****
SQL Introduction
****

--------------------------- Day 1 --------------------------------

Introduction:

Data - Raw Facts and Figure

Information - Processed data

Database -> collection of related information

DBMS -> (user <--------> DBMS <----------> Database)

RDBMS -> Information stored in form of tables and each table is connected to other by keys.

Entity -> About which information is going to store in database table.

Tables -> Information stored in form of rows and columns.

columns -> Columns,Fields,Attributes(Information type)

Rows -> Row,Record,Tuples(Actual Information)

SQL -> Structure query Language used to interact with database

----

Query:-

1) Creating Database
```
-- CREATE DATABASE database_name;
```

2) Use Database
```
-- USE database_name;
```

3) Create Table
```
-- CREATE TABLE table_name
   (
    FIELD_NAME DATA_TYPE(SIZE),
    FIELD_NAME DATA_TYPE(SIZE),
    FIELD_NAME DATA_TYPE(SIZE),
    FIELD_NAME DATA_TYPE(SIZE)
   );
```

4) To show all databases
```
-- SHOW DATABASES;
```

5) To show all tables inside a database
```
-- Go inside databse than
-- SHOW TABLES;
```

6) For checking Table structure use DESC (i.e Description)
```
-- DESC table_name;
```

```
-----------------------------------------------------------------------

------------------- Day 2 ------------------------------

Main Data Types in MYSQL:-

a) INT or INTEGER    ----- roll INT(5);
b) DEC or DECIMAL    ----- price DECIMAL(4,2);  (12.34)
c) CHAR or CHARACTER ----- name CHAR(20);
d) VARCHAR       ----- name VARCHAR(50);    max length=255
e) TEXT              ----- address TEXT;       max length=65535 characters
f) DATE (yyyy-mm-dd) ----- age DATE;
g) DATETIME(date+ hh:mm:ss) ----- Date_of_join DATETIME;
h) TIMESTAMP         ------ login_dt TIMESTAMP;


Query:-

7) Insert Data Into database
-- INSERT INTO table_name (column1,column2,column3,....)
      VALUES(value1,value2,value3,.......);
or
-- INSERT INTO table_name VALUES(value1,value2,value3,.......);
or
-- INSERT INTO table_name (column2,column1,column3,....)
      VALUES(value2,value1,value3,.......);
or
-- INSERT INTO table_name (column1,column2,column3)
      VALUES(value1,value2,value3);
or
-- INSERT INTO table_name (column1,column2,column3)
      VALUES(value1,value2,value3),
            (value1,value2,value3),
            (value1,value2,value3);

8) Select is used to selct data from a database and retrive the information
  a) Select all M table_name;
  b) Select Particular column
  -- SELECT column_name1,column_name2,.... FROM table_name;


9) Select with limit - used to specify some number of records
-- SELECT * FROM table_name LIMIT 5;    ----- Shows first 5 records
   (can also used with assending and desending functions)
-- SELECT salery from Employee DESC LIMIT 1,1;  (start from,no.of rows shows)

Note: indexing starts from 0 in my sql;


10) Single Quotes problem while entering value containing single quotes
-- INSERT INTO my_tab(t_id,c_name,address)
      VALUES(142,'K.K's Company','DELHI');

To solve this we can use:-
      a) USE backslash
            EX:- 'K.K\'s Company'
      b) Use two time single quotes
            Ex:- 'K.K''s Company'
```

---------------------- Day 3 --------------------------

11) Where Clause = It is used to search for a specific data.

-- SELECT * FROM table_name WHERE column_name operator 'value';
-- SELECT column_name FROM table_name WHERE column_name operator 'value';

Operators used in MySQL:

a) Equal	=
b) Not Equal	!= , <>
c) Greater Than	>
d) Less Than	<
f) Greater Than Or Equal	>=
g) Less Than or Equal	<=
h) BETWEEN	Betwen an inclusive range
i) LIKE	Search for a pattern
j) In	to specify multiple possible values for a column

Note:- Operators can be used with both integers and charactors as well.....

12) NUll Values :- NULL values represent mising unknown data.

-- NULL not equal to 0

-- IS NULL -> This is used to select only the records with NULL values in the column.
-- SELECT column_name FROM table_name WHERE column_name IS NULL;

--IS NOT NULL -> this is used to select only records with no NUll values in the column.
--SELECT column_name FROM table_name WHERE column_name IS NOT NULL;

13) AND -> this operator displays a record if both the first condition AND the second Condition are true.
-- SELECT *FROM table_name WHERE column_name = 'value' AND column_name ='value'

14) OR -> This operator displays a record if either the first condition OR the second condition is true.
-- SELECT * FROM table_name WHERE column_name='value' OR column_name='value'

15) Combination Of AND and OR operator:-
-- SELCET * FROM table_name WHERE column_name='value' AND (column_name='value' OR column_name1 = 'value');
Ex:
-- SELECT * FROM STUDENT WHERE name='Anu' AND (sty_id=5 OR address='Kolkata');

Note : Combination queries are written for getting more accurate result.
-----: AND and OR Operators can be used in any combiantion according to the need.

