/pemanggilan stored procedure:

```
CALL DELETE_PLAYER_83
```

Menghapus stored procedure:

```
DROP PROCEDURE DELETE_PLAYER_83
```

# Contoh Stored Procedure Percabangan

```
DELIMITER //
CREATE PROCEDURE cobaIF(
IN bil INT(3)
)
BEGIN
/*Deklarasi Variabel*/
    DECLARE str VARCHAR(50);
    if (bil<0) then
        SET str ='Bilangan Negetif';
    ELSE
        SET str='Bilangan Posistif';
    END if;
    SELECT str;
END///
DELIMITER;
```

```
1 call cobaIF(9);
```

Hasil #1 (1×1)

```
str
```

Bilangan Positif

```
1 call cobaIF(-3);
```

Hasil #1 (1×1)

```
str
```

Bilangan Negetif

# Contoh Stored Procedure Pengulangan

```
DELIMITER |
CREATE PROCEDURE ExLooping (
IN bil INT(3)
)
BEGIN
/*Deklarasi Variabel*/
    DECLARE str VARCHAR(50);
    DECLARE i int(3);
    Set i=1;
    Set str='';

    While i<= bil do
        Set str=concat (str," ",i);
        Set i=i+1;
    END WHILE;
    SELECT str;

END;
|
DELIMITER ;
```

```
CALL ExLooping(10);
```

str
1 2 3 4 5 6 7 8 9 10

# Contoh Stored Procedure

IN (Definisi dengan 1 parameter)

- **Contoh 4:** Buatlah stored procedure untuk menampilkan jenis kelamin yang inputannya ditentukan oleh user dengan 1 parameter.

DELIMITER \$\$

```
create procedure getNamaByJenisKelamin(IN jeniskelamin
varchar(1))
begin
select * from players where sex = jeniskelamin;
END$$
```

DELIMITER ;

Perintah Untuk memanggilnya :    **call** getNamaByJenisKelamin ('m') ;  
**call** getNamaByJenisKelamin ('F') ;

players (12x8)											
PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONE NO	LEAGUENO
3	Dian	Di	1999-05-20	F	2,000	Surabaya	11	60208	Surabaya	0856-4668-	12
4	Diana	Da	1999-06-21	F	2,000	Bojonegoro	12	62115	Bojonegoro	0800-0000-	13
8	Newcastle	B	1962-07-08	F	1,980	Station Road	4	6584WIO	Inglewood	070-458458	2983
27	Collins	DD	1964-12-28	F	1,983	Long Drive	804	8457DK	Eltham	079-234857	2513
28	Collins	C	1963-06-22	F	1,983	Old Main Road	10	1294QK	Midhurst	010-599599	(NULL)
104	Moorman	D	1970-05-10	F	1,984	Stout Street	65	9437AO	Eltham	079-987571	7060
112	Bailey	IP	1963-10-01	F	1,984	Vixen Road	8	6392LK	Plymouth	010-546745	1319
200	Siel	Ss	(NULL)	F	0	(NULL)	(NULL)	(NULL)	Madura	(NULL)	(NULL)

players (12x8)											
PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONE NO	LEAGUENO
2	Everett	R	1970-09-01	M	1,975	Stoney Road	43	3575NH	Stratford	070-237893	2411
6	Parmenter	R	1964-06-25	M	1,977	Haseline Lane	80	123KK	Stratford	070-476537	8467
7	Wise	GWS	1963-05-11	M	1,981	Edgecombe Way	39	9758VB	Stratford	070-317689	(NULL)
39	Bishop	D	1956-10-29	M	1,989	Eaton Square	78	9628CD	Stratford	070-393435	(NULL)
44	Baker	E	1963-01-09	M	1,990	Lewis Street	23	4444LJ	Inglewood	070-368753	1124
57	Brown	M	1971-08-17	M	1,995	Edgecombe Way	16	4377CB	Stratford	070-473458	6409
95	Miller	P	1963-05-14	M	1,972	High Street	33A	5746CP	Douglas	070-867564	(NULL)
100	Parmenter	P	1963-02-28	M	1,979	Haseline Lane	80	6494SG	Stratford	070-494993	6524

