SQL Notes: Beginner to Advanced

This document contains SQL notes covering a range of topics from beginner-level to advanced concepts. The content is structured for easy understanding and covers important commands with detailed comments.

Prepared by: Ashutosh Rai

Email: ashutoshrai70@gmail.com

GitHub: https://github.com/ashutoshrai70

Table of Contents

```
-- How to see all the database present in our sql

show databases;

-- How to create schemas

create database school;

-- how to work on any particular database

use school;

-- how to delete/drop database

drop database a;

-- how to create table

create table student
(
roll_no int,
name varchar(100),
city varchar(100)
);

-- how to see that the table has created
-- this can be done using desc command. DESC describe the table
```

```
desc table student;

-- how to insert data into table

-- Method 1

insert into student (roll_no,name,city)

values(1234, "Rahul", "Varanasi");

-- Method 2

insert into student

values (1,'Shubham','DDU');

-- inserting multiple data at a time

insert into student

values

(2,'Anoop','Ranchi'),

(4,'Nisar','VNS'),

(5,'Ashutosh','VNS');

-- Reading data from a table

-- To read all the data together

select * from student;

-- To read a particular column from a table

select city from student;
```

```
-- using where command to see a particular set of data

select * from student
where roll_no=4;

select name from student where roll_no=4;

-- Modify/update data from table

SET SQL_SAFE_UPDATES = 0;

update student
set name='Aman', roll_no=6
where name='Rahul';

SET SQL_SAFE_UPDATES = 1;

-- how to delete data from table
SET SQL_SAFE_UPDATES = 0;

delete from student
where name='Aman';

SET SQL_SAFE_UPDATES = 1;

select * from student;
```

```
-- how to drop/delete the whole table

drop table student;

show tables;



-- using new database

--create database college;

use college;
/*
CREATE TABLE customer
(
    ID int,
    Name varchar(50),
    Age int
);
*/

show tables;
desc customer;
-- where we find Null=Yes this means there we can fill any value if we want. In other word where null
is Yes there insert is allowed

insert into customer(ID,Name,Age)
```

```
values(1234,'Rahul',31);

select * from customer
where ID=1234;

-- Not Null command is used while creating any table so, that non of the data goes missing while entering the data

create table Learning_table
(
roll_no int not null,
name varchar(20) not null,
contact int not null
);

desc learning_table;

-- setting a default value for a column

create table l1
(
roll_no int not null,
name varchar(50) default('Not Known') not null,
contact int not null
);

desc l1;
```

```
insert into l1(roll_no,contact)
values (1,6754),(2,7654),(3,4321);

select* from l1;

-------------------------------- PRIMARY KEY----------------------------------
-- PRIMARY KEY constraint a uniquely identify each record in a table.
-- PRIMARY KEY must contain UNIQUE values and can not contain NULL values.
-- A table can have one ONE primary key.

create table l2
(
roll_no int primary key,
name varchar(100),
city varchar(100) default 'Varanasi'
);

desc l2;

insert into l2(roll_no,name)
values
(2,'Anoop'),
(4,'Nisar'),
(5,'Ashutosh');

select * from l2;
```

```
---------- AUTO INCRIMENT----------

CREATE TABLE L3
(
roll_no int primary key auto_increment,
name varchar(50),
city varchar(50) default 'Varanasi'
);

desc l3;

insert into l3(name)
values('Ashutosh'),('Nisar'),('Anoop');

select * from l3;

insert into l3(roll_no,name)
values(101,'Aman');

select * from l3;

insert into l3(name)
values('Shubham'),('Adarsh');

select * from l3;

-- ALIAS it is the way of changing the name of the column to make it more readable
```

```
select roll_no as 'ID', city as 'Place' from l3;
-- it does not make any change in original table name
select * from l3;

---------- Exercise1 ----------
/*
create database bank_db;

use bank_db;

create table employees
(
emp_id int primary key auto_increment,
name varchar(50) not null,
desig varchar(50) not null default 'Probation',
dept varchar(20)
);

insert into employees values(101,'Raju','Manager','Loan');

insert into employees(name,desig,dept)
values
('Sham','Cashier','Cash'),('Paul','Associate','Loan'),('Alex','Accountant','Account'),('Victor','Associate','
Deposit');

select * from employees;
```

```
select emp_id,name from employees;
```

---------- Exercise2 ----------

```
select * from employees
where dept='Loan';
```

or we can do this here

```
select * from employees
limit  2;
```

```
select * from employees
where emp_id=101;
```

or we can do this here

```
select * from employees
limit  1;
```

```
select emp_id,name  from employees
where emp_id=101;
```