16) IN Operator : The In Operator allows you to specify multiple values in a Where clause.
-- SELECT * FROM table_name WHERE column_name IN('value1','value2',.....);

-- SELECT emp_id,emp_name FROM emp WHERE city IN('Delhi','Ranchi')

17) NOT IN : Oposite of IN operator
-- SELECT * FROM table_name WHERE column_name NOT IN('value1','value2',.....);
EX:
-- SELECT * FROM Student WHERE name NOT In('Anu','Sonu');

--------------------------- Day 4 ---------------------------------------

18) BETWEEN OPERATOR : The BETQEEN operator selects values within a range. The values can be numbers,text,or dates.
-- SELECT * FROM table_name WHERE column_name BETWEEN value1 AND value2;
Ex:
-- SELECT * FROM STUDENT WHERE stu_id BETWEEN 6 AND 8; (note: both 6 and 8 is included in MYSQL)
-- SELECT * FROM student WHERE name BETWEEN 'B' AND 'J';(B included but not J)
-- SELECT * FROM STUDENT WHERE date BETWEEN '2000/01/04' AND '2001/02/14';
(include both dates)

19) NOT BETWEEN : To display data which is not in the range.
-- SELECT * FROM tbale_name WHERE column_name NOT BETWEEN value1 AND value2;

20) BETWEEN with IN
-- SELECT * FROM table_name WHERE(column_name BETWEEN value1 AND value2) AND column_name IN(value1,value2);
Ex:-
-- SELECT * FROM emp WHERE(salary BETWEEN 25000 AND 50000) AND dept IN('IT','HR');

21) BETWEEN with NOT IN
-- SELECT * FROM table_name WHERE(column_name BETWEEN value1 AND value2) AND NOT column_name IN(value1,value2);
EX:-
-- SELECT * FROM emp WHERE(salary BETWEEN 25000 AND 50000) AND NOT dept IN('IT','HR');

22) LIKE : The like operator is used to search for a specified pattern in a column.
-- SELECT * FROM table_name WHERE column_name LIKE 'pattern';

****
      Type OF Patterns
****
Wildcards : Wildcards are used to search for data within a table. These characters are used with the LIKE operator.

a) % ->   Zero or more characters

```
--------> 'Geek%'  - All starting with geek.
--------> '%shows' - All ending with shows.
--------> '%sh%'   - All containing with sh.
```

```
b) _ ->     One Single Character

--------> 'Show_' - Starting with show then any character.
--------> '_eek'  - any character then eek.
--------> 'G_e_K' - G then any character, then e then any character, then k.


c) [charlist] ->  Sets and ranges of characters to match
d) [^charlist] or [!charlist] ->   Matches only a character NOT specified within
the brackets

Note:- Both wildcards c and d now not support in Mysql


23) Not LIKE : Oposite of like oiperator
-- SELECT * FROM table_name WHERE column_name NOT LIKE 'pattern';
EX:
SELECT * FROM student WHERE name NOT LIKE '%nu';




----------------------------------------------------------------------------------

--------------------- Day 5 -----------------------------------
24) ORDER BY :    This is used to sort the records.
-- ASC  -> It sorts in ascending order(by default)
   SELECT * FROM table_name ORDER BY column_name ASC;
   SELECT * FROM emp ORDER BY emp_name ASC;
   NOte:- Writing ASC in query is optional.

-- DESC -> It sorts in descending Order.
   SELECT * FROM table_name ORDER BY column_name DESC;
   SELECT * FROM emp ORDER BY emp_name DESC;

****************************************
      KEY CONSTRAINTS
****************************************
25) NOT NULL : By default,a atable's column can hold NULL values.
      The NOT NULL constraint enforces a field to always contain a value.
      This means you cannot insert a new record,or update a record without adding
      a value to this field.

-- CREATE TABLE student
   (
   Name varchar(30),
   Roll integer(5),
   Mobile_no integer(10) NOT NULL
   );

26) UNIQUE KEY : The UNIQUE constraint uniquely identifies each record in a
database table.
      There can be many UNIQUE constraints per table.
      A UNIQUE Key column can conatins NULL values.

-- CREATE TABLE student
   (
   Name varchar(30),
   Roll integer(5) UNIQUE KEY,
   Mobile_no integer(10)
   );

27) PRIMARY KEY : The Primary Key constraint uniquely identifies each record in a
database table.
      Primary keys must contain unique value.
      Primary key column cannot contain NULL values.
      Most tables should have a primary key
      Each table can have only 1 primary key.

-- CREATE TABLE student
   (
   Roll integer(5) NOT NULL PRIMARY KEY,
   Name varchar(30),
   Mobile_no integer(10)
   );

28) AUTO INCREMENT : Auto Increment is used to generatean unique number, when a new
      record is inserted into a table.

-- CREATE TABLE student
   (
   Roll int NOT NULL PRIMARY KEY AUTO_INCREMENT,
   Name varchar(30),
   Mobile_no integer(10)
   );

29) AUTO INCREMENT FROM PERTICULAR NUMBER
-- ALTER TABLE table_name AUTO_INCREMENT=10;(auto increment starts from 10)


30) AUTO increment without specifying column name
-- INSERT INTO student VALUES(NULL,'RAM');


-----------------------------------------------------------------------------------
------------------------------

----------------------- DAY 6 ----------------------------------

31) Aliases : Aliases are used to temporarily rename a table name or a column name.

a) For Table
-- SELECT column_name FROM table_name AS alias_name;
Ex:
-- SELECT name FROM student AS widyarthi;

b) For Column
-- SELECT coulumn_name AS alias_name FROM table_name;
Ex:
SELECT name AS student_name FROM student;
SELECT name AS "Student Name" FROM student;


32) Arithmetic Operators : *,/,+,-
-- SELECT column_name,column Operaot value FROM table_name;
Ex:
-- SELECT name,cost,cost+100 FROM items_tab;
SELECT name,cost,cost+100 AS "NEW COST" FROM items_tab;
```

```
33) SELECT DISTINCT : The SELECT DISTINCT statement is used to display only
distinct (different) values.
-- SELECT DISTINCT column_name FROM table_name;
EX:
-- SELECT DISTINCT name FFROM student;

34) ALTER TABLE : This commmand is used to Add/Change/Modify/Drop existing
structure of the table.

a) ADD column : When a new column is to be added to the table structure without
constraints.

(ADD single column)
-- Alter TABLE table_name ADD COLUMN column_name datatype(size);

(Adding multiple columns)
-- Alter TABLE table_name
   ADD COLUMN column_name datatype(size),
   ADD COLUMN column_name datatype(size);

(ADD column by POSITION)
-- ALTER TABLE table_name ADD COLUMN column_name datatype(size);(LAST by default)
-- ALTER TABLE table_name ADD COLUMN column_name datatype(size) FIRST;
-- ALTER TABLE table_name ADD COLUMN column_name datatype(size) AFTER column_name;

(ADD column with constraints)
-- ALTER TABLE table_name
   ADD COLUMN column_name datatype(size) Constraint_name,
   ADD constraint_name column_name;

(ADD column with constraints with position)
-- ALTER TABLE table_name
   ADD COLUMN column_name datatype(size) Constraint_name AFTER/FIRST,
   ADD constraint_name column_name;

b) Change Column : This is used to change name and data type of an existing column.
-- ALTER TABLE tbale_name CHANGE COLUMN old_column_name new_column_name
new_data_type(size);


(change more than one column)
-- ALTER tABLE table_name CHANGE COLUMN old_column_name new_column_name
new_data_type(size),
CHANGE COLUMN old_column_name new_column_name new_data_type(size);


(change column with Constraints)
-- ALTER TABLE table_name CHANGE COLUMN old_column_name new_column_name
new_data_type(size) Constrint_name;


c) MODIFY COLUMN : This is used to modify size of the data type or the datatype
itself of ana existing column without changing column name.

-- ALTER TABLE table_name MODIFY COLUMN column_name datatype(size);
```