# Contoh Stored Procedure

## IN (Definisi dengan > 1 parameter)

- **Contoh 4:** Buatlah stored procedure untuk menampilkan jenis kelamin dan kota yang inputannya ditentukan oleh user dengan lebih parameter IN.

```
DELIMITER $$  
create procedure GETjkt( IN kota VARCHAR(20), IN jk  
VARCHAR(1))  
begin  
    select * from players WHERE SEX = jk AND TOWN = kota;  
END$$  
DELIMITER ;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window containing the stored procedure definition. Below it, a results window displays the output of the call to the stored procedure. The results window has a title bar 'players (12x6)' and contains a table with 12 rows of data. The columns are labeled: PLAYENO, NAME, INITIALS, BIRTH\_DATE, SEX, JOINED, STREET, HOUSENO, POSTCODE, TOWN, PHONENO, and LEAGUENO. The data includes various names like Everett, Parmenter, Wise, Bishop, Brown, and Parmenter, along with their respective details such as birth dates, addresses, and phone numbers.

PLAYENO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1970-09-01	M	1.975	Stoney Road	43	3575NH	Stratford	070-237893	2411
6	Parmenter	R	1964-06-25	M	1.977	Haseltine Lane	80	1234KK	Stratford	070-476537	8467
7	Wise	GWS	1963-05-11	M	1.981	Edgecombe Way	39	9758VB	Stratford	070-347689	(NULL)
39	Bishop	D	1956-10-29	M	1.980	Eaton Square	78	9629CD	Stratford	070-393435	(NULL)
57	Brown	M	1971-08-17	M	1.985	Edgecombe Way	16	4377CB	Stratford	070-473458	6409
100	Parmenter	P	1963-02-28	M	1.979	Haseltine Lane	80	6494SG	Stratford	070-494593	6524

Perintah Untuk memanggilnya :

```
call  
GETjkt ('Stratford', '  
m' );
```

# Contoh Stored Procedure IN (Penambahan Data)

- Contoh 5: Buatlah stored procedure untuk memasukan data pada tabel teams yang inputan operasi penambahan, data – data terkait diisikan melalui argumen

```
DELIMITER $$  
create procedure AddTeam( IN TEAMNO  
SMALLINT(10), IN PLAYERO NO SMALLINT(10), IN  
DIVISION VARCHAR(6))  
BEGIN  
    insert into teams values (TEAMNO, PLAYERO NO,  
DIVISION);  
END$$  
DELIMITER ;
```

Perintah Untuk memanggilnya :

```
call AddTeam(4,57,'first');
```

1 SELECT \* from teams;

TEAMNO	PLAYERO NO	DIVISION
1	6	first
2	27	second
3	100	third
81	100	third

1 SELECT \* from teams;

TEAMNO	PLAYERO NO	DIVISION
1	6	first
2	27	second
3	100	third
4	57	first
81	100	third

# Contoh Stored Procedure

```
CREATE TABLE hitung AS  
SELECT amount, 0 AS totalamount  
FROM penalties;
```

```
DELIMITER $$  
CREATE PROCEDURE updateamount (IN  
param1 DECIMAL(7,2))  
BEGIN  
UPDATE hitung  
SET totalamount = (SELECT SUM (won)  
FROM matches WHERE playerno=param1)  
WHERE playerno=param1;  
END$$  
DELIMITER ;
```

```
SELECT * FROM hitung;
```

hitung (2x8)	
amount	totalamount
100,00	0
75,00	0
100,00	0
50,00	0
25,00	0
25,00	0
30,00	0
75,00	0

# Contoh Stored Procedure

```
DELIMITER $$  
CREATE PROCEDURE INSERT_CHANGE (IN CPNO INTEGER, IN CTYPE  
CHAR(1), IN CPNO_NEW INTEGER)  
BEGIN  
INSERT INTO CHANGES (USER, CHA_TIME, CHA_PLAYERNO,  
CHA_TYPE, CHA_PLAYERNO_NEW)  
VALUES (USER, CURDATE(), CPNO, CTYPE, CPNO_NEW);  
END$$  
DELIMITER ;
```

# Contoh Stored Procedure

## Parameter Out

- **Contoh 6:** Buatlah stored procedure untuk menghitung jumlah pemain yang tercatat di dalam tabel PLAYERS, dengan paramater OUT procedure dapat mengubah parameter dan mengirimkan kembali ke program pemanggil.