or we can do this

```
select emp_id,name  from employees
```

```
limit  1;


SET SQL_SAFE_UPDATES = 0;
update employees
set dept = 'IT'
where name ='Paul';
SET SQL_SAFE_UPDATES = 1;


select * from employees;


SET SQL_SAFE_UPDATES = 0;
delete from employees
where emp_id=102;
SET SQL_SAFE_UPDATES = 1;


select * from employees;
*/
show databases;
---------------------------- String Functions ----------------------------
---------------- CONCAT ------------
/* Syntax:
concat(col_1,col_2)
OR
concat(first_word,sec_word.....)
*/
select  concat('Hey','Buddy!');
select concat('Hey',' ','Buddy!',' ','hello');
```

```
create database bank_data;

use bank_data;

create table employees
(
emp_id int primary key auto_increment,
fname varchar(50) not null,
lname varchar(50) not null,
desig varchar(50) not null default 'Probation',
dept varchar(20) not null
);

insert into employees values(101,'Raju','Rastogi','Manager','Loan');

insert into employees(fname,lname,desig,dept)
values
('Sham','Mohan','Cashier','Cash'),('Paul','Philip','Associate','Loan'),('Alex','Watt','Accountant','Account'
),('Victor','Apte','Associate','Deposit');

select * from employees;

select emp_id, concat(fname,' ',lname) as Full_name from employees;

select emp_id, concat(fname,' ','ABCD') as NEW_name from employees;
```

```
/*
Concat_ws here ws indicates to seprater
Syntax:
concat(seprator,col_1,col_2)
*/
select  concat_ws('-','Hey','Buddy!');
select concat_ws('-','Hey',' ','Buddy!',' ','hello');

select emp_id,concat_ws('_',fname,lname) as Name from employees;

---------- SUBSTRING ----------
/*
This works as indexing in python
Syntax:
substring('word',start_position,end_position)
*/

select substring('Hey Buddy!',1,4);
select substring('Hey Buddy!',5);
select substring('Hey Buddy!',-4);

select substring(emp_id,2) as Emp_id,fname from employees;

---------- REPLACE ----------
/*
Syntax:
replace('str','from_str','to_str')
```

```
*/

select replace('Hey Buddy!','Hey','Hello');

select replace('ABCDEFGH','FGH','XYZ');

select replace(emp_id,10,100) as New_ID ,fname from employees;

select replace(emp_id,10,'EMP') as New_ID,fname from employees;

---------- REVERSE ----------

/*

Syntax:

reverse(hello)

*/

select reverse('HELLO');

select reverse(emp_id) as reverse_id,fname from employees;

---------- UPPER & LOWER ----------

/*

Syntax:

upper(hello)

OR it can be also written as ucase

ucase('hello')

*/
```

```
select upper('hello');

select ucase('abcdefgh');

select lower('HeLLO');

SELECT lcase('ABCDEFGH');


select emp_id, upper(fname) as name from employees;


---------- CHAR_LENGTH ----------

/*

This works as indexing in python

Syntax:

CHAR_LENGTH('hello')

*/

select char_length('hello');

select char_length('hello buddy!');


select emp_id,fname,char_length(dept) as length from employees;


select * from employees

where char_length(fname)>5;


---------- LEFT RIGHT REPEAT TRIM INSERT ----------

/*

Syntax:

INSERT(original_string, start_position, length_to_replace, string_to_insert)


select left('word',upto_position)
```

```
select right('word',upto_position)

select repeat(to_repeat,number_of_times)

TRIM([leading | trailing | both] [removal_string] FROM string)
leading: Removes characters from the beginning (left) of the string.
trailing: Removes characters from the end (right) of the string.
both: Removes characters from both the beginning and the end (default behavior).
removal_string: The character(s) you want to remove. If omitted, the default is whitespace (spaces).
*/

select insert('Hey whatsup',5,0,'Raju ');

select left('abcdefghijkl',5);
select right('abcdefghijkl',5);

select repeat('hahahaha ',10);

SELECT TRIM('  Hello World  '); -- by default it is BOTH so it will trim from both the end

SELECT TRIM('*' FROM '*Hello World*');

SELECT TRIM(LEADING ' ' FROM '   Hello World');
SELECT TRIM(TRAILING '!' FROM 'Hello World!!!');

---------- Exercise3 ----------
/*
```

```
select concat_ws(':',emp_id,fname,lname,desig,dept) from employees
where emp_id=101;

select concat_ws(':',emp_id,concat(fname,' ',lname),desig,dept) from employees
where emp_id=101;

select concat_ws(':',emp_id,fname,lname,upper(desig),dept) from employees
where emp_id=101;

select concat(substring(dept,1,1),emp_id),fname from employees
where emp_id=101 or emp_id=102;
*/

---------------------------- DISTINCT ----------------------------
/*
This is used to get the unique values present in columns
Syntax:
select distinct (col_name) from table_name;
*/

select distinct dept from employees;

---------------------------- ORDER BY ----------------------------
/*
This is used to get data sorted
Syntax:
select  col_name/* from table_name order by col_name;
```

```
*/

select * from employees
order by fname;

-- To do sorting in reverse manner that is in decending manner

select * from employees
order by fname desc;

select dept,fname from employees
order by dept,fname;
-- Here order by has been used over 2 columns 'dept' and 'fname' then sorting will be performed on
dept first and suppose tow or more
-- persons are having same department so their sorting will be done on the basis of fname.

---------------------------- LIKE Keyword ----------------------------
/*
This is used to get data sorted
Syntax:
select  col_name/* from table_name
where col_name like "%word_looking_for%";
*/

select * from employees
where desig like "%asso%";
-- this is not case senstive which means '%asso%' and '%ASSO%' both will give same result
```

```
select * from employees
where fname like "____";
-- this returns data of those employees who char_length is 4

select * from employees
where fname like 'R___';
-- This returns all the detals of each employee whose fname starts with letter r


---- AlterTable------
alter table employees
add column
salary int not null
default 25000;

select * from employees;

SET SQL_SAFE_UPDATES = 0;

update employees
set salary=37000
where emp_id=101;

update employees
set salary=42000
where emp_id=102;
```

```
update employees
set salary=50000
where emp_id=103;

update employees
set salary=20000
where emp_id=104;

SET SQL_SAFE_UPDATES = 1;

select * from employees;

---------------------------- LIMIT ----------------------------
/*
This is used to get limited numbers of rows from the table
Syntax:
select  col_name/* from table_name
limit number_of_lines_you_need;
*/

select * from employees
limit 3;

-- Limit can even be used to see to get data from a range of rows
-- Suppose we want see the data from row 2 to 4
```

```
select * from employees
limit 2, 2;

select * from employees
order by salary desc
limit 1;

---------- COUNT ----------
/*
This is used to get numbers of records from the table
Syntax:
select  count(col_name/*) from table_name;
*/

select count(*) from employees;

select count(fname) from employees;

select count(distinct dept) from employees;

select count(*) from employees
where desig='manager';

select * from employees;

---------- Exercise4 ----------
/*
```

```
select distinct dept from employees;

select * from employees
order by salary desc;

select * from employees
limit 3;

select * from employees
where fname like 'a%';

select * from employees
where char_length(lname)=4;
*/

---------- GROUP BY ----------
/*
This is used to group the data on the bassis of common element present in the table
Syntax:
select  col_name from table_name
group by col_name;
*/

select dept from employees
group by dept;

select dept,count(*) as Number_Of_Employee from employees
```

```
group by dept;

select desig , count(*) from employees
group by desig;

---------- MIN & MAX ----------
/*
Syntax:
select  max(col_name) from table_name
select  min(col_name) from table_name
group by col_name;
*/

select max(salary) from employees;
select min(salary) from employees;

select fname from employees
where
salary=(select max(salary) from employees);
-- This is an example of SUB QUERY----------
-- In SUB QUERY the query present in the bracket execute first then the main query get executed.

select max(fname) from employees;
select min(fname) from employees;

---------- SUM & AVG ----------
/*
```

```
Syntax:
select  sum(col_name) from table_name
select  avg(col_name) from table_name
group by col_name;
*/

select sum(salary) from employees;
select avg(salary) from employees;

select dept, sum(salary) from employees
group by dept;

select dept,count(dept),sum(salary) from employees
group by dept;

---------- Exercise5 ----------
/*
select count(emp_id) from employees;

select dept, count(emp_id) as 'Number of employee' from employees
group by dept;

select min(salary) from employees;

select * from employees
where salary=(select max(salary) from employees);
```

```
select sum(salary) from employees
where dept='loan';

select dept, avg(salary) from employees
group by dept;
*/

---------- DECIMAL ----------

/*
Syntax:
decimal(number_of_digit , number_of_digit_after_decimal)
decimal(5,2)
5 indicates that there can be 3 or less than that digit before decimal 1 (.) and 1/2 digits after decimal
or more it will round off to 2 digit
*/
create table num
(
price decimal(5,2)
);

desc num;

insert into num values(135.65),(256.32),(12.503),(46.98),(23.70);

select * from num;

---------- FLOAT & DOUBLE ----------
```

```
/*
Both FLOAT & DOUBLE are another way to store decimal digits
FLOAT works fine upto 7 digits and takes 4 bytes of memory
DOUBLE works fine upto 15 digits and take 8 bytes of memory
Syntax:

*/

create table num1
(
float_value float,
double_value double
);

desc num1;

insert into num1 values(123.456,
123.456),(123.123456789,123.123456789),(123.123456789,123.12345678900987654321);
-- This is to check the limitations of FLOAT & DOUBLE

select * from num1;

---------------------------- Datatype DATE & TIME ----------------------------
/*
Format:
DATE: yyyy-mm-dd
TIME: HH:MM:SS
```

```
DATETIME: 'yyyy-mm-dd HH:MM:SS'
*/

create table person
(
jd date,
jt time,
jdt datetime
);

desc person;

insert into person values('2022-04-11', '23:00:00', '2023-04-10 20;30:00');

select * from person;

insert into person values('2020:01:20', '10:20:40', '2011:10:16 22:00:10');

select * from person;

 ---------- CURDATE CURTIME NOW ----------
/* Formate:
CURDATE() - yyyy-mm-dd
This gives us the current date
CURTIME() - HH:MM:SS
This gives us the current time
NOW() - yyyy-mm-dd HH:MM:SS
```

```
This gives us the current date and time both
*/

select curdate();
select curtime();
select now();

desc person;

insert into person values(curdate(),curtime(),now());

select * from person;

---------- function for DATE TIME  ----------
/*
Syntax:
DAYNAME
select dayname(date);
This gives us information about which day it is.

DAYOFMONTH
select dayofmonth(date);
This gives us information about which month it is.

DAYOFWEEK
select dayofweek(date);
This gives us information about which week of month it is.
```

