# DATABASE INTEGRITY

## Integrity Constraints

*Constraint :* rules given to a table for entered data is valid.

Integrity Constraints will protect the database from damage by ensuring that allowed changes do not result in data inconsistencies.

Types of Integrity Constraint :

1. Inter-relational *Constraint* (constraint in one table / relation).
   This Constraint can be classified as **Entity Integrity Constraint** and **Domain Constraint**.
2. Intra-relasional *Constraint* (constraint involves more than one table / relation).
   This constraint is **referential integrity**.
3. **Enterprise Integrity (User Defined Integration)**

# DATABASE INTEGRITY

Objective of database integrity are :

1. To ensure that database integrity works.
2. To protect the database from incomplete, inaccurate, and inconsistent data.
3. To ensure the quality of data so that data can be a reference.

In addition to the above constraints, the effort that supports for guaranteed data quality is data processing by using the **store procedure** and the **trigger**

# DATABASE INTEGRITY

Therefore, material discuss in this section includes :

1. ***Entity integrity constraints***
2. ***Domain constraints***
3. ***Referential integrity constraints***
4. ***Integritas Enterprise (User Defined Integration)***
5. ***Store Procedure***
6. ***Trigger***

# DATABASE INTEGRITY

## Entity integrity constraints

- This rule is applied by declaring a primary key for each entity to ensure that no rows in the table have the same value (duplicate rows).
- Entity integrity constraint forces the integrity of a column from a table through index, unique, primary key, not null and other constraints.

Example :

```
CREATE TABLE Buying
            (IDBuy smallint, IDModel smallint,
            descModel VARCHAR(40),);
            PRIMARY KEY (IDBuy),
            UNIQUE (IDModel));
```

implementation of entity integrity constraint

# DATABASE INTEGRITY

**Domain constraints**

- Domain constraint is the most simple of integrity constraints. Each new data entered, it will be checked immediately by the system.

- Domain constraints of an attribute include : data type, width, range of values, uniqueness and possibly null data.

- use of Domain constraints causes no data to violate the range of values

# DATABASE INTEGRITY

## Domain constraints

**Types of constraint on Domain Constraints :**

**PRIMARY KEY Constraint**

It's used to specify the attribute on table, such as NOT NULL, UNIQUE and join operations.

**FOREIGN KEY Constraint**

Foreign Key is used to connect two relation. The foreign key attributes of a relation can be used to connect with the primary key attribute of others table.

**UNIQUE Constraint**

It's used to ensure that the data of an attribute not allowed duplicate.

# DATABASE INTEGRITY

## Domain constraints

**Types of constraint on Domain Constraints :**

**CHECK Constraint**

It’s used to restrict a value to an attribute with a specified value that is allowed.

**DEFAULT Constraint**

It's used to give a specified value of attribute, if an attribute is not filled then the default value will be saved.

**NOT NULL**

It’s used to ensure that an attribute has no null value.

# DATABASE INTEGRITY

## Domain constraints

Example :

```
Create table student (regNo char(15) primary key,
        name varchar(25) not null unique, address varchar(30),
        sex char(1) default ‘M’ check (sex = ‘M’ or sex = ‘F’));
```

Note :

**regNo** attribute is primary key, not null, unique and identifier attribute

**name** attribute is not null and unique

**Sex** attribute can be filled only ‘M’ or ‘F’ character ('M' is male and 'F' is female),
If an attribute value not entered then an attribute will be filled 'M'

# DATABASE INTEGRITY

## Referential integrity constraints

- A set of rules that governs the relationship between a table's primary key with another table's foreign key to maintain data consistency

- Maintains the integrity of the value of an attribute in a table / relation that refers to the value of an attribute from another table.

- The referential integrity includes the operations such as :
  - Insert operations
  - Delete operations
  - Update operations

# DATABASE INTEGRITY

## Referential integrity constraints

Referential integrity operations can be written :

REFERENCES tbl_name (fliedkey) [ON UPDATE {RESTRICT | CASCADE | SET NULL}]
or
REFERENCES tbl_name (fliedkey) [ON DELETE {RESTRICT | CASCADE | SET NULL}]

Note :
UPDATE : states the action if on the parent table occurs the update operations.
DELETE : states the action if on the parent table occurs the delete operations
RESTRICT : operations of the delete and the update on the parent table are denied.
CASCADE : If the foreign key value of the parent table is updated (UPDATE) , then the foreign key value in the reference table will be updated with the primary key value of the parent table.
While, If the parent table occurs DELETE operations, then the foreign key value in the reference table will be deleted .
SET NULL : the foreign key value will be filled NULL, if the primary key value of the parent table is deleted or updated.