```
DELIMITER $$  
create procedure JumlahPlayers(OUT jumlah_players INT(3))  
begin  
    select count(playerno) into jumlah_players from players;  
END$$  
DELIMITER ;
```

Perintah Untuk memanggilnya :

```
call JumlahPlayers(@jumlah_players);  
select @jumlah_players;
```

@jumlah\_players

# Contoh Stored Procedure

## Parameter Out

- **Contoh 7:** Buatlah stored procedure untuk menghitung nilai tertinggi dalam tabel Penalties, dengan parameter OUT.

```
DELIMITER $$  
create PROCEDURE ambilPlay(OUT  
besar INT(20))  
begin  
select max(amount) into besar FROM  
penalties;  
END$$  
DELIMITER ;
```

1 SELECT \* FROM penalties;

penalties (4x8)			
PAYMENTNO	PLAYERNO	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
8	27	1984-11-12	75,00

Perintah Untuk memanggilnya :

```
CALL ambilPlay(@besar);  
SELECT @besar;
```

@besar

100

# Contoh Stored Procedure

## 2 Parameter OUT

- **Contoh 8:** Buatlah stored procedure untuk menghitung nilai tertinggi dalam tabel Penalties, dengan 2 parameter OUT.

```
DELIMITER $$  
create PROCEDURE hitungamount(OUT besar INT(20), OUT rata2  
DECIMAL(7,2))  
begin  
select max(amount), avg(amount) into besar, rata2 FROM  
penalties;  
END$$  
DELIMITER ;
```

Perintah Untuk memanggilnya :

```
CALL hitungamount(@besar, @rata2);  
SELECT @besar, @rata2;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window containing the stored procedure definition. Below it, a results window titled "Hasil #1 (2x1)" displays the output of the procedure call. The result is a single row with two columns: "@besar" and "@rata2". The value for "@besar" is 100, and the value for "@rata2" is 60,00.

@besar	@rata2
100	60,00

# Contoh Stored Procedure

## Parameter INOUT

- **Contoh 9:** Buatlah stored procedure untuk menghitung jumlah pemain yang tercatat di dalam tabel PLAYERS yang berjenis kelamin laki-laki atau perempuan yang inputannya dari argument, menggunakan parameter INOUT.

```
DELIMITER $$  
create procedure CountByGender(  
IN gender VARCHAR(2), OUT total INT(3))  
begin  
select count(playerno) into total from players where sex = gender;  
END$$  
DELIMITER ;
```

Perintah Untuk memanggilnya :

```
call CountByGender ('M',@total);  
SELECT @total;
```

The screenshot shows the MySQL Workbench interface with two tabs. The top tab is titled 'Hasil #1 (1x1)' and contains the SQL command: '1 call CountByGender('M',@total); 2 SELECT @total;'. The bottom tab is titled '@total' and displays the result: '8'.

# Trigger

# Trigger

- Kumpulan kode yang terdiri dari procedural dan declarative statements
- Tersimpan dalam katalog dan
- Diaktifkan oleh server database jika operasi yang spesifik dieksekusi dalam database dan jika ada kondisi yang ditentukan

# Trigger

- BEFORE INSERT – activated before data is inserted into the table.
- AFTER INSERT – activated after data is inserted into the table.
- BEFORE UPDATE – activated before data in the table is updated.
- AFTER UPDATE – activated after data in the table is updated.
- BEFORE DELETE – activated before data is removed from the table.
- AFTER DELETE – activated after data is removed from the table.

# Mengakses Nilai Baru dan Lama

- Dalam trigger untuk mengakses data lama dan data baru, data lama dapat direference dengan record OLD dan data baru dapat di reference dengan Record NEW.

OPERASI	NEW (READ/WRITE)	OLD (READ)
INSERT	✓	
UPDATE	✓	✓
DELETE		✓

- Untuk mengacu ke sebuah field dapat ditulis dengan NEW.namafiled atau OLD.namafiled.