d) DROP COLUMN : Whena column need to delete
-- ALTER TABLE table_name DROP COLUMN column_name;
EX:
-- ALTER TABLE student DROP COLUMN name;

(WHEN removing constraints from a column.)
-- ALTER TABLE table_name DROP constraint_name column_name;
EX:
-- ALTER TABLE my_tab DROP UNIQUE KEY(roll);(this will give an error like this)
Note:- Get the key_name
-- SHOW INDEX from table_name;
tha use
-- ALTER TABLE test DROP INDEX id;

-------------------- Day 7 --------------------------------

35) DROP TABLE : This command is used to delete/remove table from the database
-- DROP TABLE table_name;

36) TRUNCATE TABLE : When we only want to delte the data inside the table, and not the table itself.
-- TRUNCATE TABLE table_name;

37) RENAME TABLE : This command is used to rename one or more able.
-- RENAME TABLE old_table_name to new_table_name;

38) ALTER DATABASE :
a) Alter database enables you to change the overall characterstics of a database.
b) These characterstics are stored in the db.opt file in the database directory.
c) To use ALTER DATABASE, you need the ALTER privilege on the database.
d) ALTER SCHEMA is a synonym for ALTER DATABASE.
The database name can be omitted from the syntax,in which case the statement appplies to the default database.

-- ALTER dATABASE database_name;
Ex:
ALTER DATABSE db;

39) DROP DATABASE : The DROP DATABASE statement is used to delete a database.
-- DROP DATABASE database_name;

40) SHOW COLUMNS : It shows all the columns of table and their data type along with any other column specific details. It is just like DESC table_name.
-- SHOW COLUMNS FROM table_name;