MONTHNAME

This gives us the name of the month in a particular date provided

```
select monthname(date);
```

YEAR

This extract the year from the date

```
select year(date);
```

HOUR

This extract the hour from the time

```
select year(time);
```

MINUTE

This extract the minute from the time

```
select year(time);
*/

select dayname('2020-01-20');

select dayofmonth('2020-01-20');

select dayofweek('2020-01-20');

select monthname('2020-01-20');

select dayname(curdate());
```

```
select dayofmonth(curdate());

select dayofweek(curdate());

select monthname(curdate());

select jd,monthname(jd) from person;

select jd, year(jd) from person;

select jd, dayname(jd) from person;



select jt,hour(jt) from person;

select jt, minute(jt) from person;

---------- function for DATE FORMATTING  ----------

/*

Syntax:

date_format()

*/

select date_format(now() , '%d');

select date_format(now() , '%d %a');

select date_format(now() , '%d %a at %T');

select date_format(now(), '%d/%m/%y');
```

-- We can modiy it according to our need

select jdt,date_format(jdt,'%D %m at %k') from person;

---------- FUNCTION OF MATHS IN DATE ----------

/* Formate:

DATEDIFF(expr1,expr2)

This gives us the difference between two given dates

DATE_ADD(date,INTERVAL expr unit)

This gives us the return date after adding any given interval like 5 day, 1 year, 4 month

DATE_SUB(date,INTERVAL expr unit)

This gives us the return date after subtracting any given interval like 5 day, 1 year, 4 month

*/

select datediff('2023-04-16','2023-03-04');

select date_add('2023-04-16', interval 1 year);

select date_add(now(), interval 5 day);

select date_sub('2023-04-16', interval 2 year);

select date_sub(now(), interval 1 month);

---------- FUNCTION OF MATHS on TIME ----------

/* Formate:

TIMEDIFF(expr1,expr2)