# Trigger (Contd-2)

```
<create trigger statement> ::=  
CREATE TRIGGER <trigger name>  
<trigger moment>  
<trigger event>  
[ <trigger condition> ]  
<trigger action>  
  
<trigger moment> ::=  
BEFORE | AFTER | INSTEAD OF  
  
<trigger event> ::=  
{ INSERT | DELETE | UPDATE [ OF <column list> ] }  
{ ON | OF | FROM | INTO } <table specification>  
[ REFERENCING { OLD | NEW | OLD_TABLE | NEW_TABLE }  
AS <variable> ] FOR EACH { ROW | STATEMENT }  
  
<trigger condition> ::= ( WHEN <condition> )  
<trigger action> ::= <begin-end BLOCK>
```

# Contoh Trigger

## Contoh 1: Buatlah tabel CHANGES!

```
CREATE TABLE CHANGES
  (USER CHAR(30) NOT NULL,
   CHA_TIME TIMESTAMP NOT NULL,
   CHA_PLAYERNO SMALLINT NOT NULL,
   CHA_TYPE CHAR(1) NOT NULL,
   CHA_PLAYERNO_NEW INTEGER,
   PRIMARY KEY (USER, CHA_TIME,
   CHA_PLAYERNO, CHA_TYPE));
```

# Contoh Trigger Lanjutan

**Contoh 2:** Buatlah trigger yang akan meng-update tabel CHANGES secara otomatis setiap kali ada penambahan baris baru di dalam tabel PLAYERS!

```
DELIMITER $$
```

```
CREATE TRIGGER INSERT_PLAYERS
AFTER INSERT ON PLAYERS FOR EACH ROW
BEGIN
INSERT INTO CHANGES
(USER, CHA_TIME, CHA_PLAYERNO, CHA_TYPE, CHA_PLAYERNO_NEW)
VALUES (USER, CURDATE(), NEW.PLAYERNO, "I", NULL);

END$$
DELIMITER ;
```

# Contoh Trigger Lanjutan

**Contoh 4:** Buatlah trigger yang meng-update tabel CHANGES setiap kali ada baris di dalam tabel PLAYERS yang dihapus!

```
DELIMITER $$  
  
CREATE TRIGGER DELETE_PLAYER  
AFTER DELETE ON PLAYERS FOR EACH ROW  
BEGIN  
    CALL INSERT_CHANGE (OLD.PLAYERNO, 'D', NULL);  
  
END$$  
DELIMITER ;
```

# Contoh Trigger Lanjutan

**Contoh 5:** Buatlah trigger yang meng-update tabel CHANGES setiap kali ada baris di dalam tabel PLAYERS yang di-update!

```
DELIMITER $$
```

```
CREATE TRIGGER UPDATE_PLAYER
AFTER UPDATE ON PLAYERS FOR EACH ROW
BEGIN
CALL INSERT_CHANGES (NEW.PLAYERNO, 'U', OLD.PLAYERNO);

END$$
DELIMITER ;
```

# Contoh Trigger Lanjutan

**Contoh 6 :** Buat trigger untuk menyimpan history division, jika division berubah, maka division lama harus disimpan ke tabel history division.

```
DELIMITER $$  
DROP TRIGGER if EXISTS  
coba_update_teams$$  
CREATE TRIGGER coba_update_teams  
AFTER UPDATE ON teams FOR EACH ROW  
BEGIN  
INSERT INTO history_div_teams VALUES  
(NOW(), old.teamno, old.division,  
USER());  
END$$  
DELIMITER ;
```

Contoh Penggunaan Trigger

```
UPDATE teams SET division =  
"thrid" WHERE teamno=1;
```

```
SELECT * FROM teams;  
SELECT * from history_div_teams;
```

teams (3x5)		history_div_teams (5x8)		
waktu	teamno	playerno	division	oleh
2019-04-11 20:04:49	1	(NULL)	first	root@localhost
2019-04-11 20:05:09	1	(NULL)	first	root@localhost
2019-04-11 20:05:24	1	(NULL)	second	root@localhost
2019-04-11 20:26:20	1	(NULL)	thrid	root@localhost
2019-04-11 20:26:26	1	(NULL)	coba	root@localhost
2019-04-11 20:26:30	1	(NULL)	lagi	root@localhost
2019-04-11 20:31:14	1	(NULL)	sekali	root@localhost
2019-04-11 20:33:12	1	(NULL)	firste	root@localhost