--------------------------- Day 8 ----------------------------------
41) SHOW CREATE DATABASE : It shows commands which you have written while creating your DATABASE.
-- SHOW CREATE DATABASE database_name;

```
Ex:
-- SHOW CREATE DATABASE my_db;

42) SHOW CREATE TABLE : It shows commands which you havewritten while creating your
table.
-- SHOW CREATE TABLE table_name;


************************************************
------------- update table ---------------------
*************************************************
43) UPDATE : The UPDATE statement is uded to update existing records in a table.
-- UPDATE table_name
   SET column1=value1,column2=value2,.....
   WHERE some_column = some_value;
EX:
-- UPDATE emp
   SET emp_name='AKASH',salary=35000
   WHERE emp_id = 101;

NOTE : WHERE is necessary otherwise all the records will be replace with given
value.

(MULTIPLE UPDATES WITH CASE):

44) UPDATE with CASE
-- UPDATE table_name
   SET new_column=
   CASE
       WHEN column_name1 = some_value THEN new1
       WHEN column_name2 = some_value2 THEN new_value2
       ElSE new_value3
   END;
EX:
-- UPDATE student
   SET Result =
   CASE
       WHEN mark>=300 THEN 'FIRST'
       WHEN mark<=300 AND mark>=250 THEN 'SECOND'
       WHEN mark<250 AND marks>=150 THEN 'THIRD'
       ELSE 'FAIL'
   END;


45) DELETE : The DELETE statement is used to delete records in a table.

a) DELETE a Specific Record
-- DELETE FROM table_name WHERE some_column=sone_value;
EX:
-- DELETE FROM emp WHERE emp_name = 'SONAM' AND emp_id = 105

Note : WHERE is necessary otherwise all records will be delted.

b) DELETE all RECORDS
-- DELETE FROM table_name;
or DELETE * FROM table_name;

EX:
-- DELETE FROM emp;
```

```
or DELETE * FROM emp;

Note : We can not undo it.


46) COPY old table to new table within same database.

-- CREATE TABLE new_table LIKE old_table;
-- INSERT new_table SELECT * FROM old_table;

EX:
   CREATE TABLE teacher LIKE student;
   INSERT teacher SELECT * FROM student;


47) Copy table from one database to another
NOTE : Use the database where you want to copy the lod table;

-- USE new_database
-- CREATE TABLE new_table LIKE old_db.old_table;
-- INSERT new_table SELECT * FROM old_db.old_table;

EX :
-- CREATE TABLE teacher LIKE my_db.student;
-- INSERT teacher SELECT * FROM my_db.student;



-----------------------------------------------------------------------------------
---------

----------------------- DAY 9
------------------------------------------------------


************************************************************
------------------ SQL FUNCTIONS -------------------------
************************************************************
---------------------------------------------------------------------------------

a) MIN(column_name) - Smallest value of the selected column.
b) MAX(column_name) - Largest value of the selected column.
c) SUM(column_name) - The total sum of anumeric column.
d) AVG(column_name) - The average value of a numeric column.
e) SQRT(column_name) - The square root of a numeric column.
f) Round(column_name,decimal) - Function is used to round a numeric field to the
number of decimals specified.
g) COUNT(column_name) - The count(column_name) function returns the number of
values (NULL values will not be counted) of the specified column.
h) UPPER(column_name) OR UCASE(column_name) - Converts the value of a field to
uppercase.
i) LOWER(column_name) or LCASE(column_name) - converts the value of a field to
lowercase.
j) MID(column_name,start,length) or SUBSTRING(column_name,start,length) - function
is used to extract characters from a text field.
k) LENGTH(column_name) - the length of the value in a text field.
l) CONCAT(column_name1,column_name2,....) - It joins two column.
m) REVERSE(column_name) - It reverse the order of letter in string.
n) NOW() - Function returns the current system date and time.
```

```
o) FORMAT(column_name,format_type) - Function is used to format how a field is to
be displayed.

-----------------------------------------------------------------------------------




a) MIN(column_name):
-- SELECT MIN(salary) FROM emp;
-- SELECT MIN(salary) AS newsalary FROM emp;

b) MAX(column_name):
-- SELECT MAX(salary) FROM emp;

c) SUM(column_name):
-- SELECT SUM(salary) AS TOTAL_SALARY FROM emp;

d) AVG(column_name):
-- SELECT AVG(salary) AS AverageSalary FROM emp;

e) SQRT(column_name):
-- SELECT salary, SQRT(salary) FROM emp;


*******************
DECIMAL DATATYPE :
*******************
-- column_name DECIMAL(T,D)
where
T = total digites. Range 1-65 (number of digits)
D = digits after decinmal. Range 0-30 and must not be more than T
EX:
-- price DECIMAL(7,2)

---------------------------------
---- Storage Description---------
---------------------------------
```

No. Of Digits	No. of Bytes
0	0
1-2	1
3-4	2
5-6	3
7-8	4

```
-- CREATE TABLE product( Id INT AUTO_INCREMENT PRIMARY KEY, Pname VARCHAR(40),
PRICE DECIMAL(7,2) NOT NULL);
-- INSERT INTO product (pname,price) VALUES("MOBILE",16999.25),
("COmputer",25000.65);

*******************
--- Zerofill ------
*******************
It will fill 0 automatically according to the need.
```

```
EX:
-- price DECIMAL(10,4) ZEROFILL
=> 250.65
=>000250.6500


f) Round(column_name,decimal):
-- SELECT *, ROUND(price,1) FROM product;


g) COUNT(column_name)
-- count(*) return the number of records in a table.
-- SELECT COUNT(*) FROM table_name;

-- SELECT COUNT(custID) FROM orders;
-- SELECT COUNT(DISTINCT custID) FROM orders;
-- SELECT COUNTcustID) from orders where custID=101;

h) Upper(column_name)
-- SELECT UPPER(emp_name) emp emp;

i) LOWER(column_name)
-- SELECT LOWER(emp_name) FROM emp;

j) MID(column_name,start,length)
-- SELECT MID(city,1,3) AS SHORT_CITY FROM emp;

k) LENGTH(column_name)
-- SELECT city,LENGTH(city) FROM emp;

l) CONCAT(column_name1,column_name2)
-- SELECT emp_name,CONCAT(city,' ',pin) AS New_Adddress FROM emp;

m) REVERSE(column_name)
-- SELECT REVERSE(city) from emp;

n) NOW()
-- SELECT emp_name,salary NOW() AS DateTime FROM emp;

----------------------------------------------------------------------------------
------------------
--------------------- Day 10 --------------------------------

48) Group BY : Used to group items on a specific condition.
-- SELECT emp_name,MIN(salary) FROM emp GROUP BY emp_name;
-- SELECT cus_ID,count(*) FROM customer GROUP BY cus_ID;

49) Having :
-- SELECT emp_name,MIN(salary) FROM emp GROUP BY emp_name HAVING MIN(salary)>25000;


*************************
RELATIONSHIP-------------
*************************
1) One to One   - (HUSBAND & WIFE)(one aadhar number for one person)(one employee
id for one employee)
2) One to Many  - (Father & children)(Customer & order)
3) Many to Many - (authors & books) NOTE:- a third table is used in this known as
```