This gives us the difference between two given time

*/

```
select timediff('20:00:00','19:00:50');

select timediff(curtime(),'16:00:00');

---------- DEFAULT & ON UPDATE TIMESTAMP  ----------

create table blogs
(
text varchar(200),
created_at datetime default current_timestamp,
updated_at datetime on update current_timestamp
);

desc blogs;

insert into blogs(text)
values('This is my first blog.');

select * from blogs;

set sql_safe_updates=0;

update blogs
set text='Hi myself ASHUTOSH RAI and this is my first blog.';

select * from blogs;
```

```
update blogs

set text = 'Hi myself ASHUTOSH RAI and this is my first blog from INDIA.';


set sql_safe_updates=1;


select * from blogs;


---------- Exercise6 ----------

/*

select curtime();


select curdate();


select dayname(curdate());


-- CHAR datatype is used when we need to store any string of fix length


-- FLOAT and DOUBLE can be used to store value 123.456


select date_format(curdate(), '%d:%m:%Y');


SELECT DATE_FORMAT('2023-04-22 20:00:00', '%M %D at %T');

*/


---------- RELATIONAL OPERATORS ----------
```

```
select * from employees;

-------- GREATER THAN (>)----------

select * from employees

where salary>35000;

-------- LESS THAN (<)----------

select * from employees

where salary<35000;

-------- LESS THAN EQUAL TO (<=)----------

select * from employees

where salary<=37000;

-------- GREATER THAN EQUAL TO (>=)----------

select * from employees

where salary<=37000;

-------- NOT EQUAL TO (!=)----------

select * from employees

where salary!=25000;
```

```
select * from employees
where dept!='loan';

 ---------------------------- LOGICAL OPERATORS ----------------------------

-------- AND OPERATOR ----------
-- CONDITION 1 AND CONDITION 2
-- We only get the output when both the conditions are true.

select * from employees
where salary=25000 and dept='deposit';

-------- OR OPERATOR ----------
-- CONDITION 1 OR CONDITION 2
-- We only get the output when either of the conditions are true.

select * from employees
where salary > 25000 or dept='loan';

select * from employees
where salary=20000 or salary=25000 or salary=50000;

---------------------------- USE OF IN & NOT IN ----------------------------
/*
Syntax:
select col_name/* from table_name
where col_name in (val1,val2...);
```

```
select col_name/* from table_name
where col_name not in (val1,val2...);
*/

select * from employees
where dept in ('account','loan','deposit');

select * from employees
where dept not in ('account','loan','deposit');

---------- USE OF BETWEEN ----------
/*
Syntax:
select col_name/* from table_name
where col_name between val1 and val2;
In this range both val1 and val2 are inclusive
*/

select * from employees
where salary between 20000 and 37000;

---------- APPLYING CONDITION USING CASE ----------
/*
Syntax:
select col_name1,col_name2.....,
case
```

```
when condition then output1
else
output2
end
from table_name;
*/

select fname,salary,
case
when salary>37000 then 'High Salary'
else
'Low Salary'
end
as 'Salary Status' from employees;

select concat(fname,' ',lname) as 'Employee Name', salary,
case
when salary>=42000 then 'High Salary'
when salary between 30000 and 45000 then 'Medium Salary'
else 'Low Salary'
end
as 'Salary Category' from employees;

---------- USE OF IS NULL & NOT LIKE ----------
/*
Syntax:
IS NULL:
```

```
This is used to check if we are having null values in any column or in our table
select col_name/* from table_name
where col_name is null;

NOT LIKE:
select col_name/* from table_name
where col_name not like condition;
*/

select * from person;

insert into person (jt)
value (curtime());

insert into person(jd)
value(curdate());

insert into person(jdt)
value(now());

select * from person;

-- Now checking for IS NULL
select * from person
where jd is null;

select * from person
```

```
where jt is null;

select * from person

where jdt is null;

-- Now checking for NOT LIKE

select * from employees;

select * from employees

where dept not like 'l%';

---------- Exercise7 ----------

/*

select * from employees

where salary between 30000 and 40000;

SELECT * FROM employees

WHERE fname LIKE 'R%' OR fname LIKE '%s';

select * from employees

where salary=25000 and dept='deposit';

select * from employees

where desig in ('manager','lead','associate');

select fname ,salary,

case
```

```
when salary>0 then (salary/80)
else 'Invalid Salary Input'
end
as 'sal in dollars' from employees;
*/

---------- CONSTRAIN ----------
-- These are the extra features given to any particular column in a table according to our need
-- Some most common contrains that are used in SQL are PRIMARY KEY, UNIQUE

---------- UNIQUE CONSTRAIN ----------
/*
UNIQUE CONSTRAIN is used when we need to feed unique values in any column.
There can be more than one UNIQUE CONSTRAIN in a table.
*/

create table contact
(
mob int unique
);

desc contact;

insert into contact values(12345);

select * from contact;
```

-- If we try to insert same contact details again in this column, it won't allow this because our column has unique contrain.

```
insert into contact values(12345);
```

-- Error Code: 1062. Duplicate entry '12345' for key 'contact.mob'

---------- CHECK CONSTRAIN ----------

```
/*
CHECK CONSTRAIN is used when we need to check/verify that while entering the data into column it must satisfy given function.
For entering a contact data the the contact number needs to be of a particular legth.
*/

create table contact1
(
mob int unique check (length(mob)>=10 and length(mob)<=12)
);

desc contact1;

insert into contact1 values(1234567890);

select * from contact1;
```

-- Trying to insert a number less tha 10 digits

```
insert into contact1 values(123456789);
```

-- Error Code: 3819. Check constraint 'contact1_chk_1' is violated.

```
---------- USING CHECK CONSTRAIN WITH A DEFAULT MESSAGE ----------
-- This is used to return a default message when the CHECK condition gets voilated

create table contact2
(
mob int unique
constraint Invalid_Contact_Detail check (length(mob)>=10 and length(mob)<=12)
);

insert into contact2
values(1234567890);

insert into contact2
values(123456789);
-- Error Code: 3819. Check constraint 'Invalid_Contact_Detail' is violated.

select * from contact2;

---------- ALTERING TABLE ----------
/*
ALTER is used to change the structure of any created table by adding or deleting
columns in the table.
*/

---------- ADDING COLUMN ----------
/*
```

```
Syntax:
alter table table_name
add column column_name column_datatype;
*/

alter table contact
add column City varchar(50);

desc contact;

---------- DROPPING COLUMN ----------
/*
Syntax:
alter table table_name
drop column column_name column_datatype;
*/

ALTER TABLE contact
drop column city;

---------- RENAMING COLUMN ----------
/*
This is used to rename the column or to rename the table
Syntax:
COLUMN RENAME:
alter table table_name
rename column column_name to new_column_name;
```