# DATABASE INTEGRITY

## Referential integrity constraints

Example :

Studen Table

CREATE TABLE student (regNo char(8), name varchar(20), PRIMARY KEY (regNo));

Course Table

CREATE TABLE course (course_ID char(3), course_name varchar(20), PRIMARY KEY (course_ID));

Takes Table

CREATE TABLE takes(regNo char(8), course_ID char(3), grade dec(3,2), PRIMARY KEY (regNo, course_ID), FOREIGN KEY (regNo) REFERENCES student (regNo) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (course_ID) REFERENCES course (course_ID) ON DELETE CASCADE ON UPDATE CASCADE);

# DATABASE INTEGRITY

**Enterprise Integrity / User Defined Integration / General constraints**

This constraints refers to applications, business rules are defined by user or DBA (Database Administrator)

Example :

a branch may only have a maximum of 20 employees, so that each new employee to be placed in a branch then checked whether the number of employees in the branch has reached 20 people, then the placement of the employee will be placed in another branch

# DATABASE INTEGRITY

## Store Procedure

Store Procedure is the new facility in MySQL 5.x. it is contain a set of SQL statements that stored in MySQL server.

store procedure is a procedure (like subprogram) that stored in database. Procedure is called through CALL operations and can return a value through output variable.

Benefits of store procedure are :

1. Minimize the request traffic from application to database, because all the business process are performed in MySQL database and applications will receive the process result only.
2. Increase the security of database, a user can not access certain table directly but can access through the store procedure.

# DATABASE INTEGRITY

General Syntax :

CREATE PROCEDURE <procedure_name> (parameter)
<procedure_characteristic>
<program_body>

## Example 1:

Store procedure is used to display the employee based on certain job from emp table :

```
Delimiter  $$  /* change delimiter ; into $$ */
Create procedure display_emp (xjob char(10))
Begin
    Select * from emp where job=xjob;
End;
$$
```

*the changes of delimiter is temporary (during the MySQL database is still active)*

procedure can be accessed through :

**Mysql>call display_emp('salesman') $$**

# DATABASE INTEGRITY

## Example 2 :

Store procedure is used to calculate the number of record from emp table. The number of record is stored to param1 variable:

```
Delimiter $$
Create procedure display_emp 1 (out param1 int)
Begin
    Select count(*) into param1 from emp;
End;
$$
```

procedure can be accessed through :

**Mysql>call display_emp1(@hasil) $$**
**Mysql>select @hasil $$**

Type of parameter is used in procedure consist of :

IN → parameter variable serve as input (opsional)
OUT → parameter variable serve as output ( stores output value)
INOUT → parameter variable serve as output and input

# DATABASE INTEGRITY

Example 3 :

Store procedure is used to fill table grade with table structure is:

Create table grade (ID char(2) primary key, name varchar(10), exam1 dec(5,2), Exam2 dec(5,2), final dec(5,2), letter_value char(1))

```
Delimiter //
Create procedure grade_fill (x_ID char(2), x_name varchar(10), x_exam1 dec(5,2), x_exam2 dec(5,2))
Begin
    declare x_final dec(5,2);                    /* declaration of temporary variable */
    declare x_letter_value char(1);
    set x_final = (x_exam1 * 0.3 + x_exam2 *. 0.7);
    set x_letter_value = if(x_final>=85,'A',if(x_final>=70,'B', if(x_final>=60,'C', if(x_final>=50,'D','E'))));
    insert into grade values (x_ID, x_name, x_exam1, x_exam2, x_final, x_letter_value);
    Select * from grade;
End;
//
```

procedure can be accessed through :

**Mysql>call grade_fill('11', 'Ahmad', 80, 90) //**

# DATABASE INTEGRITY

## Trigger

Trigger is a type specially of store procedure that attached to the table. It's accessed automatically when occurs data manipulation of the table (such ac insert, update and delete operations).

General Syntax :

CREATE TRIGGER *trigger_name trigger_time trigger_event* ON *table*
FOR EACH ROW *trigger_statement*

Note :
***Trigger_time*** shows when trigger is executed consisting of BEFORE or AFTER
***Trigger_event*** shows when trigger is executed consisting of :

- INSERT : trigger is active, if insert operations is executed
- UPDATE : trigger is active, if update operations is executed
- DELETE : trigger is active, if delete operations is executed

# DATABASE INTEGRITY

## *Trigger*

Example 1 :

```
Create trigger i_dept before insert on emp for each row
Begin
    declare xdname char(10);
    declare xloc char(10);
    set  xdname = if(new.deptno='10','ACCOUNTING', if(new.deptno='20','RESEARCH',
                    if(new.deptno='30','SALES','OPERATIONS')));
     set  xloc = if(new.deptno='10','NEW YORK', if(new.deptno='20','DALLAS',
                    if(new.deptno='30','CHICAGO','BOSTON')));
     insert into dept values (new.deptno, xdname, xloc);
End;
//
```

If in the emp table is executed insert operations :

Mysql> **insert into emp values ('8888','JOHN','SALESMAN','7698','1984-10-12',1450,0,'30');**

Then the trigger is executed automatically

# DATABASE INTEGRITY

## Combine store procedure and trigger

Consider the two tables :

```
CREATE TABLE student (regNo  char(8), name varchar(20), PRIMARY KEY (regNo));
CREATE TABLE log_student (event  varchar(15), time datetime);
```

Create a store procedure is used to input the student table.
Next, create a trigger to store the insert activity in the log table automatically .