junction table or merjing table.

```
*************************
NORMALIZATION------------
*************************
```
-- Normalization is the process of minimizing redundancy from the database.

Forms of Normalization:-
a) 1NF
b) 2NF
c) 3NF

a) 1NF :
-- 1) Each attribute has an atomic value
-- 2) We can not create repeating group in 1NF (User and its interest)

b) 2NF:
-- Table must be 1NF
-- Non key fileds must be depend on primary key (we can use composite key in this)

** Composite key : When two fields are combine to create a uniqe key or Primary key.

c) 3NF:
-- Table must be in 2NF.
-- Non key field not depend on any other non key field.

```
----------------------------------------------------------------------------------
-------------------- DAY 11 ------------------------------

***************
FOREIGN KEY----------------
***************
```

--A FOREIGN KEY in one table points to a PRIMAARY KEY in another table.
-- A FOREIGN KEY can have a different name than the primary key it comes from.
-- The primary key used by a foreign key is also known as a parent key. The table where the primary key is from is known as a parent table.
-- The foreign key can be used to make sure that the row in one table have corresponding row in another table.
-- Foreign key value can be null,even though primary key value can't.
-- Foreign key don't have to be unique in fact, they often aren't.

1) Create table using foregin key:
```
-- CREATE TABLE department
   (
   d_id int NOT NULL AUTO_INCREMENT RIMARY KEY,
   D_Name varchar(40),
   E_id int,
   CONSTRAINT employee_Eid_fk FOREIGN KEY(E_id) REFERENCES employee(E_id)
   );
```

Note:- The CONSTRAINTS clause allows to define constrints name for the foreign key constarint.If we omit it, MySQL will generate a name automatically. It is optional.

Note :- The REFERENCES clause specifies the parent table and its columns to which the column in the child table refer. The number of columns in the child table and parent table specified in the FOREIGN KEY and REFERENCES must be the same.

2) How to Find constrant name??
```
--  SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    WHERE TABLE_NAME='department';
```

3) REMOVE FOREIGN KEY FROM TABLE:
a) FIRST FInD THE CONSTARINTS KEY
b) ALTER TABLE department DROP FOREIGN KEY constaint_name;
or ALETER TABLE department DROP CONSTRAINT constraint_name;

4) How to add foreign key in existing table
```
-- ALTER TABLE department ADD CONSTARINT employee_eid_fk FOREIGN KEY(empid)
REFERENCES employee(eid);
```

5) ADD COLUMN WITH FOREIGN KEY
```
-- ALTER TABLE department ADD COLUMN empid int not null, ADD CONSTRAINT
employee_id_fk FOREIGN KEY(empid) REFERENCES empoyee(eid);
```

6) Unable to delete parent table
NOTE:- you have to delete the child table first which contains a foregin key after that you can delete parent table;

7) Unable to delete parent table record
NOTE:- parent table dont refer to any record in child table

-------------------- Day 12 ----------------------------

*) ON DELETE CLAUSE : use with foreign key

a) ON DELETE CASCADE
b) ON DELETE SET NULL
c) ON DELETE NO ACTION
d) ON DELETE RESTRICT

a) ON DELETE CASCADE : Child records are deleted automatically when the parent record deleted while using foreign key.
```
--  CREATE TABLE department
    (
    did int not null auto_increment primary key,
    dname varchar(40),
    empid int not null,
    CONSTRAINT employee_id_fk
    FOREIGN KEY(empid) REFERENCES employee(eid)
    ON DELETE CASCADE
    );
```

b) ON DELETE SET NULL : The foreign key value become null in the child table column

```
instead of compltely deleteing the entire row.
--  CREATE TABLE department
    (
    did int not null auto_increment primary key,
    dname varchar(40),
    empid int not null,
    CONSTRAINT employee_id_fk
    FOREIGN KEY(empid) REFERENCES employee(eid)
    ON DELETE SET NULL
    );
```

c & d) ON DELETE NO ACTION & ON DELETE RESTRICT : Both are same as both stops the column froom deletetion.

```
--  CREATE TABLE department
    (
    did int not null auto_increment primary key,
    dname varchar(40),
    empid int not null,
    CONSTRAINT employee_id_fk
    FOREIGN KEY(empid) REFERENCES employee(eid)
    ON DELETE RESTRICT
    );
```

