

PENGELOLAAN BASIS DATA

06. SQL Query Select

07. SQL Query Where / Groups

08. SQL Query Join

09. Stored Procedure / Function

10. Trigger

11. SQL DDL

Doni Abdul Fatah

Universitas Trunojoyo Madura

Pokok Bahasan (Revisi)

01. Database Design Review

02. Database User Account Management

03. Database Backup & Recovery

04. Database Replication

05. Database Optimization

06. SQL Query Select

07. SQL Query Where / Groups

08. SQL Query Join

09. Stored Procedure / Function

10. Trigger

11. SQL DDL

12. Optimising DB Structure

13. Project

14. UAS

Penjelasan Pokok Bahasan (Revisi)

Database Design Review

- Relational Model
- ERD
- Conceptual Database Design
- Logical Database Design
- Physical Database Design

Database User Account Management

- Grant
- Revoke
- Privelege
- Adding User
- Limiting User Resource
- Tugas Besar 1

Database Backup & Recovery

- Backup
- Restore
- Check
- Repair
- Table Maintenance

Database Replication

- Overview
- Setup Replication
- Sql command (slave)

Penjelasan Pokok Bahasan (Revisi)

Database Optimization	SQL Query Select	SQL Query Where / Groups	SQL Query Join
<ul style="list-style-type: none">• Definisi• Design limitation / tradeoff• Benchmarking• Explain• Query performance	<ul style="list-style-type: none">• Select, where• Insert, update, delete• Join• Union• Truncate• replace	<ul style="list-style-type: none">• Select, where• Insert, update, delete• Join• Union• Truncate• replace	<ul style="list-style-type: none">• Select, where• Insert, update, delete• Join• Union• Truncate• replace

Penjelasan Pokok Bahasan (Revisi)

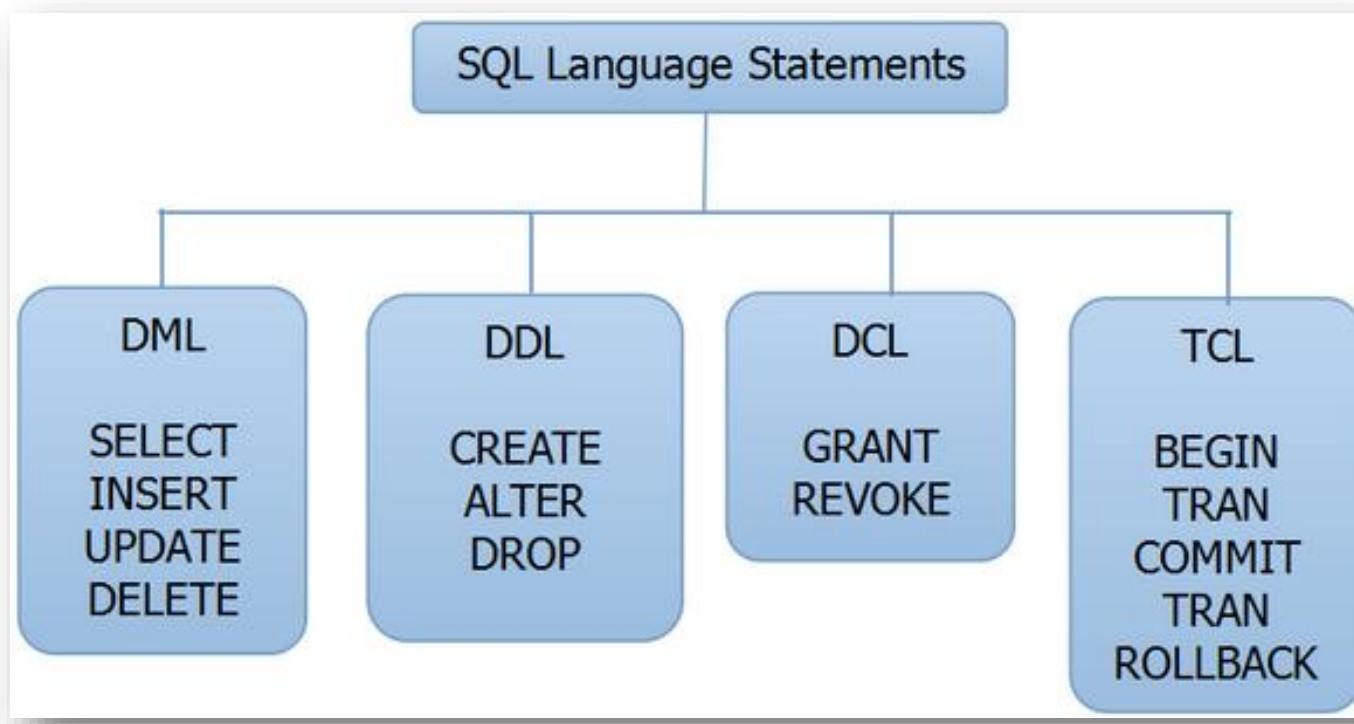
Stored Procedure / Function	Trigger	SQL DDL	Optimising DB Structure
<ul style="list-style-type: none">• Definisi• Parameter• Body• Local variable• Set• Flow control• Call stored procedure• Error message	<ul style="list-style-type: none">• Konsep Trigger• Create trigger• Trigger as Integrity constraints• Trigger and catalog	<ul style="list-style-type: none">• Definisi• Creating tables• Data types• Temporary table• Copying tables• Naming tables and columns• Table options• Integrity constraints• Primary keys• Alternate keys• Foreign keys	<ul style="list-style-type: none">• Konsep Design choices• Indexes• Multiple column indexes• Tuning server parameter• How mySQL uses memory