### Store Procedure

```
Delimiter //
Create procedure i_student (xregNo char(2),xname varchar(20))
Begin
    insert into student values (xregNo, xname);
    select * from student;
End
//
```

### Trigger

```
Create trigger  i_log  after  insert on  student for each row
Begin
    insert into log_student  values ('add data', now());
End
//
```

Execution :
Mysql>call i_student ('11','Joni')//
Mysql>call i_student ('22','Smith')//

student

| regNo | name |
|---|---|
| 11 | Joni |
| 22 | Smith |

log_student

| Event | Time |
|---|---|
| Add data | 2017-10-19 11:30:00 |
| Add data | 2017-10-19 11:31:10 |

# DATABASE INTEGRITY

## Practical Task

1. Create the triggers are used to save data in log_student table, if the update or delete operations in student table is executed :
   a. If update operations is executed, then the event attribute of log_student table is filled ‘update data’
   b. If delete operations is executed, then the event attribute of log_student table is filled ‘delete data’

log_student

| Event | Time |
|---|---|
| Add data | 2017-10-19 11:30:00 |
| Add data | 2017-10-19 11:31:10 |
| …. | |
| Update data | 2017-10-19 11:36:10 |
| Delete data | 2017-10-19 11:38:20 |

records are stored from trigger operations result

# DATABASE INTEGRITY

2. Consider the tables follows :

```
CREATE TABLE stock (ID char(5) primary key, name varchar(20),
                    unit varchar(10), stock_quantity int(5))

CREATE TABLE  sale (sale_no char(5) Primary key, cons_name varchar(10),
                    address varchar(20), city varchar(20), sale_date date,
                    ID char(5), sale_quantity int(4))

CREATE TABLE purchase (purchase_no char(5) Primary key,
                     supplier_name varchar(10), purchase_date date,
                     ID char(5), purchase_quantity int(4))
```

In the trigger statement, we can access table records before or after a process using **NEW** and **OLD.**

**NEW** is used to to retrieve records is used to to retrieve records to be processed (insert or update), while **OLD** is used to access records already saved (update or delete)

Example :

```
Update stock set stock_quantity = stock_quantity - new.sale_quantity
where ID = new.ID;
```

# DATABASE INTEGRITY

a. Create the store procedure is used to enter (insert) the data of sale
b. Create the store procedure is used to enter (insert) the data of purchase
c. Create the trigger is used to subtract the stock_quantity attribute of stock table with the sale_quantity attribute of sale table based on transaction in point a.
d. Create the trigger is used to add the stock_quantity attribute of stock table with the purchase_quantity attribute of purchase table based on transaction in point b.
e. Create the trigger is used to subtract the stock_quantity attribute of stock table with the purchase_quantity attribute of purchase table, if in the purchase table is done the data deletion (purchase transaction is canceled). the subtraction process is done in accordance with the deleted ID in the purchase table.
f. Create the trigger is used to add the stock_quantity attribute of stock table with the sale_quantity attribute of sale table, if in the sale table is done the data deletion (sale transaction is canceled). the subtraction process is done in accordance with the deleted ID in the sale table.
g. Create the trigger is used to update the stock_quantity attribute of stock table with the purchase_quantity attribute of purchase table, if in the purchase table is done the data update. the update process is done in accordance with the update ID in the purchase table.
h. Create the trigger is used to update the stock_quantity attribute of stock table with the sale_quantity attribute of sale table, if in the sale table is done the data update. the update process is done in accordance with the update ID in the sale table.

# DATABASE INTEGRITY

sale

| sale_no | cons_name | address | city | sale_date | ID | sale_quantity |
|---|---|---|---|---|---|---|
| S0001 | Smith | Jl. A. Yani 20 | Semarang | 2017-10-11 | 22222 | 5 |
| | | | | | | |
| …… | | | | | | |

stock

| ID | name | unit | Stock_quantity |
|---|---|---|---|
| 11111 | shirt | piece | 10 |
| 22222 | jacket | piece | 15 |
| …… | | | |

Stock_quantity + sale_quantity

Stock_quantity - sale_quantity

Example :
Update stock set stock_quantity = stock_quantity - new.sale_quantity
where ID = new.ID;

How to update stock_quantity,
if the update operation is done ???

if the delete operation is done
(transaction is canceled)

if the insert operation is done

Stock_quantity + purchase_quantity

Stock_quantity - purchase_quantity

purchase

| purchase_no | supplier_name | ID | purchase_quantity |
|---|---|---|---|
| P0001 | CV. Prima | 22222 | 20 |
| | | | |
| …… | | | |

The report of the practical task is sent via email maximum at monday (23 October 2017)