**) ON UPDATE CLAUSE:
a) ON UPDATE CASCADE
b) ON UPDATE SET NULL
c) ON UPDATE NO ACTION
d) On UPDATE RESTRICT

a) ON UPDATE CASCADE : changes in primary key of parent table reflects effect on the foreign key of child table.

```
--  CREATE TABLE department
    (
    did int not null auto_increment primary key,
    dname varchar(40),
    empid int not null,
    CONSTRAINT employee_id_fk
    FOREIGN KEY(empid) REFERENCES employee(eid)
    ON UPDATE CASCADE
    );
```

b) ON UPDATE SET NULL : changes in primary key of parent table reflects effect on the foreign key of child table all the affected column become NULL.

```
--  CREATE TABLE department
    (
    did int not null auto_increment primary key,
    dname varchar(40),
    empid int,
    CONSTRAINT employee_id_fk
    FOREIGN KEY(empid) REFERENCES employee(eid)
    ON UPDATE CASCADE
    );
```

c & d) ON UPDATE NO ACTION AND ON UPDATE RESTRICTS: No updation directly from parent table.

```
--  CREATE TABLE department
```

```
    (
    did int not null auto_increment primary key,
    dname varchar(40),
    empid int not null,
    CONSTRAINT employee_id_fk
    FOREIGN KEY(empid) REFERENCES employee(eid)
    ON UPDATE RESTRICT
    );
```

**) COMPOSITE KEY : When a table has no primary key to uniquely identify a record than we can combine two or more columns to form a composite key to uniquely identify a record.

```
-- CREATE TABLE course
   (
   CourseCode varchar(40),
   Date date,
   Cname Varchar(40),
   Seat int,
   Remain int,
   Room int,
   Rcapa int,
   PRIMARY KEY(coursecode,date)
   );
```

**) SHORT HAND NOTATIONS : table_name.column_name
--

```
----------------------------------------------------------------------------------

----------------- DAY 13 ---------------------------------------

*************************************
Table Joins--------------------------
*************************************

***********************************************************************************
```

a) CROSS JOIN / CARTESIAN JOIN / CARTESIAN PRODUCT / CROSS PRODUCT:
-- The cross Join returns every row from one table crossed with every row from the second table.

```
        Employee             Department
-------------------------       --------------------
| Empid | Name  | City  |       | did | DepName    |
-------------------------       --------------------
|   1   | Rahul | Delhi |       | 101 | IT         |
|   2   | Krish | Kol   |       | 102 | HR         |
|   3   | JAY   | Mum   |       | 103 | Admin      |
-------------------------       --------------------

-- SELECT * FROM Employee CROSS JOIN Department;
-- SELECT Name,DepName FROM Employee CROSS JOIN Department;
-- SELECT * FROM Employee,Department;
-- SELECT * FROM Department,Employee;
```

Note:- The group name will be created on the baseis of table name written on the last.

```
---- SELECT * FROM Department,Employee;
---- Groupname created on the basis of Employee table
output :

---------------------------------------------
| Name  |  Department   |
--------------------------------------------
| Rahul | IT            |------>
| Rahul | HR            |---------> GROUP name 1 Rahul
| Rahul | ADMIN         |------>
---------------------------------------------
| Krish | IT            |------>
| Krish | HR            |---------> GROUP name 2 Krish
| Krish | ADMIN         |------>
---------------------------------------------
| Jay   | IT            |------>
| Jay   | HR            |----------> GROUP name 3 Jay
| Jay   | ADMIN         |------>
---------------------------------------------
```

```
**********************************************************************************
```

b) INNER JOIN : An INNER JOIN is a CROSS JOIN with some result rows removed by a condition in the query.

1) EQUIJOIN
2) NON-EQUIJOIN
3) NATTURAL JOIN

1) EQUIJOIN :

```
-- SELECT column_name FROM table1 INNER JOIN table2 ON
table1.column_name=table2.column_name;
```

EX:

```
        Employee                Department
-------------------------       ----------------------------
| Empid | Name  | City  |       | did | DepName    | Empid |
-------------------------       ----------------------------
|   1   | Rahul | Delhi |       | 101 | IT         | 3     |
|   2   | Krish | Kol   |       | 102 | HR         | 1     |
|   3   | JAY   | Mum   |       | 103 | Admin      | 2     |
-------------------------       ----------------------------
```

```
-- SELECT Employee.Name,Department.DepName FROM Employee INNER JOIN Department on
Employee.Empid = Department.Empid
```

OUTPUT When:

```
Employee Inner Join Department
-------------------
| Name  | DepName |
-------------------
| Jay   | IT      |
| Rahul | HR      |
| Krish | Admin   |
-------------------

Department INNER JOIN EMPLOYEE
-------------------
| Name  | DepName |
-------------------
| Rahul | HR      |
| Krish | ADMIN   |
| Jay   | IT      |
-------------------
```

Note : matching will take place on the basis of the table name written on the last.