01. Pengelolaan BasisData

- 1) Stored Procedure / Function
- 2) Trigger
- 3) Projek Akhir
- 4) Referensi

SQL

b) DDL DML



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

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
```

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.

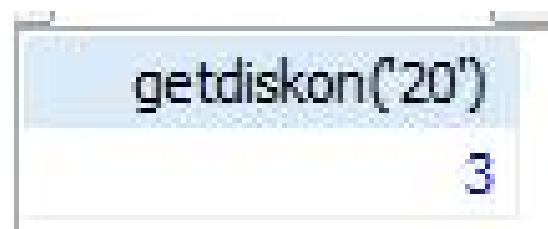
Function Lanjutan - Contoh

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

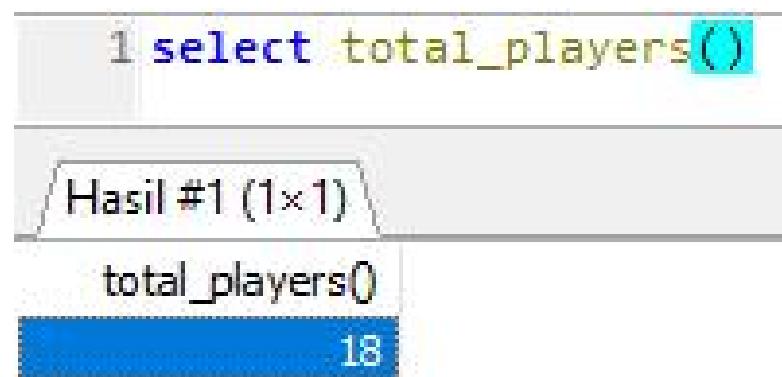
Function Lanjutan - Contoh

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 ;
```

Perintah Untuk memanggilnya :

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

The screenshot shows the MySQL Workbench interface. At the top, there is a command line with the following text:
1 call GETjkt('Stratford', 'm');
Below the command line is a table titled "players (12x6)". The table has the following data:

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
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

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

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 alamat 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 alamat 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;
```

8) Proyek Akhir

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.

9) 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

10) Contact

Contact

- Bahan Kuliah : github.com/doniaft
- Email : doniaft@gmail.com
- WA/Telegram :
- Komting TIF 4D : Ali Mustofa : 081 231 202 253

II) Referensi

Referensi (I)

- ❑ 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
- ❑ Raghu Ramakhrisnan, Johannes Gehrke , “Database Management System” 6rd Edition, Mc Graw Hill, 2003 - 2006
- ❑ Arief, Rudiyanto M, Pemrograman Basis Data menggunakan Transact-SQL dengan Microsoft SQL Server 2000, Andi Yogyakarta.
- ❑ 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
- ❑ Elmasri and Navathe, Fundamental of Database Systems 4th Edition, Addison-Wesley, 2004
- ❑ Silberschatz, Korth and Sudarshan, Database System Concepts, 5th Edition, Mc Graw Hill, International Edition, 2006.
- ❑ Connoly, Thomas and Begg, Carolyn: Database Sytems 4th edition, Prentice Hall, 2005

Referensi (2)

- Andrea Tar. 2012. PHP and MySQL 24-Hour Trainer
- Brett McLaughlin. 2012. PHP & MySQL- The Missing Manual. USA-Brett McLaughlin. USA-O'REILLY Media
- Brett McLaughlin. 2013. PHP & MySQL- The Missing Manual, 2nd Edition. USA-Brett McLaughlin. USA-O'REILLY Media
- Head First PHP & MySQL
- Kroenke, David. 2013. Database Processing 12th Edition
- Mysql Official. 2016. MySQL 5.7 Reference Manual-en
- PHP6 and MySQL Bible by Steve Suehring
- Rochkin Mark. 2013. Expert PHP and MySQL
- Ruehning, dkk. php_mysql_javascript__html5_all-in-one_for_dummies
- Sams.Sams.Teach.Yourself.PHP.MySQL.and.Apache.All-in-One.ISBN0672326205
- Solichin, Achmad. Pemrograman Web dengan PHP MySQL
- Tutorialpoints.com - mysql tutorial
- Valade, Janet. PHP & MySQL Web Development All-in-One Desk Reference For Dummies. CanadaWiley Publishing, Inc
- Widigdo, Anon Kuncoro. 2003. php dan mysql
- <https://www.duniaIlkom.com/tutorial-belajar-mysql-dan-index-artikel-mysql/#mysqllanjutan>