TABLE RENAME:

alter table table_name

rename to new_table_name

OR

rename table table_name to new_table_name;

*/

alter table contact

add column name varchar(50);

desc contact;

alter table contact

rename column name to fname;

-- Renaming table:

alter table contact

rename to mycontacts;

desc mycontacts;

---------- MODIFY COLUMN PROPERTY ALTER ----------

/*

This is to make changes in the property of the column like changing datatype or default or any other

change in property.

*/

desc contact1;

ALTER TABLE contact1

MODIFY mob VARCHAR(13);

alter table mycontacts

modify fname varchar(20) default 'Unknown';

insert into mycontacts (mob) values(1234467890);

select * from mycontacts;

---------- RELATIONSHIP ----------

/*

Types of relationship:

1) ONE TO ONE:

In ONE TO ONE relationship only one record will be avilable in the other table.

Table-1 (Employees)

!----------!----------!----------!

! emp_id ! name ! dept !

! 101 ! Raju ! IT !

! 102 ! Sham ! Finance !

Table-2 (Employee Details)

!----------!----------!----------!----------!----------!

! emp_id ! addr ! city ! phone ! title !

----------!

! 101 ! CP ! IT ! 1234567 ! Manager !

! 102 ! Bhandup ! Finance ! 9087654 ! Accountant !

----------!

If the repeation of data is only one time then it is an example of ONE TO ONE relationship.

For examplw above the epm_id of raju has just appear for one time in table 2 so it is an example of ONO TO ONE relationship.

2) ONE TO MANY:

Table-1 (Employees)

!----------!----------!----------!

! emp_id ! name ! dept !

! 101 ! Raju ! IT !

! 102 ! Sham ! Finance !

Table-3 (Employee Task)

!----------!----------!----------!

! emp_id ! task_no ! task_detail !

!----------!

! 101 ! TS-1 ! Opening account for Ram !

! 102 ! TS-2 ! closing account for Neru!

! 101 ! TS-3 ! Loan senction !

----------!

Here in table 3 we can see that emp_id 101 has been assinged 2 task TS-1 and TS-3 so two records

can be found for 101 in table 3 this is an example of ONE TO MANY relationship.

3) MANY TO MANY:

Suppose 1 book has 2 author and both the author have writeen many other books also (individually and in colabration as well)

this type of relationship is MANY TO MANY relationship.

*/

---------- FOREIGN KEY RELATIONSHIP ----------

/*

When a key or we can say column is present in more than one table to use as an reference than that

key is known as FOREIGN KEY.

*/

```
create database stores;

use stores;

create table customers
(
```

```
 cust_id int primary key auto_increment,
 name varchar(50) not null,
 email varchar(50) not null
 );

desc customers;

create table orders
(
ord_id int auto_increment primary key,
date date,
amount decimal (8,2),
cust_id int ,
foreign key (cust_id) references customers(cust_id)
);

desc orders;

insert into customers(name,email)
values ('Raju','raju@email'),('Sham','sham@email');

insert into orders(date,amount,cust_id)
values(curdate(),105.38,1);

select * from orders;

insert into orders(date,amount,cust_id)
```

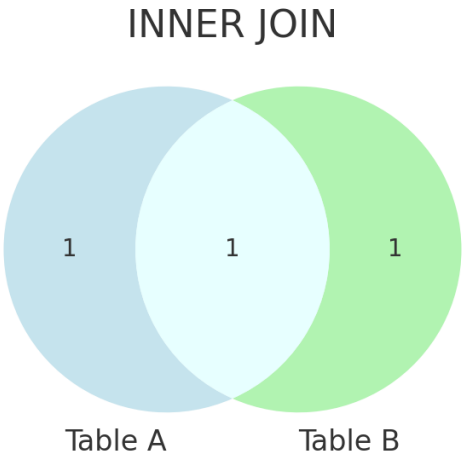
```
values (curdate(),500.38,1),(curdate(),503.38,1),(curdate(),503.38,1);

select * from orders;

insert into orders(date,amount,cust_id)
values(curdate(),105.38,10);
-- Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`stores`.`orders`, CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`cust_id`) REFERENCES `customers` (`cust_id`))
-- This fails to add data because cust_id is a foreign key and we are
-- adding cust_id 10 here which is not present in cust_id of customers table.

 ---------------------------- WHAT ARE JOINS ----------------------------
 /*
  JOIN operation is used to combine rows from two or more table based on the related column between them.
 */

 insert into customers(name,email)
 values ('Baburao','babu@email'),('Paul','paul@email'),('Alex','alex@email');

 select * from customers;

 insert into orders(date,amount,cust_id)
 values (curdate(),1234.70,4),(curdate(),379.49,3),(curdate(),234.20,5),(curdate(),1250.25,3);

 select * from orders;
```

GitHub: https://github.com/ashutoshrai70

SQL Joins Explained with Venn Diagrams

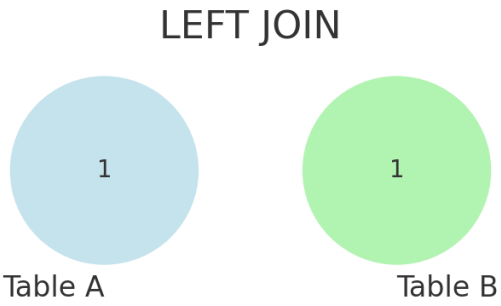
INNER JOIN

An INNER JOIN returns records that have matching values in both tables.