2) NON-EQUIJOIN (symbol = '<>'):
-- SELECT column_name FROM table1 INNER JOIN table2 ON table1.column_name <> table2.column_name;

EX:

```
        Emp                        Dep
-------------------------       ----------------------------
| Empid | Name  | City  |       | did | DepName    | Empid |
-------------------------       ----------------------------
|   1   | Rahul | Delhi |       | 101 | IT         | 3     |
|   2   | Krish | Kol   |       | 102 | HR         | 1     |
|   3   | JAY   | Mum   |       | 103 | Admin      | 2     |
-------------------------       ----------------------------
```

-- SELECT Emp.Name,DEp.DepName FROM Dep INNER JOIN Emp ON Emp.Empid <> Dep.Empid;

OUTPUT :

```
-------------------
| Name  | DepName |
-------------------
| Rahul | IT      |
| Rahul | Admin   |
-------------------
| Krish | IT      |
| Krish | HR      |
-------------------
| JAY   | HR      |
| JAY   | Admin   |
-------------------
```

3) NATURAL JOIN :
-- SELECT column_name FROM table1 NATURAL JOIN table2;

Note: IN case ofnatural join it is mandetory that the name or primary key column and name of foreign key column must be same.

EX:

```
        Emp                        Dep
-------------------------       ----------------------------
```

```
| Empid | Name  | City  |       | did | DepName    | Empid |
-------------------------       ----------------------------
|   1   | Rahul | Delhi |       | 101 | IT         | 3     |
|   2   | Krish | Kol   |       | 102 | HR         | 1     |
|   3   | JAY   | Mum   |       | 103 | Admin      | 2     |
-------------------------       ----------------------------
```

```
-- SELECT Emp.Name,Dep.DepName FROM dep NATURAL JOIN Emp
```

OUTPUT :

```
-------------------
| Name  | DepName |
-------------------
| Rahul | HR      |
| Krish | ADMIN   |
| Jay   | IT      |
-------------------
```

****

c) OUTER JOINS : An outer joins returns all rows from one of the tables, along with matching information from another table.

1) LEFT OUTER JOIN / LEFT JOIN
2) RIGHT OUTER JOIN / RIGHT JOIN
3) FULL OUTER JOIN / FULL JOIN

1) LEFT OUTER JOIN / LEFT JOIN :
-- The LEFT JOIN keyword returns all rows from the left table,with the matching row in the right table. The result is NULL in the right side when there is no match.

-- In a LEFT JOIN the table that comes before the join is the left table, and the table that comes after the join is the right table.

```
-- SELECT column_name FROM table1 LEFT JOIN table2 on column_name = column_name;
```

EX:

```
        Emp                       Dep
-------------------------       ----------------------------
| Empid | Name  | City  |       | did | DepName    | Empid |
-------------------------       ----------------------------
|   1   | Rahul | Delhi |       | 101 | IT         | 3     |
|   2   | Krish | Kol   |       | 102 | HR         | 1     |
|   3   | JAY   | Mum   |       | 103 | Admin      | 2     |
|   4   | Sonam | Hyd   |       | 104 | IT         | 1     |
|   5   | Mona  | Patna |       ----------------------------
-------------------------
```

```
-- SELECT Emp.name , Dep.DepName FROM Emp LEFT JOIN Dep ON Emp.EMPID=Dep.Empid;
```

OUTPUT :

```
-------------------
```

```
| Name  | DepName |
-------------------
| Jay   | IT      |
| Rahul | HR      |
| Krish | ADMIN   |
| Rahul | IT      |
| Sonam | NULL    |
| Mona  | NULL    |
-------------------
```

2) RIGHT OUTER JOIN / RIGHT JOIN :
-- The RIGHT JOIN keyword return all rows from the right table, with trhe matching rows in the left table. The result is NULL in the left side when there is no match.

-- In a RIGHT OUTER JOIN the table that comes before the join is the Right table, and the tablethat comes after the join is the Left table.

EX:

```
        Emp                       Dep
-------------------------       ----------------------------
| Empid | Name  | City  |       | did | DepName    | Empid |
-------------------------       ----------------------------
|   1   | Rahul | Delhi |       | 101 | IT         | 3     |
|   2   | Krish | Kol   |       | 102 | HR         | 1     |
|   3   | JAY   | Mum   |       | 103 | Admin      | 2     |
|   4   | Sonam | Hyd   |       | 104 | IT         | 1     |
|   5   | Mona  | Patna |       ----------------------------
-------------------------
```

```
-- SELECT Emp.NAME,Dep.DepName FROM Emp RIGHT JOIN dep ON emp.empid = dep.empid;
```

```
-------------------
| Name  | DepName |
-------------------
| Jay   | IT      |
| Rahul | HR      |
| Krish | ADMIN   |
| Rahul | IT      |
-------------------
```

3) FUll OUTER JOIN / FULL JOIN :

Note : MYSQL does not support FULL JOIN

--The FULL OUTER JOIN returns all rows from the left table and from the right table.
--The FULL OUTER JOIN combines the result of both LEFT and RIGHT joins.

```
-- SELECT column_name FROM table1 FULL JOIN table2 ON column_name=column_name;
```

****

c) SELF JOIN : SELF JOIN is a table join to itself.

Qus : Why do we need SELF JOIN ?
Ans : When we have to find the data from the same table or when we have to do sub query than we use self joins.
When Hierarchical Problems occurs than we use self join.