Untuk melihat semua division yang pernah digunakan oleh teams yang bernomor =1

```
(SELECT NOW() waktu, teamno, division  
FROM teams WHERE teamno=1)  
UNION  
(SELECT waktu, teamno, division FROM  
history_div_teams WHERE teamno=1)  
ORDER BY waktu DESC;
```

2019-04-11 20:26:20	1	thrid
2019-04-11 20:05:24	1	second
2019-04-11 20:05:09	1	first
2019-04-11 20:04:49	1	first

# Contoh Trigger Lanjutan

Trigger yang pertama mempunyai kekurangan yaitu ketika ada perubahan di tabel teams walaupun tdk mengubah kolom division, maka statement INSERT di tabel history akan dijalankan.

```
DELIMITER $$  
DROP TRIGGER if EXISTS  
coba_update_teams$$  
CREATE TRIGGER coba_update_teams  
AFTER UPDATE ON teams FOR EACH ROW  
BEGIN  
if old.division <> new.division  
then  
INSERT INTO history_div_teams  
VALUES (NOW(), old.teamno,  
old.division, USER());  
END if;  
END$$  
DELIMITER ;
```

Contoh Penggunaan Trigger

```
UPDATE teams SET division =  
"sekali" WHERE teamno=1;  
  
UPDATE teams SET division =  
"firste" WHERE teamno=1;  
  
UPDATE teams SET division =  
"first" WHERE teamno=1;  
  
SELECT * FROM teams;  
SELECT * from history_div_teams;
```

Untuk melihat semua division yang pernah digunakan oleh teams yang bernomor =1

```
(SELECT NOW() waktu, teamno,  
division FROM teams WHERE  
teamno=1)  
UNION  
(SELECT waktu, teamno, division  
FROM history_div_teams WHERE  
teamno=1)  
ORDER BY waktu DESC;
```

# Contoh Trigger Lanjutan

**Contoh 7 :** Buat trigger akan dieksekusi ketika ada perubahan teamno di tabel teams yang akan melakukan update ke tabel history\_div\_teams untuk menyesuaikan Teamno agar relasi tidak terlepas.

```
DELIMITER $$  
DROP TRIGGER if EXISTS coba_update_teams$$  
CREATE TRIGGER coba_update_teams  
AFTER UPDATE ON teams FOR EACH ROW  
BEGIN  
if old.division <> new.division then  
INSERT INTO history_div_teams VALUES (NOW(),  
old.teamno, old.division, USER());  
END if;  
if old.teamno <> new.teamno then  
UPDATE history_div_teams SET  
teamno=new.teamno WHERE teamno=old.teamno;  
END if;  
END$$  
DELIMITER ;
```

Contoh Penggunaan Trigger

```
ALTER table history_div_teams  
ADD COLUMN playerno  
SMALLINT(6) AFTER teamno;
```

```
SELECT * from history_div_teams;
```

# Contoh Trigger Lanjutan

**Contoh 7 :** Buat trigger akan dieksekusi ketika ada perubahan teamno di tabel teams yang akan melakukan update ke tabel history\_div\_teams untuk menyesuaikan Teamno agar relasi tidak terlepas.

```
DELIMITER $$  
DROP TRIGGER if EXISTS coba_update_teams$$  
CREATE TRIGGER coba_update_teams  
AFTER UPDATE ON teams FOR EACH ROW  
BEGIN  
if old.division <> new.division then  
INSERT INTO history_div_teams VALUES (NOW(),  
old.teamno, old.division, USER());  
END if;  
if old.playerno <> new.playerno then  
UPDATE history_div_teams SET  
playerno=new.playerno WHERE  
playerno=old.playerno;  
END if;  
END$$  
DELIMITER ;
```

## Contoh Penggunaan Trigger

```
UPDATE teams SET teamno  
= 111 WHERE teamno=1;
```

```
UPDATE teams SET  
playerno =111 WHERE  
playerno=6;
```