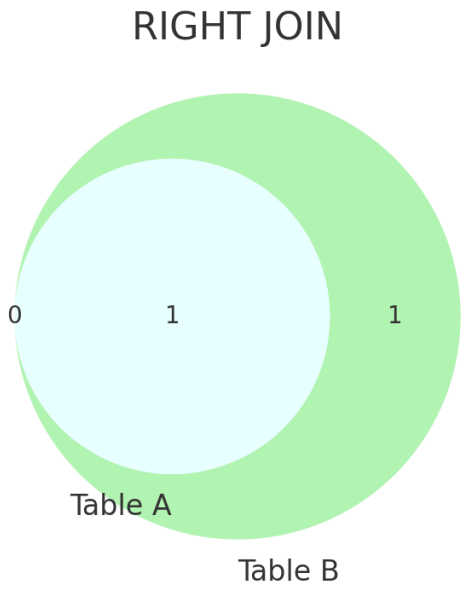
LEFT JOIN

A LEFT JOIN returns all records from the left table, and the matched records from the right table.



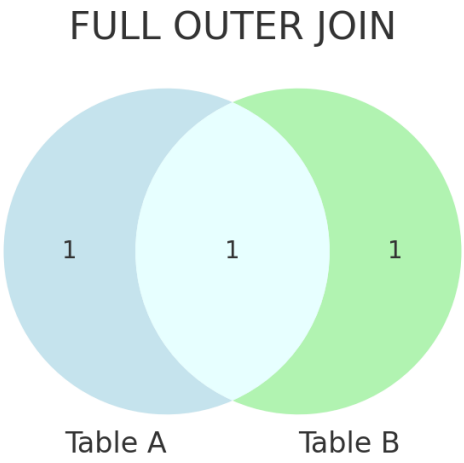
RIGHT JOIN

A RIGHT JOIN returns all records from the right table, and the matched records from the left table.



FULL OUTER JOIN

A FULL OUTER JOIN returns all records when there is a match in either left or right table.



---------- TYPES OF JOINS ----------

/*

Types of joins:

CROSS JOIN

INNER JOIN

LEFT JOIN

RIGHT JOIN

*/

---------- CROSS JOIN ----------

-- Every row of one table is combined withevery row of another table.

```
select * from customers,orders;
```

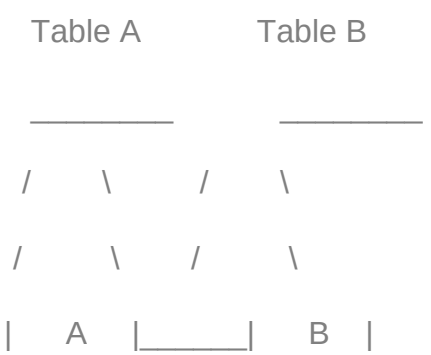
-- It is not very useful because it shows every possible combination and it is a kind of repetative data.