EX:

-- From the given below table find the employee under the managers.

```
       empman
-----------------------------
| Empid | Name  | ManID     |
-----------------------------
|  1    | Rahul | 3         |
|  2    | Jay   | 3         |
|  3    | Sonam | 4         |
|  4    | Kunal | 5         |
|  5    | Ram   | 6         |
|  6    | Rani  | NULL      |
|  7    | Veeru | 6         |
-----------------------------
```

```
-- SELECT e.name Name,m.name Manager FROM empman e INNER JOIN empman m ON
e.manid=m.empid;
```

OUTPUT :

```
-------------------
| Name  | Manager |
-------------------
| Rahul | Sonam   |
| Jay   | Sonam   |
| Sonam | Kunal   |
| Kunal | Ram     |
| Ram   | Rani    |
| Veeru | Rani    |
-------------------
```

```
-- SELECT e.name Name,m.name Manager FROM empman e INNER JOIN empman m ON
m.empid=e.manid;
```

OUTPUT :

```
-------------------
| Name  | Manager |
-------------------
| Rahul | Sonam   |
| Jay   | Sonam   |
| Sonam | Kunal   |
| Kunal | Ram     |
| Ram   | Rani    |
| Veeru | Rani    |
-------------------
```

```
-- SELECT e.name Name,m.name Manager FROM empman e INNER JOIN empman m ON
e.empid=m.manid;
```

```
OUTPUT :
-------------------
| Name  | Manager |
-------------------
| Sonam | Rahul   |
| Sonam | Jay     |
| Kunal | Sonam   |
| Ram   | Kunal   |
| Rani  | Ram     |
| Rani  | Veeru    |
-------------------
```

Note : Its Better to use Left join instead of inner join.

Ex :

```
-- SELECT e.name Name,m.name Manager FROM empman e LEFT JOIN empman m ON
e.manid=m.empid;
```

```
OUTPUT :
-------------------
| Name  | Manager |
-------------------
| Rahul | Sonam   |
| Jay   | Sonam   |
| Sonam | Kunal   |
| Kunal | Ram     |
| Ram   | Rani    |
| Rani  | NULL    |
| Veeru | Rani    |
-------------------
```

Note : In case of self join Please create separate alias of table so that you can get proper result.

Q) How to replace NULL with other word??
--

1) IFNULL(exp1,exp2)

```
-- SELECT IFNULL('AKASH','MATHUR') AS EXAMPLE;
```

```
OUTPUT:
-----------
| EXAMPLE |
-----------
| AKASH   |
-----------
```

```
-- SELECT IFNULL(NULL,'MATHUR') AS EXAMPLE;
```

```
OUTPUT:
-----------
| EXAMPLE |
-----------
| MATHUR  |
-----------
```

```
-- If exp1 is not NULL then IFNULL() returns eexp1 otherwise it returns exp2.

-- SELECT e.name Name,IFNULL(m.name,'No Manager') AS Manager FROM empman e LEFT
JOIN empman m ON e.manid=m.empid;
OUTPUT :
-------------------------
| Name  | Manager       |
-------------------------
| Rahul | Sonam         |
| Jay   | Sonam         |
| Sonam | Kunal         |
| Kunal | Ram           |
| Ram   | Rani          |
| Rani  | No Manager    |
| Veeru | Rani          |
-------------------------

2) COALESCE(exp1,exp2...expn);
-- SELECT COALESCE(NULL,'MATHUR') AS EXAMPLE;



**********************************************************************************


******************************
UNION ------------------------
******************************

-- A UNION combines the results of two or more queries into one table.

Rules:-
-- The number of column in each SELECT statement must match.
-- The data types in the columns should be same.
-- The columns in each SELECT statement must be in same order.
-- Must have the same expressions and aggregate function in each SELECT statement
-- The UNION operator selects only distinct values by default and It doesn't select
duplicate values.

EX : -
-- SELECT name FROM student UNION SELECT name FROM Employee;


        employee                        student
-------------------------       ----------------------------
| Empid | Name  | City  |       | stuid |   name    | city  |
-------------------------       ----------------------------
|   1   | Rahul | Delhi |       | 1     | Rahul     | Delhi |
|   2   | Krish | Kol   |       | 2     | Jay       | Kol   |
|   3   | JAY   | Mum   |       | 3     | Sonam     | Delhi |
-------------------------       | 4     | Kunal     | Hyd   |
                                | 5     | Ram       | Delhi |
                                | 6     | Rani      | Patna |
                                | 7     | Veeru     | Kol   |
                                ----------------------------

OUTPUT:
```

```
---------
| name  |
---------
| Rahul |
| Jay   |
| Sonam |
| kunal |
| Ram   |
| Rani  |
| Veeru |
| krish |
---------
```

**) UNION ALL : UNION ALL returns every match whether it is duplicate or dustict ones.

```
-- SELECT name from student UNION ALL SELECT name FROM employee;
```

****

****
INSERSECT and EXCEPT-------
****
NOTE: MYSQL NOT SUPPORT INTERSECT and EXCEPT KEYWORD;

1) INTERSECT : INTERSECT returns those columns that are in the first query and also in the second query.
```
-- SELECT name FROM student INTERSECT SELECT name FROM employee;
```

2) EXCEPT : EXCEPT return only those columns that are in the first query but not in the second query.
```
-- SELECT name FROM student EXCEPT SELECT name FROM EMPLOYEE;
```

**************************** THE END **