## 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.

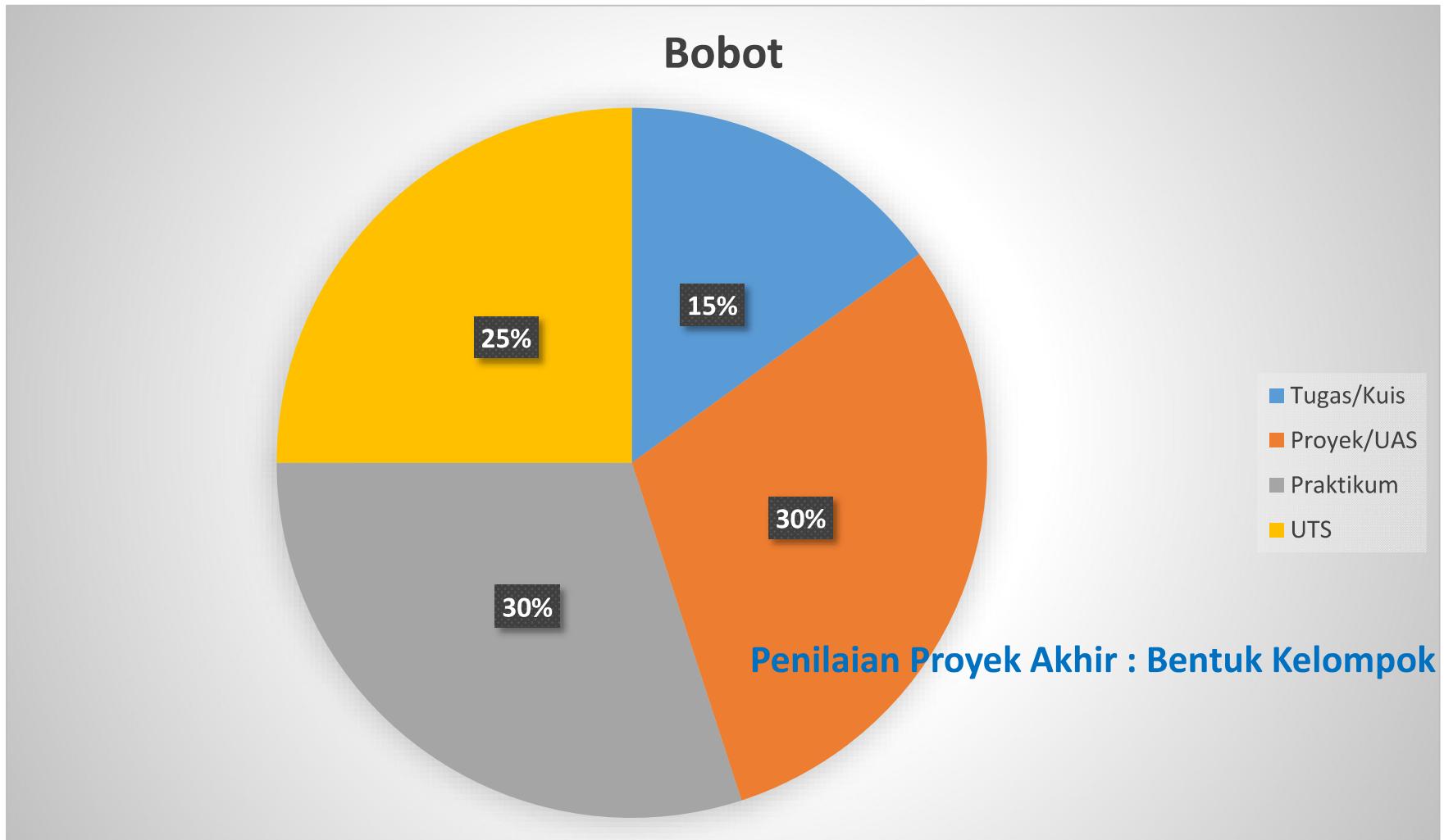
## □ Bimbingan dan Arahan

- 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](https://github.com/doniaft)
- ❑ Email : [doniaft@gmail.com](mailto: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” 3<sup>rd</sup> 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