---------- INNER JOIN ----------

/*

I returns the rows where there is a match between the specific columns

in both left(or we can say first) and right (second) tabels.



```
 \        /       \        /
  \_______/         \_______/

  Inner Join: The overlapping part between A and B
*/

select * from customers
inner join orders
on customers.cust_id = orders.cust_id;

-- Here we can see that we did not get any data for Sham
-- because Sham has not placed any order yet so it's cust_id is not present in orders table.

select name, sum(amount) as 'Total order amount' from customers
inner join orders
on customers.cust_id =  orders.cust_id
group by customers.cust_id;

-- customers.cust_id; here we can use either of the table name orders.cust_id will also give the same result.
-- from customers here we can use either of the table name from orders will also give the same result.

-- OR
select name, sum(amount) as 'Total order amount' from customers
inner join orders
on customers.cust_id =  orders.cust_id
```

```
group by name;


---------- LEFT JOIN ----------
/*
 While writing the query the table name that is used firstly is considerd as left table
 and the other one is considerd as right table.

 In LEFT JOIN we get every information from the left and common informations from the right table.
 */

 select * from customers
 left join orders
 on customers.cust_id = orders.cust_id;

 select name,sum(ord_id) as 'Total Order' from customers
 left join orders
 on customers.cust_id = orders.cust_id
 group by name;

 -- OR
 select name,ifnull(sum(ord_id),'No Purchase') as 'Total Order' from customers
 left join orders
 on customers.cust_id = orders.cust_id
 group by name;

 -- OR
```

```
select name,ifnull(sum(ord_id),'No Purchase') as 'Total Order' from customers
left join orders
on customers.cust_id = orders.cust_id
group by customers.CUST_ID;

---------- RIGHT JOIN ----------
/*
While writing the query the table name that is used firstly is considerd as left table
and the other one is considerd as right table.

In RIGHT JOIN we get every information from the right table and common informations from the left table.
*/

 select * from orders
right join customers
on  orders.cust_id= customers.cust_id;

 select name,sum(ord_id) as 'Total Order' from customers
right join orders
on customers.cust_id = orders.cust_id
group by name;

---------- ON DELETE CASCADE ----------
/*
Suppose we want to delete any customers data from our table customers
we wont be able to do it becase that table (customers) is linked with the other table 'orders' as a
```

```
parent table
for solving this problem we use ON DELETE CASCADE while creatin the table(orders)

By doing this if we delete any row from parent table(customers) then its data/row
gets automaticly deleted from other table(orders).
*/

drop table orders;

create table orders
(
ord_id int primary key auto_increment,
date date,
amount decimal(5,2),
cust_id int,
foreign key(cust_id) references customers(cust_id) on delete cascade
);

insert into orders(date,amount,cust_id)
values (curdate(),100.50,1),(curdate(),500.40,2),(curdate(),300.30,1);

select * from orders;

set sql_safe_updates=0;

delete from customers
where name='Raju';
```

```
set sql_safe_updates=1;

select * from customers;

select * from orders;

-- We can see that just by deleting row of Raju from table customers
-- data of raju from both the tables deleted.

---------- Exercise8 ----------
/*
create table Authors
(
author_id int primary key auto_increment,
author_name varchar (50) not null
);

create table Books
(
book_id int primary key auto_increment,
title varchar(50),
ratings int check (ratings between 1 and 5),
au_id int,
foreign key(au_id) references Authors(author_id) on delete cascade
);
```

```
insert into  authors(author_name)
values ('Raju'),('Sham'),('Baburao'),('Paul');

select * from authors;

insert into books(title,ratings,au_id)
values('Story of Raju',5,1),('Story of Baburao',4,3),('Raju - The Great Man',2,1),('Love story by Sham',1,2);

select * from books;

select title,ratings,au_id from authors
inner join books
on authors.author_id=books.au_id;

select author_name,title,ratings from authors
left join books
on authors.author_id = books.au_id;

select author_name,ifnull(title,'Not Found'),ifnull(ratings,0) from authors
left join books
on  authors.author_id=books.au_id;

select author_name,ratings,
case
when ratings>=3 then 'Good'
else 'Average'
```

```
end
as 'Remark' from books
inner join authors
on authors.author_id = books.au_id;
*/

 ---------- USE CASE OF MANY TO MANY RELATIONSHIP ----------
/*
---------- CONCEPT OF BRIDGE TABLE/JUNCTION TABLE ----------
BRIDGE TABLE are those table which is used join or we can say used to
stablish connection between any two pre existing tables.

we will create a table with name 'students' with columns (id,student_name) in it.
we will create another table with name 'course' with columns (id,course_name,fees) in it.
Now we will create a third table with 'stdent_course' with columns (student_id,course_id) in it.
Now this third table will be used as BRIDGE TABLE to join the other two tabels.
*/

create database Institute;

use institute;

create table students
(
id int primary key auto_increment,
student_name varchar(50)
);
```

```
create table courses
(
id int primary key auto_increment,
course_name varchar(100) not null,
fees int not null
);

create table student_course
(
student_id int,
foreign key(student_id) references students(id) on delete cascade,
course_id int,
foreign key(course_id) references courses(id) on delete cascade
);

insert into students(student_name)
values ('Raju'),('Sham'),('Paul'),('Alex');

select * from students;

insert into courses(id,course_name,fees)
values (101,'PD',3000);

insert into courses(course_name,fees)
values ('Java',5000),('SQL',4000),('Python',6000),('Linux',10000);
```

```
select * from courses;

insert into student_course
values(1,101),(1,102),(2,105),(1,105),(3,103),(2,102),(4,104);

select * from student_course;

select student_name , course_name from student_course
inner join students
on student_course.student_id = students.id
inner join courses
on courses.id = student_course.course_id;

-- OR

select student_name , course_name from student_course
join students
on student_course.student_id = students.id
join courses
on courses.id = student_course.course_id;

---------- Exercise9 ----------
/*
select course_name,count(course_id) as 'Number of students' from courses
inner join student_course
on student_course.course_id = courses.id
group by id;
```

```
-- OR

select course_name,count(course_id) from students
inner join student_course
on student_course.student_id = students.id
inner join courses
on student_course.course_id = courses.id
group by courses.id;

-- OR

select course_name,count(student_name) from students
inner join student_course
on student_course.student_id = students.id
inner join courses
on student_course.course_id = courses.id
group by course_name;

select student_name,count(student_id) as 'Number of course' from student_course
inner join students
on student_course.student_id = students.id
group by student_course.student_id;

-- OR

select student_name,count(course_name) from student_course
```

```
inner join students
on student_course.student_id=students.id
inner join courses
on student_course.course_id=courses.id
group by student_name;

select student_name,sum(fees) as 'Total fees paid' from student_course
inner join students
on student_course.student_id=students.id
inner join courses
on student_course.course_id=courses.id
group by student_name;

*/

---------------------------- VIRTUAL TABLES ----------------------------
/*
------------------ VIWE --------------------

VIEW:
VIEW is a virtual table which is used when we want to use our bridge table or any table
many times
By using VIEW we can bypass the process of writing same query again and again.
*/

create view Stud_info as
select student_name,course_name,fees from student_course
```

```
inner join students
on student_course.student_id = students.id
inner join courses
on student_course.course_id = courses.id;

show tables;

select * from stud_info;

select student_name,fees from stud_info;

select course_name, sum(fees) as total_fees_collected from stud_info
group by course_name
order by course_name;

select * from stud_info
where student_name='Raju';

-------- DROP VIEW ---------
drop view stud_info;

show tables;

---------- HAVING & ROLLUP WITH CLAUSE ----------

---------- HAVING Caluse ----------
create view inst_info as
```

```
select student_name,course_name,fees from student_course
inner join students
on student_course.student_id = students.id
inner join courses
on student_course.course_id = courses.id;

-- For now treat inst_info as a normal table not as a virtual table.

select * from inst_info;

select student_name,sum(fees) from inst_info
group by student_name;

-- We want to see who have paid more than 10000

select student_name,sum(fees) from inst_info
where sum(fees)>=10000
group by student_name;

-- Here where function will not work because where dose not work with group by funtion like this.

select student_name,sum(fees) from inst_info
group by student_name
having sum(fees)>=10000;

---------- GROUP BY ROLLUP ----------
-- ROLLUP always works with GRUP BY function
```

-- ROLLUP can be use with sum(),avg(),count() and many more

```
select student_name, sum(fees) from inst_info
group by student_name
with rollup;

select ifnull(student_name,'Total Amount'),sum(fees) from inst_info
group by student_name
with rollup;

select ifnull(course_name,'Total Students'),count(course_name) from inst_info
group by course_name
with rollup;
```

---------- STORED ROUTINE ----------

/*

It is an SQL statement or a set of SQL statement that can be stored on database server

which can be call number of times.

TYPES OF STORED ROUTINES:

1) STORED Procedure

2) User defined functions

*/

---------- STORED Procedure ----------

/*

STORED Procedurese are routines that contain a series of SQL statements and procedural logic.

Often used for performing actions like data modification, transaction control,

and executing sequences of statements.

*/

```
use bank_data;
-- (;) This is called as delimiter

-- Temporary changing the delimiter
DELIMITER $$

CREATE PROCEDURE p_name()
BEGIN
    SELECT * FROM employees
     LIMIT 5; -- If we don't change delimiter above, MySQL would consider the semicolon as the end
of the query
END $$

DELIMITER ; -- Resetting delimiter back to semicolon

-- To drop the procedure, you can use the following command separately:
DROP PROCEDURE p_name;

delimiter $$
create procedure emp_info()
begin
```

```
select * from employees;
-- if we want we can add multiple queries here like
-- select fname from employees;
end $$

delimiter ;

-- If we are using the same database we can directly call the STORED PROCEDURE otherwise syntax will be
-- Syntax:
-- call database_name.procedure_name;

call bank_data.emp_info();

-- OR

call emp_info();

---------- Argument Passing in STORED Procedure ----------

delimiter $$

create procedure get_empid(in p_fname varchar(50)) -- in is used here to pass argument in variable p_fname
begin
select emp_id from employees
where fname=p_fname;
```

```
end $$

delimiter ;

call bank_data.get_empid('Raju');

-- OR

call get_empid('Raju');

---------- Returning output in a variable using STORED Procedure ----------

DELIMITER $$

CREATE PROCEDURE get_dept_sum(IN p_dept VARCHAR(20), OUT p_sum DECIMAL(10,2))
BEGIN
    -- Use SELECT INTO only if there is a single row to be returned
    SELECT SUM(salary) INTO p_sum
    FROM employees
    WHERE dept = p_dept;
END $$

DELIMITER ;

-- Set the variable to store the output
SET @emp_s = 0;
```

```
-- Call the procedure
CALL get_dept_sum("Loan", @emp_s);

-- Check the result
SELECT @emp_s;

---------- USER DEFINED FUNCTION ----------
-- It is very much similar to PROCEDURE

delimiter $$
 create function max_salary_emp_name() returns varchar(50)
 begin
 declare s_max int;
 declare s_name varchar(50);
 select max(salary) into s_max from employees;
 select fname into s_name from employees
 where s_max= salary;
 return s_name;
 end $$
delimiter ;
-- Error Code: 1418. This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

-- This dosen't work because we need to put 'DETERMINISTIC NO SQL READS SQL DATA' below create funtion line.
```

```
delimiter $$
 create function max_salary_emp_name() returns varchar(50)
 DETERMINISTIC NO SQL READS SQL DATA
 begin
 declare s_max int;
 declare s_name varchar(50);
 select max(salary) into s_max from employees;
 select fname into s_name from employees
 where s_max= salary;
 return s_name;
 end $$
delimiter ;

-- Calling function
select bank_data.max_salary_emp_name();

-- OR if you are already using same database just call function by it name.
-- For function we dont use CALL we usse SELECT because it directly returns us a value.
select max_salary_emp_name();

--------------------------- WINDOW FUNCTIONS ----------------------------
/*

WINDOW FUNCTIONS are widely used for data analysis purpose.
Windows functions, also known as analytic functions allow
us to perform calculations across a set of rows related to
current row.
```

```
Defined by an OVER() clause.
*/

select * from employees;
-- We want to calculate the total sum of salary
select sum(salary) from employees;

-- Doing it with windows function
select emp_id,fname,
sum(salary) over(order by emp_id) as "Total salary" from employees;

-- Finding sum of salary for every department
select emp_id,dept,
sum(salary) over(order by emp_id) as 'Total salary' from employees;

---------- PARTITION BY ----------
-- Grouping by department
select emp_id,dept,
sum(salary) over(partition by emp_id) as 'Total salary'
from employees;
-- Here sorting is automaticaly done while using partition on basis of emp_id

-- using partition on department column (dept) now it will do sorting on dept by itself
select emp_id,dept,sum(salary)
over(partition by dept) as 'Total Department Salary'
from employees;
```

```
-- using partition on department column (dept) to get maximum salary of each department
select emp_id, dept,max(salary)
over(partition by dept) as 'Maximum Salary'
from employees;

/*
---------- PRE DEFINED WINDOW FUNCTIONS ----------

ROW_NUMBER()
RANK()
DENSE_RANK()
LAG()
LEAD()

*/

---------- ROW_NUMBER() ----------

select row_number() over() as 'S.NO.',
emp_id,
dept,
salary
from employees;

select row_number() over(order by salary) as 'Row Number',
emp_id,
```

```
dept,
salary
from employees;

---------- RANK() ----------

select rank() over(order by salary desc) as 'Rank',
emp_id,fname,dept,salary from employees;

-- OR

select *, rank() over(order by salary desc) as 'Rank'
 from employees;

 ---------- DENSE_RANK() ----------

 select dense_rank() over(order by salary desc) as 'Rank',
emp_id,
fname,
dept,salary
from employees;

-- OR

select *, dense_rank() over(order by salary desc) as 'Rank'
 from employees;
```

```
---------- LAG() ----------
-- LAG function is used over a column and it reflects the previous value in front of current value.

select emp_id,fname,dept,salary,
lag(salary) over() as 'Previous Salary'
from employees;

-- Using LAG function to get the difference between the salary

select emp_id,fname,dept,salary,
salary-lag(salary) over(order by salary desc) as 'Salary Difference'
from employees;

---------- LEAD() ----------
-- LEAD function is used over a column and it reflects the next value in front of current value.

select emp_id,fname,dept,salary,
lead(salary) over() as 'Lead Salary'
from employees;

-- Using LEAD function to get the difference between the salary

select emp_id,fname,dept,salary,
salary-lead(salary) over(order by salary desc) as 'Salary Difference'
from employees;
```

---------- END ----------